Construction et déploiement de systèmes d' information répartis ouverts et adaptables au moyen d'agents mobiles et de composants

Jean-Paul ARCANGELI[*] — **Sébastien LERICHE**[*] — **Marc PANTEL**[**]

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

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

RÉSUMÉ. *Les applications réparties à l'échelle du réseau mondial, et en particulier les systèmes d'information, sont complexes par leur nature décentralisée et ouverte. Leur construction doit prendre en compte les évolutions fréquentes de la structure du réseau ainsi que la volatilité des ressources et des services (localisation, disponibilité…). Les approches classiques pour le développement ne permettent pas de répondre de manière satisfaisante à ces contraintes. Nous proposons dans ce papier une approche à base d'agents (autonomie, mobilité, adaptabilité) et de composants logiciels dont le déploiement, l'adaptation dynamique et la localisation sont assurés par ces agents. Après avoir étudié les caractéristiques de quelques applications réelles, nous discutons des technologies logicielles, puis nous proposons un framework pour leur mise en place. Enfin, nous présentons quelques éléments d'implantation dans ce framework des applications précédemment étudiées.*

ABSTRACT.

MOTS-CLÉS : *agents mobiles, composants, déploiement, adaptation, architecture logicielle, systèmes pair à pair, répartition à grande échelle, grille, systèmes d'information*

KEYWORDS:

1. Introduction

La démocratisation des moyens informatiques permet à la plupart des particuliers de disposer maintenant d'un ordinateur et d'un accès à un réseau. Le développement des réseaux à grande échelle permet de relier non seulement les différents sites académiques, industriels et institutionnels, mais également la grande majorité des particu-

1re soumission à *Ł'Objet OCM-SI*, le 26 janvier 2005.

liers. Le réseau mondial ainsi obtenu permet de disposer d'une quantité d'information extrêmement importante et d'offrir un grand nombre de services. La notion de grille de calcul ou de stockage assimile celui-ci à un réseau de distribution semblable à la distribution d'eau ou d'électricité : de nombreux fournisseurs interconnectés offrent des services ou des données de manière relativement uniforme à l'ensemble des utilisateurs connectés au réseau. Le réseau mondial peut donc être considéré comme un système d'information très riche mais peu structuré. Les services les plus utilisés sont d'ailleurs les services de recherche d'information tels Google.

Ce réseau possède par contre un défaut majeur : une très forte variation de la qualité de service. Son utilisation n'est pas du tout régulière mais connaît des pics de sur-utilisation et de sous-utilisation qu'il est parfois difficile de prévoir et de compenser. De plus, un site peut à tout instant sans contrainte décider de quitter le réseau ou de retirer certaines ressources ou certains services. De plus, le nombre d'éléments intervenant dans son architecture, même avec des taux de pannes extrêmement faibles et une redondance élevée, fait qu'en permanence de nombreux éléments matériels ou logiciels sont en panne. Il n'est donc pas possible de connaître précisement à chaque instant la structure du réseau et l'ensemble des informations et des services disponibles. Il est alors qualifié de réseau ouvert dans le sens où il n'est possible d'en disposer que d'une connaissance partielle ou ancienne.

Différentes approches sont possibles pour rendre ce réseau global aussi utilisable que possible. D'une part, les couches réseau et système visent à masquer ces problèmes vis-à-vis de l'application. Cette approche se révèle actuellement trop coûteuse dans un contexte de qualité de service fortement variable. D'autre part, les *middlewares*[1] sont des intermédiaires entre l'application et les couches réseau et système qui offrent à l'application un certain niveau de contrôle et des possibilités d'exploitation directe de la structure réelle de ces couches. En particulier, ils permettent à l'application de découvrir et parcourir la dtructure du réseau pour rechercher les ressources disponibles (données ou services).

De nombreux modèles d'exécution sont proposés par les différents *middlewares*. Le plus courant est le modèle client-serveur (appels de procédures à distance, objets répartis). Cette approche repose sur l'hypothèse d'une qualité de service relativement bonne qui permet à un objet distant de rester accessibles pendant une durée importante. Cette hypothèse est réaliste dans le cadre d'un réseau local ou d'un réseau interne à une seule entité administrative. Dans le cadre de réseaux à grande distance faisant intervenir de nombreuses entités administratives (les particuliers par exemple), le coût du maintien des propriétés nécessaires à ce modèle est trop élevé. Actuellement, l'approche « pair à pair » vise à compenser les difficultés liées à la centralisation.

Dans ce travail, nous explorons l'intérêt d'associer les technologies d'agent (autonomie, mobilité, adaptabilité) et de composant logiciel pour la construction, le dé-

1. Un *middleware* (ou intergiciel) 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

ploiement et la maintenance d'applications réparties ouvertes et adaptables dans un contexte « pair à pair ». Notre proposition est de nature « génie logiciel » : son objectif est de permettre une meilleure prise en compte des spécificités de l'application afin d'exploiter au mieux les différentes couches, et en particulier la structure dynamique du réseau.

Dans la section suivante, nous décrivons quelques applications réelles qui illustrent la problématique et les besoins associés. Dans la section 3, nous étudions et nous proposons différentes technologies logicielles utiles pour leur développement. Nous présentons ensuite, en section 4, une méthodologie et un *framework* à base d'agents mobiles adaptables et de composants logiciels pour la construction et le déploiement de systèmes d'information répartis et ouverts. Enfin, nous concluons en section 5 et nous évoquons quelques perspectives à ce travail.

2. Exemples

Pour mieux introduire les besoins qui justifient l'approche présentée dans cette communication, nous allons détailler plusieurs exemples réels allant de la simple mutualisation de ressources à une possible mise en œuvre informatique du futur dossier médical personnalisé.

2.1. *Exemple 1 : mutualisation de ressources*

Considérons un logiciel qui permet à une communauté d'utilisateurs de fournir, de rechercher et d'utiliser des ressources réparties sur un réseau hétérogène de grande taille, par exemple à l'échelle du réseau mondial (par exemple, dans un contexte pair à pair, citons Napster, FreeNet, Gnutella, Kazaa, BitTorrent, voir sous-section 3.1.1). Ces ressources peuvent être des fichiers de type quelconque (son, image, texte…) ou bien des services (*web service*, base de données…). Chaque utilisateur connaît d'autres membres de la communauté et dispose de certaines connaissances sur les ressources qu'ils fournissent. Un utilisateur peut à la fois rendre accessibles des ressources dont il est propriétaire, ainsi que rechercher puis utiliser des ressources parmi les ressources disponibles dans la communauté. La recherche s'appuie sur les connaissances de l'utilisateur, et ces connaissances évoluent en fonction des résultats obtenus.

Le logiciel étant déployé dans un contexte de répartition à grande échelle, il doit supporter les pannes et les déconnexions de certains sites ainsi que les évolutions des ressources (nouvelle, mise à jour, supprimée). En complément, le logiciel doit être suffisamment flexible pour permettre l'utilisation de différents protocoles adaptés à chaque type de recherche.

Un scénario d'utilisation pourrait être le suivant :

On s'intéresse à cinq sites S_i d'une communauté, proposant chacun des ressources R_j, sauf S_4. Les liens de connaissances initiaux sont représentés sur la figure 1. Un

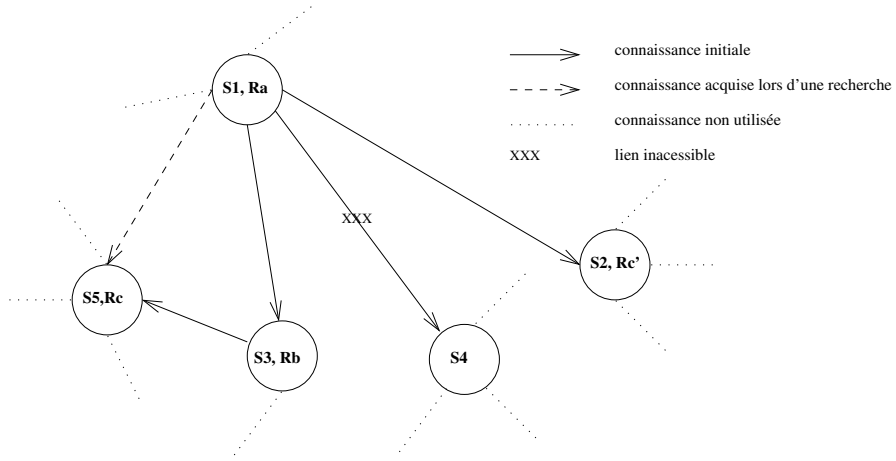

Figure 1. *Scénario d'utilisation du logiciel de mutualisation de ressources*

utilisateur sur le site S_1 souhaite utiliser une ressource R_c. Celle-ci a été exploitée antérieurement sur S_2, donc la première tentative de recherche se fait sur ce site. Mais la ressource (notée R'_c) a évolué entre temps (mise à jour, suppression...) et s'avère inexploitable. Le système poursuit alors la recherche sur les autres sites connus. S_4 ne répond pas (panne, déconnexion ou saturation...). Le site S_3 ne contient pas la ressource recherchée, mais connaît deux sites initialement inconnus de S_1. Sur le site suivant, S_5, se trouve cette fois la ressource. Sur S_1, le logiciel ajoute le lien de connaissance vers S_5 (cf. schémas). L'exploitation de Rc peut être par exemple un téléchargement de fichier de S_5 vers S_1, ou bien l'exécution d'une requête sur une base de donnée du site S_5.

2.2. *Exemple 2 : déploiement automatique de logiciels*

Nous considérons ici une application de déploiement de modules logiciels (applications entières ou parties d'application à des fins de mise à jour). Le déploiement de logiciels [DEC04] recouvre les activités de distribution, d'installation et de mise à jour d'un logiciel, ainsi que sa mise en production (configuration, activation). Il s'agit d'un processus transactionnel complexe qui suppose la localisation, l'authentification et l'agrément des différents partis (à l'extrême, une mise à jour peut être effectuée « à chaud » c'est-à-dire sans arrêter le fonctionnement du logiciel). Son automatisation est de nature à réduire les délais de diffusion des logiciels ainsi que l'impact du nombre de machines concernées. Cette activité est maintenant disponible dans de nombreux systèmes d'exploitation grand public (distribution Linux Debian, logiciels Microsoft, plugins Adobe Acrobat, Mozilla, Eclipse...).

Le déploiement automatique implique différents acteurs reliés par un réseau de communication :

– des producteurs de logiciel qui proposent les codes et leur documentation,

– des utilisateurs chez qui les modules sont installés,

– des distributeurs dont le rôle est d'organiser et d'assurer le service de déploiement.

Producteurs et utilisateurs n'interagissent pas directement, mais par l'intermédiaire des distributeurs suivant deux modes distincts, le flot de données étant orienté des producteurs vers les utilisateurs. En mode *push*, le distributeur (éventuellement, en amont, le producteur) est l'initiateur de l'interaction avec l'utilisateur. Par exemple, il peut s'agir d'une mise à jour générale de tous les postes client au sein d'une entreprise ou d'un groupe d'utilisateurs. Le distributeur doit d'abord identifier les utilisateurs cibles (éventuellement abonnés au service de distribution), ce qui peut demander une recherche sur le réseau et une analyse de l'existant chez les utilisateurs. En mode *pull*, c'est l'utilisateur (ou son système) qui est à l'origine de l'interaction et qui demande l'installation d'un module ; dans ce cas, c'est l'utilisateur qui doit trouver un distributeur et pouvoir consulter un catalogue qui décrit les modules disponibles.

De la même manière, un distributeur peut être ou non l'initiateur de l'interaction avec le producteur. On peut considérer que les distributeurs indexent et distribuent les modules logiciels et qu'ils jouent à la fois le rôle de client pour les producteurs et le rôle de serveur pour les utilisateurs. Si l'on considère le service de distribution, producteurs et utilisateurs sont symétriques.

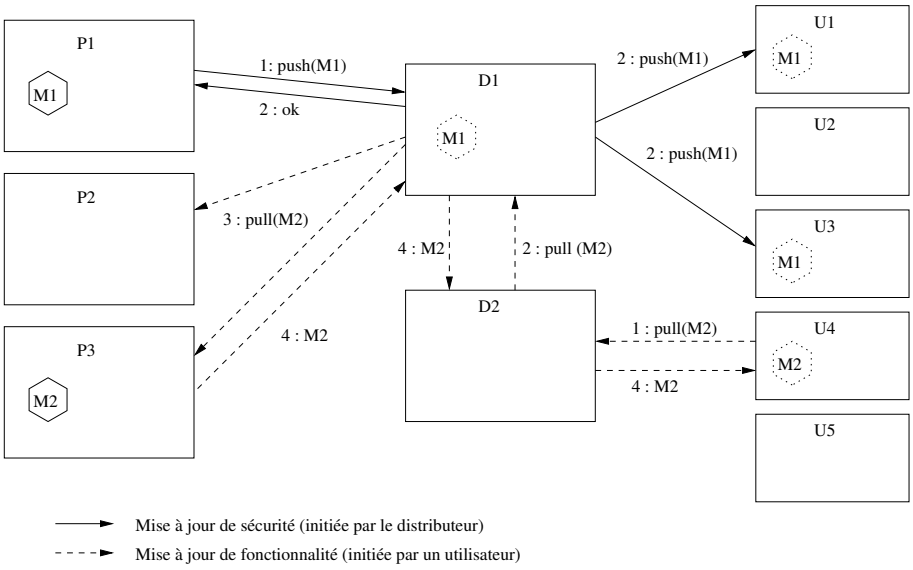

Figure 2. *Scénarios d'utilisation du système de déploiement*

Illustrons le fonctionnement de l'application par deux scénarios impliquant 3 producteurs, 2 distributeurs et 5 utilisateurs (cf. figure 2) :

– P_1 produit un module M_1 (par exemple, une mise à jour de sécurité) et le fournit à un distributeur D_1 (1). D_1 enregistre le module M_1, puis retourne un acquittement à P_1 et distribue M_1 aux utilisateurs qu'il connaît (2).

– U_4 recherche un module M_2 (nouvelle fonctionnalité, mise à jour…). Il s'adresse à un distributeur D_2 (1) qui ne dispose pas du module demandé. D_2 peut s'adresser à différents producteurs et à d'autres distributeurs, en l'occurrence il s'adresse à D_1 (2). D_1 s'adresse à P_2 et à P_3 (3). P_3 dispose du module M_2 et le fournit au demandeur (4). D_1 fournit le module à D_2 (5) qui l'installe chez l'utilisateur U_4. Éventuellement, D_2 pourrait ajouter M_2 à son catalogue.

Dans les deux cas, on peut imaginer que lors de l'installation du module logiciel, on détecte qu'un module M_3 est nécessaire au fonctionnement. Si M_3 n'est pas présent chez l'utilisateur, il faut entreprendre sa distribution chez le client et ainsi de suite jusqu'à ce que l'ensemble des dépendances soient satisfaites.

2.3. *Exemple 3 : le dossier médical personnalisé*

Nous considérons ici les éléments nécessaires au traitement informatique d'un «*dossier médical personnalisé*[2]».

Plusieurs acteurs interviennent dans ce système :

– le patient qui doit pouvoir être traité où qu'il se trouve en France et dans les pays étrangers conventionnés,

– le médecin généraliste qui peut être le référent du patient mais aussi un autre médecin sur un lieu où peut se trouver temporairement le patient,

– le médecin spécialiste auquel le patient peut être adressé par un généraliste, un autre spécialiste ou consulté spontanément par le patient,

– les auxiliaires de santé (pharmacien(ne), infirmier(ère)…),

– les équipements utilisés par certains spécialistes pour examiner le patient (radiologie, échographie…),

– les centres de soins regroupant généralistes, spécialistes, auxiliaires et équipements (hôpitaux, cliniques…),

– des chercheurs menant des études épidémiologiques,

– les prestataires de services informatiques (stockage, exploitation des données…) pour les différents acteurs,

– les services institutionnels de gestion de la santé (CPAM…).

2. Proposition du Ministère des Solidarités, de la Santé et de la Famille

Notons que le patient, les généralistes et certains auxiliaires de santé sont physiquement mobiles (par exemple, dans le cadre de consultations à domicile).

2.3.1. *Fonctionnement actuel*

Actuellement, chaque personnel de santé possède un dossier par patient sans aucune mise en commun. Les résultats des examens sont transmis au prescripteur sous une forme papier (conclusions, clichés, ...) extrêmement pauvre par rapport aux possibilités exploratoires des instruments actuels.

2.3.2. *Fonctionnement futur*

L'objectif initial du dossier médical personnalisé est de permettre un accès à toutes les informations concernant un même patient afin d'augmenter l'efficacité du système de soins. La proposition actuelle consiste à centraliser ces informations chez un prestataire de stockage sécurisé. Cette approche est nécessairement coûteuse car elle requiert une qualité de service importante au niveau du prestataire (disponibilité des données, temps d'accès...). Nous proposons une architecture alternative plus flexible et plus ouverte qui exploite un stockage réparti des données chez différents prestataires associés aux différents intervenants dans le traitement d'un même patient au cours de sa vie.

Pour des raisons de sécurité, chaque patient et chaque personnel de santé disposera d'une carte électronique d'authentification (par exemple, Vitale 2).

Chaque personnel de santé disposera d'un terminal informatique portable avec différentes formes de connection réseau mais il est nécessaire de prévoir également un fonctionnement en mode déconnecté.

Considérons un patient sur son lieu de résidence qui consulte son médecin référent pour une pathologie classique. Le patient et le médecin s'authentifient pour autoriser l'accès aux éléments du dossier médical. Le médecin prescrit des médicaments. Cette prescription est validée par rapport à toutes les informations disponibles au sujet du patient. La prescription est ensuite ajoutée au dossier médical du patient. Le patient souhaite ensuite se rendre chez un pharmacien. L'application indique au patient quelles sont les pharmacies qui disposent des médicaments prescrits et réservent les médicaments sur demande du patient. Si le patient se rend directement chez un pharmacien qui ne possède pas les médicaments prescrits, après authentification, l'application consulte le dossier médical et les références du prescripteur et propose une prescription alternative adaptée au patient. En cas de nécessité, l'application demande une confirmation au prescripteur si celui-ci est joignable.

Considérons un patient sur son lieu de vacances qui consulte un généraliste local qui n'est pas son référent. Après authentification, celui-ci peut accéder à la partie du dossier médical du patient qui concerne la pathologie présentée si le patient donne son accord. Pour obtenir des détails supplémentaires, il prescrit un examen radiologique, sa prescription contenant tous les éléments nécessaires à l'exploitation optimale des instruments d'examen. Après authentification, l'instrument peut également accéder au

contenu du dossier pour se configurer en fonction des résultats d'examens précédents qu'aurait subi le patient. Le volume de données généré par l'examen est très important. Seuls des éléments synthétiques décrivant d'une part l'ensemble de l'examen et d'autre part les éléments précis correspondant à la prescription sont inscrits dans le dossier. Après traitement, le patient retourne sur son lieu de résidence et suite à une évolution de la pathologie consulte son généraliste référent qui souhaite alors approfondir l'examen. Il prescrit une exploitation plus sophistiquée des données de l'examen précédent qui ont été conservées par le cabinet concerné. Après authentification, l'application accède aux résultats, produit de nouveaux résultats synthétiques qui sont ajoutés au dossier. Ce second examen n'a demandé aucune action du spécialiste, il s'agissait uniquement d'accéder de nouveau aux données de l'examen initial.

2.4. *Caractéristiques applicatives et implications sur le développement*

2.4.1. *Echelle et ouverture*

Les architectures des différentes applications présentées précédemment s'articulent autour de plusieurs unités physiques ou logiques autonomes, concurrentes et réparties, dont le rôle est de fournir ou d'exploiter des ressources ou des services. Ces unités sont administrées indépendamment les unes des autres. Au sein des applications, le contrôle est décentralisé et distribué à travers les différentes unités.

En terme d'échelle, le nombre d'unités peut-être très grand et leur couplage faible (interconnexions via un réseau filaire de grande taille ou sans fil, à faible qualité de service). Aussi, le nombre et les volumes des interactions entre les unités influent fortement sur le fonctionnement et la performance.

Une autre conséquence de l'échelle est l'ouverture des systèmes : outre les problèmes de pannes, les unités peuvent intégrer ou quitter le système dynamiquement, voire se déplacer sur le réseau, sans contrôle global et sans qu'il ne soit possible de prévoir en totalité les évolutions.

2.4.2. *Synthèse des principaux besoins applicatifs*

Nous essayons ici d'identifier les principaux besoins pour l'exécution de ces applications qui doivent être pris en compte dès la conception. On peut les regrouper en quatre catégories :

– Recherche d'information : localisation d'entités sur le réseau, découverte de nouveaux services, gestion des connaissances des différents acteurs, fouille locale sur les sites…

– Traitement des données : services externes d'exploitation des données (tiers par rapport à l'application), traitements locaux et distants par rapport à un utilisateur (un compromis lié au coût de transfert des données et de déploiement de service est nécessaire au niveau du choix du lieu de traitement)…

– Sûreté de fonctionnement : qualité de service, *i.e.* tolérance aux pannes, mécanismes de stockage réparti (disponibilité, redondance…), gestion des volumes et des flux de données, gestion des différentes formes d'interface réseau et système, gestion du mode déconnecté…

– Sécurité : services d'authentification des entités et de leurs représentants, communications sécurisées…

2.4.3. *Vers la recherche et la personnalisation des services*

Lors de la construction de systèmes ouverts répartis à grande échelle, on ne peut pas faire d'hypothèse forte sur l'existence ou la forme d'un service, ni sur l'organisation du système.

Outre les problèmes de sécurité, l'exigence première est la robustesse des applications dans un tel contexte de volatilité et d'instabilité. Il faut donc intégrer aux applications la capacité de rechercher et de déployer des services déployer dynamiquement, et de les personnaliser afin de les adapter aux conditions d'exécution et aux besoins. Par exemple, déporter sur un site serveur un traitement spécifique qui met en œuvre l'expertise d'un client peut permettre de réduire le nombre et le volume des données échangées sur le réseau qui sont nécessaires à l'interaction ou qui en résultent. De la même manière, la localisation sur le réseau de ressources et de services peut également gagner à être effectuée au moyen d'une algorithmique spécifique qui dépend des connaissances et des besoins du demandeur.

2.4.4. *Implications sur le développement et déploiement*

Les technologies de type client-serveur, RPC[3] à objets ou *Web Services*, ont été développées dans le cadre de systèmes stables dans lesquels les interactions peuvent être prévues. Essentiellement, elles proposent un modèle de programmation qui permet au développeur de faire abstraction de la répartition. Elles permettent d'activer à distance des services génériques conçus pour un grand nombre d'utilisateurs.

Elles n'offrent cependant pas le niveau de flexibilité et d'extensibilité souhaitable pour la construction et le déploiement d'applications dans le contexte décrit ci-dessus. Dans ce travail, nous explorons l'utilisation et l'association de technologies logicielles complémentaires (agents, code et agents mobiles, composants logiciels) qui permettent au contraire la prise en compte de la répartition, des problèmes d'échelle et d'ouverture lors du développement.

---

3. *Remote Procedure Call*

3. Technologies

3.1. *Les architectures « pair à pair »*

Les systèmes « pair à pair » (*peer-to-peer* ou P2P) [MIL 02] sont des systèmes répartis à l'échelle du réseau Internet. Ils reposent sur le principe de mutualisation (échange et partage), au sein d'un réseau logique, de services et de ressources comme par exemple des données, des programmes, des capacités de stockage ou de calcul. Tous les participants (appelés pairs) sont à égalité de devoirs et de droits : chacun peut à la fois rendre accessibles des ressources dont il dispose (opération de publication) et exploiter des ressources fournies par d'autres. En ce sens, le modèle P2P est une alternative au modèle client-serveur classique, les pairs pouvant jouer en même temps le rôle de serveur et le rôle de client. Les systèmes P2P sont par ailleurs des systèmes ouverts : les pairs peuvent librement intégrer et quitter le réseau, et fournir les ressources de manière intermittente et dans une forme susceptible d'évoluer au cours du temps.

Au-delà des problèmes légaux (voire moraux) que posent quelques applications phares permettant l'échange de musique et de films (cf. section 3.1.1), le P2P constitue un véritable modèle d'organisation répartie extensible et robuste qui permet la mutualisation et la capitalisation au sein de communautés ou d'entreprises. En particulier, il semble d'un grand intérêt pour la construction de systèmes d'information répartis à grande échelle qui, par nature, sont hétérogènes et instables.

3.1.1. *Quelques exemples de systèmes P2P*

Le modèle P2P a été popularisé par différentes applications de partage et d'échange de fichiers de musique ou de vidéo. Parmi les plus connues, on peut citer **Napster**[4], **Freenet**[5], **eDonkey**[6], **Gnutella**[7], ou encore **Kazaa**[8]. Aujourd'hui, les échanges au moyen de ces applications représentent une part importante du trafic sur Internet. A titre d'exemple, nous présentons ci-dessous le protocole BitTorrent. Les principales caractéristiques des autres systèmes sont données en section 3.1.2.

Le protocole **BitTorrent**[9] est l'un des plus récents pour le partage de fichiers en réseau, qui profite des expériences de ses prédécesseurs autant sur le plan juridique (l'expérience *Napster*) que concernant la résistance à la montée en charge. Son succès est probablement du à son ouverture (il est *Open Source*, d'où l'implémentation de nombreux clients) mais aussi à l'efficacité de son mode de recherche. Il est particulièrement connu pour être le moyen préconisé de diffusion et de téléchargement des distributions Linux.

---

4. http://www.napster.com
5. http://freenet.sourceforge.net
6. http://www.edonkey2000.com
7. http://www.gnutella.com
8. http://www.kazaa.com
9. http://bittorrent.com/protocol.html

La phase de publication d'une donnée est originale, puisque le propriétaire (appellé *Seeder*) référence son fichier sur un site thématique, via un lien (lien *.torrent*) contenant des méta-données sur son fichier. Parmi les méta-données, la première information est l'URL d'un *Tracker*, un serveur spécifique qui associe à l'identifiant du fichier une liste de clients qui le possèdent (intégralement ou en morceaux) et le partagent.

Côté client, la recherche se fait initialement par l'intermédiaire d'un navigateur Web, sur des sites référençant des liens *.torrent*. Le téléchargement débute lors d'un simple clic sur un lien donné par le navigateur, provoquant l'activation du logiciel. Celui-ci contacte le *Tracker* associé au lien, afin de récupérer une liste de clients possédant le fichier choisi. Enfin, les transferts de données se font entre clients par l'intermédiaire d'une connexion directe en TCP/IP.

3.1.2. *Typologie des systèmes P2P*

Par nature, l'instabilité du système et la volatilité des ressources sont fortes et incontrôlées. Un système P2P robuste doit donc être flexible et capable de s'adapter dynamiquement, de manière transparente pour l'utilisateur. Du fait de l'échelle (la dimension du réseau et le nombre de pairs, typiquement des milliers, des dizaines de milliers, voire davantage), la communauté ne peut pas être informée des changements (évolutions, connexions, déconnexions, pannes). Ainsi, lorsqu'un pair veut accéder à une ressource, il n'a *a priori* qu'une connaissance partielle (qui peut être en partie obsolète) de l'état du système P2P (de la disponibilité des serveurs et de la qualité des services). La localisation des ressources est donc une fonctionnalité essentielle.

On peut distinguer trois types d'architecture de systèmes P2P, en fonction du mode de mise en relation entre demandeur et fournisseur, c'est-à-dire en fonction du mécanisme de localisation des ressources. Ensuite, les échanges de données (plus généralement l'exploitation des ressources) se font directement entre pairs sans intermédiaire.

1) Les systèmes les plus simples, à la limite du P2P et du client-serveur, sont dits **centralisés**. Ils se basent sur un serveur unique pour indexer les ressources (i.e. Napster). A chaque connexion d'un pair au réseau, ce dernier annonce au serveur la liste des ressources qu'il met à disposition de la communauté. En phase de recherche, le client effectue une requête auprès du serveur qui retourne la ou les références des pairs qui offrent une ressource correspondant aux critères de la recherche. L'échange se fait ensuite directement entre les pairs concernés. Même si l'expérience « Napster » a démontré la viabilité de cette solution sur le plan technique, la centralisation au niveau du serveur présente des limites en terme d'extensibilité (passage à l'échelle) et constitue un point de faiblesse en cas de surcharge voire de panne.

2) Les systèmes **semi-décentralisés** ou **hybrides** conservent le principe d'un serveur de localisation mais s'appuient sur un ensemble de serveurs répartis (i.e. eDonkey) afin de minimiser les conséquences des pannes et de réduire les risques de contention. Dans certains cas (i.e. Kazaa), des pairs peuvent être spécialement choisis pour leur capacité de calcul et de bande passante (des *superpeers*) ; leur rôle est d'indexer les ressources d'un sous-ensemble du réseau.

3) Enfin, en mode P2P pur, il n'existe pas de point de centralisation ; au contraire, les systèmes P2P purs sont complètement **décentralisés** et auto-organisés. Dans ce cas, localiser une ressource est une opération critique qui demande *a priori* le parcours d'une partie du réseau. On peut distinguer deux grandes catégories de systèmes P2P décentralisés :

a) Les réseaux dits **structurés** (i.e. Freenet, Chord...) conservent un index qui est réparti sur l'ensemble des pairs sur le principe d'une table de hachage distribuée [RAT 01, STO 01, CLA 00]. La recherche consiste en un routage de la requête à travers le réseau piloté par l'index. La longueur du chemin d'accès à une ressource est bornée (il dépend de la topologie du réseau). Mais, l'ajout et le retrait d'un pair sont des opérations complexes et coûteuses. Par ailleurs, si l'un des pairs devient indisponible, les ressources qu'il indexe ne peuvent plus être retrouvées alors qu'elles peuvent être disponibles.

b) Les réseaux dits **non structurés** (i.e. Gnutella, PeerCast...) se passent d'organisation globale et d'index. Ainsi, ils éliminent le coût de maintien de l'index réparti dans un état cohérent. Les recherches consistent à explorer le réseau. Elles s'effectuent dynamiquement d'abord auprès de voisins (des pairs connus du client) puis récursivement au sein du réseau. Il résulte de la propagation de la requête un phénomène « d'inondation » (*flooding*) coûteux en bande passante, dont on contrôle les effets en associant aux requêtes une durée de vie limitée (*Time To Live*) ; alors, les recherches ne sont plus exhaustives et la localisation d'une ressource disponible n'est pas assurée, même en l'absence de panne. Néanmoins, cette capacité de découverte dynamique permet de fonctionner dans des réseaux inorganisés, instables et en constante évolution. Aussi, des travaux sont en cours afin de proposer des solutions (à base de profil client et d'apprentissage) pour rendre moins aveugle et donc plus efficace les procédures de recherche (voir par exemple [HAN 04]).

3.1.3. *Proposition d'architecture pour les systèmes P2P décentralisés*

Les systèmes P2P purs décentralisés sont les mieux adaptés à la mise à disposition de ressources volatiles ou évolutives et aux systèmes à grande échelle et à topologie instable. Dans cette section, nous proposons d'identifier leurs différentes fonctionnalités et d'en déduire l'ensemble des composants[10] d'une architecture générique de ces systèmes.

Outre les fonctionnalités d'adhésion volontaire au réseau et de retrait, les pairs doivent au minimum pouvoir

– mettre des ressources à disposition de la communauté (publication),

– rechercher une ressource à partir d'une description,

10. Nous employons ici le terme « composant » pour désigner une unité de structuration de l'application. Le concept de « composant logiciel » est présenté plus précisément dans la section 3.4.

– exploiter une ressource trouvée.

1) La publication d'une ressource consiste à l'insérer dans le réseau et à permettre à d'autres pairs d'y accéder. Au plus simple, la ressource est hébergée localement chez le propriétaire, mais elle peut aussi être hébergée ailleurs sur le réseau. Dans le cas des réseaux structurés, c'est l'index qui est hébergé quelque part sur le réseau. Un composant de **publication** est donc nécessaire, et du côté de l'hébergeur un mécanisme doit permettre la publication de ressources par un tiers.

2) Chaque pair doit pouvoir accéder à d'autres pour y découvrir les ressources disponibles. La localisation dans les systèmes P2P purs ne passe pas par l'utilisation de serveurs ; il y a donc au départ un besoin de découverte et de localisation des pairs serveurs, puis ensuite sur ces pairs un besoin de fouille locale pour l'extraction des ressources et services. Pour cela nous distinguons deux composants au sein du mécanisme de recherche. L'un de **localisation** pour la découverte et l'accès aux pairs, l'autre pour la **recherche locale** sur le pair serveur[11].

3) En complément à la localisation, le système doit permettre à un pair de découvrir des méta-informations sur le système (sur les pairs et les ressources) et de maintenir un ensemble de connaissances sur le réseau P2P. Ces connaissances sont exploitées pour optimiser les recherches futures (par inondation). Pour cela, un élément d'**information** répertorie les connaissances des pairs sur eux-mêmes et sur le réseau P2P (méta-informations).

4) Après avoir découvert une ressource, il faut pouvoir l'exploiter. Dans le cas le plus simple, il s'agit de télécharger un fichier trouvé. Plus généralement, il peut s'agir par exemple de communiquer à l'utilisateur la description de la ressource trouvée, d'exécuter un service puis de transmettre les résultats, de lancer une nouvelle recherche si la ressource trouvée dépend d'une autre, et complémentairement de superviser l'exploitation de la ressource. Ces opérations sont à la charge d'un composant dédié à **l'exploitation des ressources**.

5) De plus, dans un contexte de ressources volatiles et évolutives, un composant supplémentaire fournissant un moyen de s'interfacer de manière uniforme avec les ressources (accès à une base de données, à un système de fichiers, à un web service...) peut être ajouté. De cette façon par exemple, si la forme d'un web service change (nouvelle version) seul le composant d'**accès aux ressources** doit être modifié. Des systèmes comme Spitfire[12], utilisé dans le projet DATAGRID pour l'accès uniforme aux bases de données, existent déjà.

3.1.4. *Plateformes de développement de systèmes*

Nous avons étudié quelques environnements de développement susceptibles de permettre la construction d'applications P2P, dont les deux derniers utilisant le paradigme agent, afin d'étudier leur adéquation à ce contexte et de situer notre approche.

---

11. Ces deux composants jouent un rôle équivalent au composant d'allocation de ressources dans les systèmes centralisés
12. http://edg-wp2.web.cern.ch/edg-wp2/spitfire

XtremWeb[13] est une plateforme pour la distribution de tâches en mode P2P sur des grilles de calcul, utilisée dans le cadre de l'ACI GRID. Elle a été conçue pour la distribution de calculs sur des pairs qui peuvent être demandeurs ou fournisseurs de ressources de calcul. La distribution est par nature centralisée même si une hiérarchie de serveurs permet de réduire la charge sur le serveur central.

Globus est une alliance pour le développement de technologies autour de la grille de calcul ouverte. Le Globus Toolkit[14] est une implémentation en logiciel libre du standard OGSI (Open Grid Services Infrastructure), un modèle de *Web Services* adaptés à la grille. Cette boîte à outils offre les services et les outils de base requis pour construire une grille de calcul (sécurité, localisation des ressources, gestion des ressources, transmissions). La lourdeur du *framework* et les difficultés de déploiement le rendent moins adapté aux systèmes P2P les plus dynamiques.

JXTA[15] est une initiative de Sun Microsystems, proposant un ensemble de protocoles ouverts pour interconnecter des dispositifs allant du téléphone cellulaire jusqu'aux serveurs. Le fonctionnement repose sur le découpage de parties du réseau réel en réseaux virtuels, dans lesquels chaque pair peut accéder aux ressources des autres sans se préoccuper de leur localisation ou de leur environnement d'exécution. JXTA propose une base de six protocoles (services de découverte des pairs, de rendez-vous, d'informations sur les pairs, de communication, de routage et de résolution des pairs) dans lesquels la communication se fait par des messages XML. Plusieurs implémentations existent, par exemple en Java avec JXME[16]. Cette approche semble intéressante dans le contexte visé, néanmoins elle nous semble plutôt réservée à l'exécution de services distants, sans possibilité de personnalisation.

Un système **Anthill**[17] est un ensemble de pairs sur lesquels est déployé un système multi-agents, dont l'interaction permet de résoudre des problèmes complexes (grâce à des comportements émergents et des algorithme génétiques) comme par exemple le routage des messages. Anthill s'inspire des colonies de fourmis, en proposant des agents aux comportements simples, autonomes, et pourvus d'un environnement sur lequel ils basent leurs actions [BAB 02]. Chaque pair est un « nid », qui fournit à l'application des services spécifiques au P2P : gestion de ressources, communication, gestion de la topologie, planification des actions.

Jade[18] (Java Agent DEvelopment Framework) est un *framework* Java pour l'implémentation de systèmes multi-agents au dessus d'un *middleware* conforme aux standards FIPA[19], notamment pour permettre l'interopérabilité entre agents. Elle permet de réaliser des systèmes P2P dans lesquels chaque pair peut fonctionner de manière *proactive*, communiquer sans se soucier de la localisation, et se coordonner pour ré-

13. http://www.lri.fr/~fedak/XtremWeb
14. http://www-unix.globus.org/toolkit
15. http://www.jxta.org
16. http://platform.jxta.org
17. http://www.cs.unibo.it/projects/anthill
18. http://jade.tilab.com
19. http://www.fipa.org

soudre un problème complexe grâce à l'utilisation d'agents. Un agent spécial unique (Agent Managing System ou AMS) sert de superviseur pour la plateforme, et fournit les services d'annuaire, de cycle de vie [BEL 04]. Tout agent créé doit s'enregistrer auprès de l'AMS pour obtenir son identifiant, indispensable à la communication. Ce point de centralisation est un frein à l'adoption de Jade dans des systèmes répartis à grande échelle, et restreint son usage à des clusters de machines ou des réseaux de petite taille.

Hormis JXTA, aucune de ces plateformes n'est vraiment adéquate pour la construction de systèmes P2P purs. En particulier, la plupart des *middleware* pour la grille reposent sur des services centralisés (éventuellement hiérarchisés) d'allocation de ressources à des tâches indépendantes. Nous proposons une approche qui se démarque des technologies citées ci-dessus par la décentralisation, l'utilisation de composants génériques et la personnalisation des services au moyen d'agents mobiles.

3.2. *Les agents mobiles*

3.2.1. *Définitions*

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

Un **agent mobile** [FUG 98, BER 02] est un agent logiciel qui peut se déplacer d'un site à un autre en cours d'exécution pour se rapprocher de données ou de ressources. Il se déplace avec son code et ses données propres, mais aussi avec son état d'exécution. L'agent décide lui-même de manière autonome de ses mouvements. En pratique, la mobilité ne se substitue pas aux capacités de communication des agents mais les complète (la communication distante, moins coûteuse dans certains cas, reste possible). Afin de satisfaire aux contraintes des réseaux de grande taille ou sans fil (latence, non permanence des liens de communication), les agents communiquent par messages asynchrones.

En complément, un agent logiciel est dit **adaptable** si certains de ses mécanismes internes, opérationnels (envoi de messages, déplacement…) ou fonctionnels (comportement), sont modifiables en cours d'exécution. Conformément à la propriété d'autonomie, l'agent contrôle lui-même ses propres évolutions.

3.2.2. *Apport des agents mobiles*

L'utilisation d'agents conduit à la décentralisation de la connaissance et du contrôle. Développé aux frontières du génie logiciel et des systèmes répartis, le concept d'agent mobile est *a priori* destiné à la mise en œuvre d'applications dont les performances varient en fonction de la disponibilité et de la qualité des services et des ressources, ainsi

que du volume des données déplacées : par exemple, un agent mobile peut adapter un service aux capacités du poste client et à la bande passante disponible.

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 [CHE 94], 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 qui ne transporte plus que les données utiles (éventuellement pré-traitées). En outre, cela permet des transactions plus robustes que les transactions distantes.

– L'exécution d'agents spécialisés offre davantage de souplesse que l'exécution d'une procédure standard sur les sites serveurs.

– L'asynchronisme, l'autonomie et la réactivité 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 (clients ou serveurs d'information).

Le principal apport des agents mobiles se situe sur un plan du génie logiciel : les agents mobiles sont des unités de structuration des applications ; ils unifient en un modèle unique différents paradigmes d'interaction entre entités réparties (client-serveur, communication par messages, code mobile). Nous considérons que, par nature, ils constituent un outil privilégié pour l'adaptation et le déploiement.

3.2.3. *Discussion*

Plusieurs expériences significatives ont été menés concernant l'apport des agents mobiles, en particulier dans le domaine de la recherche d'information [BRE 99, PAP 00, HAM 00, KOS 01, THA 01]. Cependant, on constate qu'en pratique les technologies à base d'agents mobiles n'ont encore été que peu utilisées. On peut avancer ici quelques éléments d'explication :

– Les problèmes de sécurité posés par l'utilisation d'agents mobiles constituent un frein à l'expansion de cette technologie. Les risques concernent la confidentialité, l'intégrité et la disponibilité des machines hôtes et des agents, ainsi que des échanges sur le réseau [TSC 99]. Des éléments de solutions existent à base de chiffrement symétrique ou asymétrique (clé publique/privée) et de confinement (cf. section 4.1.2). Le problème reste cependant ouvert et dépasse largement la problématique des agents mobiles (comme par exemple le problème des attaques par déni de service[20]).

– Par rapport aux technologies plus classiques, le concept d'agent mobile offre un cadre fédérateur pour traiter des problèmes différents 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.

---

20. *denial of service* : un service est saturé de requêtes afin de le rendre indisponible

– Les expériences primitivement menées avec des applications relativement figées dans des environnements stables et des réseaux locaux mettaient peu en évidence l'intérêt des agents mobiles en terme de flexibilité. On peut penser qu'aujourd'hui le contexte applicatif a changé et qu'il est nécessaire d'explorer davantage cette technologie au regard des problèmes posés par le passage à l'échelle et l'ouverture des systèmes.

3.3. *JAVACT : un middleware à base d'agents mobiles adaptables*

Pour la mise en œuvre des solutions présentées dans la section suivante, nous nous appuyons sur JAVACT[21] [ARC 04], un middleware 100 % Java développé dans notre équipe. Outre un haut niveau d'abstraction par rapport à la répartition et aux opérations distantes, JAVACT offre un modèle de programmation à base d'agents mobiles ainsi que des mécanismes d'évolution et d'adaptation fins au niveau des agents[22].

Une application JAVACT s'exécute sur un ensemble de places (des machines virtuelles Java) connectées en réseau, *via* un *middleware* de plus bas niveau (de type Java RMI, CORBA, SOAP, voire socket TCP/IP) dont l'utilisation est cachée au programmeur de l'application. Sur les places, un ensemble d'agents répartis et mobiles coopèrent pour résoudre un problème commun. 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 [AGH 86], entités logicielles autonomes qui communiquent par messages asynchrones et traitent en série les messages reçus. Le comportement des acteurs évolue en cours d'exécution : le traitement d'un message définit le comportement pour le message suivant. On en déduit une sémantique claire de la mobilité d'acteur : le déplacement s'effectue après le traitement du message courant ; l'acteur traite alors à distance le message suivant avec son nouveau comportement, l'acteur d'origine devenant un simple relais vers l'acteur mobile et assurant l'acheminement des messages.

La bibliothèque JAVACT offre une dizaine de primitives pour la programmation des comportements d'agents. Voici celles que l'on retrouve dans les codes présentés plus bas :

– $create(comportement)$: crée localement un agent dont le comportement est passé en paramètre ;

– $become(comportement)$: l'agent change de comportement pour traiter le prochain message

---

21. JAVACT (cf. http://www.irit.fr/recherches/ISPR/IAM/JavAct.html) est distribué sous forme de logiciel libre sous licence LGPL

22. Sur ce point, JAVACT diffère de Jade et de ProActive (cf. http://www.inria.fr/oasis/ProActive) par exemple. Précisons que JAVACT n'est pas une plateforme dédiée au développement de systèmes multi-agents mais plutôt un outil de programmation répartie.

– $suicide()$: termine l'exécution de l'agent ;

– $myPlace()$: renvoie à l'agent l'adresse de la place sur laquelle il s'exécute ;

– $send(message, destinataire)$: envoie un message à un autre agent ;

– $go(destination)$: l'agent se déplace pour traiter le message suivant sur la nouvelle place.

Outre l'évolution par changement de comportement, l'adaptabilité est permise par la réflexivité de l'architecture [MIG 99] qui associe à chaque fonctionnalité élémentaire de l'agent un micro-composant dédié. Ces micro-composants peuvent être remplacés, spécialisés ou composés, afin de changer la sémantique des primitives offertes au niveau applicatif, sans avoir besoin de retoucher le code des comportements (voir [LER 04a] pour plus de détails). La plupart du temps, ces micro-composants sont spécialisés pour offrir de nouvelles propriétés non-fonctionnelles (ajout de code de cryptage, de sûreté…), comme nous l'illustrons dans la section 4.3.

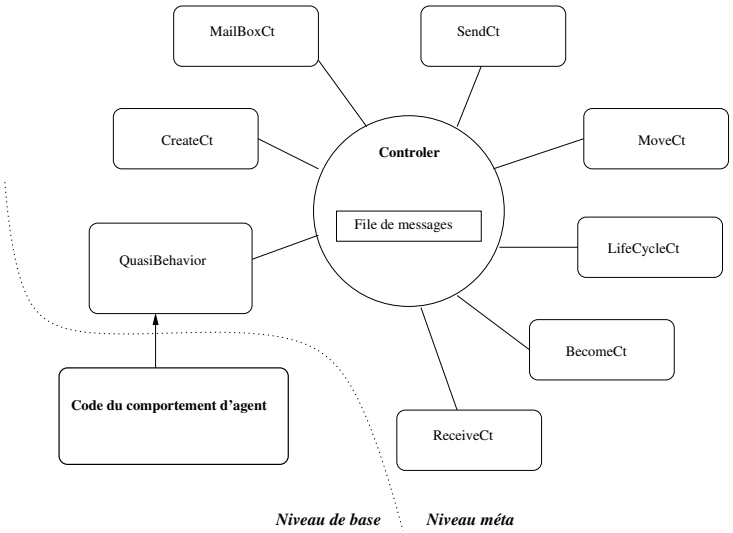

Figure 3. *Architecture simplifiée d'un agent* JAVACT *: réflexivité et micro-composants*

3.4. *Les composants*

Sur le plan du génie logiciel, développer une application aujourd'hui revient souvent à trouver le bon compromis entre la production de code sur mesure (spécifique mais coûteux et long à développer) et la réutilisation de logiciel existant (qu'il suffit parfois de configurer, mais qui peut être moins adapté aux besoins). L'approche « composant » facilite ce compromis et constitue un paradigme de programmation particulièrement avantageux dans le cadre des systèmes répartis.

Un **composant logiciel** [SZY 02, MAR 02] est un morceau de logiciel « assez petit pour que l'on puisse le créer et le maintenir, et assez grand pour qu'on puisse l'installer et le réutiliser ». C'est une unité de structuration et de composition, qui spécifie précisément via des interfaces quelles sont ses dépendances ainsi que ce qu'elle fournit (services, évènements...). Le composant logiciel est une évolution du concept d'objet qui reprend ses objectifs d'origine tels la séparation des préoccupations, la réduction de la complexité et la réutilisation dans une perspective de structuration plus forte des applications. Un composant peut être déployé indépendamment de toute application, et notamment de son langage de conception.

Chaque modèle de composant raffine cette définition et précise éventuellement des règles d'assemblage. Le modèle des Enterprise JavaBeans (EJB) de Sun Microsystems[23] présente le composant comme l'unité d'architecture de sa plate-forme serveur d'application (J2EE) ; celui de Microsoft, le Component Object Model (COM)[24] ainsi que le Corba Component Model [25] permettent l'assemblage à plat des composants d'une application ; le modèle de composant Fractal[26] ajoute une dimension hiérarchique, dans laquelle des composants peuvent être assemblés en composites, et être ainsi perçus comme de nouveaux composants.

Pour chaque modèle, un environnement d'exécution (ou plateforme à composants) fournit l'ensemble des services permettant leur instanciation, leur configuration et leur exécution, en leur offrant des services non fonctionnels comme par exemple des mécanismes de communication, de transaction, d'authentification, de persistance... Les composants accèdent à ces services au travers d'une entité appelée **conteneur**, sorte de capsule au sein laquelle s'exécute le composant et qui permet son adaptation. L'architecture conteneur-composant accentue la séparation entre les aspects non fonctionnels et le code métier du composant.

4. Un *framework* à base de composants et d'agents

Dans la section 3.1.3, nous avons proposé une décomposition des systèmes P2P et une architecture générique dont les constituants peuvent être naturellement implantés par des composants logiciels. Nous montrons dans cette section comment, en associant les concepts de composant logiciel et d'agent, nous dérivons de l'architecture générique un *framework* qui peut servir de base à la construction de différentes applications (pour une étude sur les apports mutuels entre agents et composants, on peut se référer à [BRI 04]). En outre, nous montrons le rôle essentiel joué par la mobilité et l'adaptabilité des agents dans la phase de déploiement.

---

23. http ://java.sun.com/products/ejb
24. http ://www.microsoft.com/com
25. http ://forge.objectweb.org/projects/openccm
26. http ://forge.objectweb.org/projects/fractal

4.1. *Principes généraux*

4.1.1. *Déploiement et adaptation des composants par des agents*

Il est possible de faire une analogie entre le modèle composant-conteneur et notre modèle d'agent adaptable, particulièrement si l'on considère la séparation des préoccupations. D'une part, les composants comme les comportements des agents représentent le code fonctionnel (code métier). D'autre part, le conteneur permet au composant d'instancier, d'activer et d'accéder aux services non fonctionnels fournis par l'environnement d'exécution (par exemple la gestion du cycle de vie, de la sécurité ou des communications). Dans notre modèle, l'agent joue le rôle de conteneur : ses micro-composants enveloppent le comportement et implantent les mécanismes non-fonctionnels (envoi des messages, réception, cycle de vie…) ou éventuellement les délèguent au système d'accueil.

Dans ce qui suit, nous proposons d'utiliser des composants (sans supposer de modèle particulier) sous la forme de comportements d'agents (ou en leur sein). Pour déployer les composants, on bénéficie alors des avantages des agents en termes de mobilité et d'adaptation : on peut déplacer et adapter le composant via l'agent mobile qui le contient.

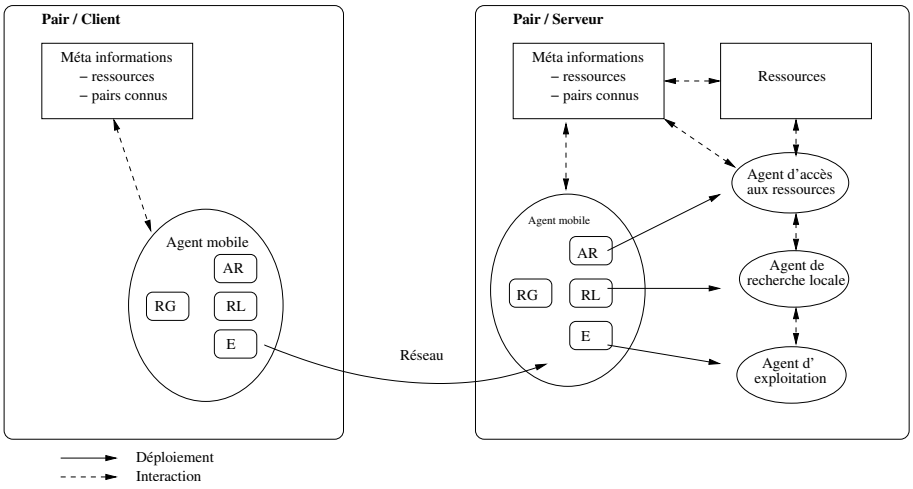

Figure 4. *Déploiement des composants*

Le principe du déploiement est schématisé dans la figure 4 (pour une meilleure lisibilité, on a distingué le pair jouant le rôle de client de celui jouant le rôle de serveur mais ils sont bien sûr complètement symétriques) :

– Pour chaque requête d'un client, un agent mobile (ou plusieurs pour exploiter le parallélisme) transporte les composants sur le réseau. Le composant de localisation constitue l'essentiel de son comportement. Cet agent mobile se déplace de pair en pair

en effectuant dynamiquement la localisation, ce qui lui permet d'adapter la recherche à l'état courant du système P2P. Ainsi, l'agent mobile est le **vecteur du déploiement** adaptatif des composants.

– Lorsqu'il arrive à destination, l'agent mobile installe un mini système multi-agent déployé à partir des composants, pour traiter la requête localement. Là, chaque composant est encapsulé dans un agent dont il constitue le comportement et à travers lequel il offre ses services. Inversement, l'agent sert de conteneur et d'activateur au composant et lui fournit les mécanismes non fonctionnels nécessaires à son exécution. Notre architecture d'agent mobile adaptable permet d'obtenir la flexibilité et l'extensibilité nécessaire à l'adaptation : adapter un agent revient à changer un ou plusieurs de ses micro-composants. L'agent est ainsi le **vecteur de l'adaptation individuelle** des composants. Cette forme d'adaptation est complètement séparée de la programmation des composants, ce qui contribue à simplifier leur développement.

De manière générale, les composants peuvent être fournis par le pair client, par le pair serveur ou par un tiers. Ainsi, le composant d'accès aux ressources peut ne pas être systématiquement fourni par le serveur. Mais en cas d'*upgrade* côté serveur (évolution du mécanisme d'accès aux ressources), il faut prévoir un protocole permettant l'acquisition *in situ* et à la volée du nouveau composant d'accès. Celui-ci doit pouvoir être authentifié (signature du fournisseur) pour éviter les problèmes de sécurité. Ceci contribue à la robustesse de la recherche, permet l'adaptation dynamique aux évolutions des serveurs et limite les opérations de maintenance des serveurs à un cadre local.

4.1.2. *Sécurité*

Un défaut souvent reproché aux systèmes d'agents mobiles est le faible niveau de sécurité qui entoure leur exécution [TSC 99]. En particulier les communications sur le réseau peuvent être écoutées et modifiées, la confidentialité et l'intégrité des agents (données, code) peuvent être atteintes par le système d'accueil ou d'autres agents, enfin les agents peuvent nuire au système d'accueil *via* un accès non autorisé aux ressources. La définition de politiques de sécurité sort du cadre de ce travail. Ici, nous montrons seulement comment la mise en œuvre de quelques solutions classiques [SCH 01] permet de sécuriser le partage et la recherche d'information.

L'authentification des différentes entités ainsi que le cryptage des informations peut être réalisé à partir de cryptographie à clé publique. Ainsi, d'une part l'intégrité des agents peut être vérifiée par leurs propriétaires, d'autre part, les hôtes peuvent s'assurer de l'origine et du propriétaire de l'agent qui va s'exécuter sur leur système. On peut penser que l'implantation d'une requête par assemblage de composants réutilisables testés et certifiés contribue à préserver la sécurité des serveurs.

Le confinement des sites permet la réduction des risques d'atteinte des systèmes d'accueil par des agents néfastes. Par exemple, une machine virtuelle java peut être paramétrée par un ensemble de permissions[27], permettant de filtrer les interactions

27. http://java.sun.com/j2se/1.4.2/docs/guide/security/permissions.html

du niveau réseau (plages d'adresses) jusqu'au niveau langage. Des infrastructures de filtrage matérielles tels les *firewall* peuvent également être employées. De plus, les agents mobiles de recherche peuvent ne pas avoir besoin d'accès aux ressources en écriture.

Enfin, dans certaines applications de recherche d'information (par exemple celles de la distribution de logiciel et du dossier médical), des hypothèses de confiances peuvent être faites sur les entités jouant le rôle de serveur, simplifiant le problème de protection des agents.

4.2. *Mise en œuvre d'un prototype de logiciel P2P*

Afin de vérifier la validité de nos propositions, nous avons conçu un prototype de logiciel P2P que nous avons appellé JAVANE à partir de notre *framework* [LER 04b]. Ce prototype, qui a été entièrement implanté, testé et validé expérimentalement dans des configurations variées (réseaux internet et intranet, wifi, PC, stations Sun…), répond à la spécification proposée au 2.1.

Au niveau de l'application, différents composants ont été implantés sous forme de comportement d'agents JAVACT, mettant en œuvre différents protocoles :

– composant de localisation : recherche de pairs par inondation (type Gnutella), par cheminement d'agent mobile, hybride

– composant d'exploitation : téléchargement de fichier

– composant de recherche locale : recherche de fichier par filtre multi-critères (taille, type, nom…)

– composant d'accès aux ressources : accès à une base de donnée au format texte (fichiers + descriptions)

En complément, un mécanisme d'acquisition dynamique d'informations sur le domaine a été ajouté. En parcourant le réseau et en explorant des zones primitivement inconnues du client, il est possible pour ce dernier de glaner des connaissances sur les sites visités lors de la phase de recherche. Ces connaissances permettent de mettre à jour les bases de données qui, du côté du client, répertorient les propriétés des serveurs distants connus et alimentent les (futures) sélections de serveur. Plus le client dispose d'informations sur d'autres sites, plus les chances de trouver l'information recherchée augmentent. Le protocole de recherche présenté ci-dessous permet l'acquisition dynamique d'informations par le client, *via* un simple envoi de message de l'agent au client.

4.2.1. *Un composant de localisation*

Dans le cadre de cette étude, plusieurs composants de recherche globale (localisation) ont été effectivement conçus, implémentant différent protocoles de recherche. Nous décrivons ici l'un d'entre eux, afin d'illustrer la mise en œuvre au moyen des agents mobiles. Celui-ci s'inspire des systèmes de recherche par inondations (type

Gnutella), paramétré par un nombre maximal d'agents de recherches déployés en parallèle, et limités par une variable décroissante, ici d'énergie (*Time To Live*).

La recherche se déroule en deux phases. Dans la première, un agent représentant le client crée n agents sur n sites initialement connus du client, sur lesquels ils effectuent leurs recherches. Lors de la seconde, chaque agent de recherche globale peut se déplacer sur un site connu de son hôte (et pas nécessairement du client originel) pour y poursuivre ses recherches.

La figure 5 présente le code fonctionnel du comportement de l'agent de recherche globale (avec le mécanisme d'acquisition dynamique d'informations sur le domaine), débarrassé des déclarations de variables et des traitements d'erreurs. La méthode $run()$ est automatiquement exécutée lors de l'arrivée de l'agent sur une place (déclarée dans l'interface $StandAlone$).

L'expansion du système de recherche est limitée par l'utilisation d'une variable d'énergie interne à chaque agent qui représente son potentiel d'activité et qui décroît à chaque mouvement et en fonction des résultats (trouvé, non trouvé, erreur). Un agent qui n'a plus d'énergie ou qui ne peut peut plus se déplacer (panne, déconnexion...) s'arrête. La couverture du réseau est donc partielle (sur un réseau de grande taille l'exhaustivité est trop coûteuse voire impossible), mais en précisant les paramètres n et $energie$, l'utilisateur peut influencer sur la quantité de documents trouvés ou la rapidité de réponse. Par ailleurs pour des raisons d'efficacité, 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 ; dans le cas général, il évolue par changement de comportement.

4.2.2. *Un composant de téléchargement de fichier*

Le composant de téléchargement proposé ici 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[28]).

Un agent récepteur situé sur le site client, récupère en parallèle des parties distinctes du fichier à télécharger, qui lui sont envoyées par des agents émetteurs (agents d'exploitation) créés sur les sites serveurs sélectionnés. Afin de s'adapter aux bandes passantes disponibles, l'agent récepteur peut stopper ou réviser dynamiquement les tailles des tranches allouées à chaque agent émetteur en fonction du débit de chaque canal. Côté client, un contrôle d'intégrité est effectué lorsque le fichier est réputé complet, en recalculant sa signature et en la comparant avec celle fournie par la recherche.

---

28. http://www.itl.nist.gov/fipspubs/fip180-1.htm

```
public class InondationChercheur extends ComposantRG implements StandAlone {

    public InondationChercheur(Actor client, Vector parms, ComposantAR cAR,
                                                ComposantRL cRL, ComposantE E) {
        energie=((Integer) parms.get(0)).intValue();
        signatureRech=(String) parms.get(1);
        ...
    }

    public void run() {
        // Si plus d'energie, on stoppe la recherche globale
        if (energie <= 0) {
            suicide();
            return;
        }

        // A-t-on deja visite la place courante ?
        if (! marquerPlace(myPlace(), signatureRech, client))
            energie -= E_VISITEE;
        else {
            energie -=E_NOUVEAU;

            // Creation du èsystme d'agents
            AgentAR=create(cAR);
            AgentRL=create(cRL);
            AgentE=create(E);

            // Activation de l'agent de recherche locale,
            // puis poursuite de la recherche globale.
            // Les activites sont completement paralleles
            send(new JAMstart(client,AgentAR,AgentE),AgentRL);
        }

        // Collecte d'informations sur le domaine
        String[] infosPlaces = getInfosPlaces(myPlace());
        send(new JAMaddInfosPlaces(infosPlaces), client);

        // Deplacement aleatoire s'il reste de l'energie, sinon suicide
        if (energie > 0) {
            String[] places = getPlaces(myPlace());
            String prochainePlace = places[random.nextInt(places.length)];
            go(prochainePlace);
        } else suicide();
    }
}
```

Figure 5. *Comportement de l'agent de recherche*

Dans la figure 6, le code a été débarrassé des déclarations de variables et des traitements d'erreurs. La méthode $executer()$ (respectivement $adapter()$ et $stop()$) est exécutée lorsque l'agent traite un message de la classe $JAMexecuter$ (respectivement $JAMadapter$ et $JAMstop$).

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.

```
public class ExploitationTelechargementFichier extends ComposantE{

    public void executer(Object gestFichiers) {
        nomComplet=(String) gestFichiers;

        // Lors de la premiere activation, on ouvre le fichier et on positionne
        // le pointeur au édbut de la tranche allouee
        if (premiereActivation) {
          fichier=new FileInputStream(nomComplet);
          fichier.skip(maTranche.getDebut());
        }
        premiereActivation=false;
        if (maTranche.Finie()) {
          stop();
          return;
        }

        // Lecture de la tranche suivante
        maTranche.progression(nbLus);
        if (maTranche.getAvancement()+TAILLE_MORCEAU > maTranche.getFin()+1)
                              nbALire=maTranche.getFin() -
          maTranche.getAvancement() + 1;
        else nbALire=TAILLE_MORCEAU;
        tab=new byte[(int)nbALire];
        nbLus=fichier.read(tab);

        // Envoi du morceau au client et reactivation
        morceau=new MorceauDeTranche(maTranche.getNumero(), tab,
                                     numTelechargeur);
        send(new JAMtraiterResultat(morceau), client);
        send(new JAMexecuter(gestFichiers), ego());
    }

    public void stop() {
        fichier.close();
        suicide();
    }

    public void adapter(int nouvelleTaille) {
        TAILLE_MORCEAU = nouvelleTaille;
    }
```

Figure 6. *Composant de téléchargement de fichier*

4.3. *Exemples d'adaptation du middleware*

Pour augmenter la robustesse de l'application face aux pannes et aux problèmes de sécurité des communications à l'échelle de l'Internet, nous avons également écrit un ensemble de micro-composants au niveau du *middleware* JAVACT, en particulier communication cryptée, tolérance aux déconnections et authentification des comportements. Nous présentons brièvement ci-dessous comment sont implantés les deux premiers. Ces micro-composants ont une bonne réutilisabilité, puisqu'ils peuvent être déployés sur les agents utilisés dans les exemples qui suivent.

4.3.1. *Envoi de messages robuste*

La tolérance aux déconnexions est une qualités de service nécessaire au bon fonctionnement des systèmes d'agents mobiles, en particulier lorsque ceux-ci évoluent dans des environnements sans fil : liaisons infra-rouges, radio (WiFi, GSM, etc...). En effet, un éloignement trop important entre l'émetteur et la base peut interrompre sans préavis la communication, de la même façon que la présence d'un obstacle qui perturberait les signaux (passage dans un tunnel...). Ce type de déconnexion est généralement de courte durée, et les agents mobiles doivent pouvoir faire face à de telles situations.

Pour résoudre cela, nous avons développé un micro-composant d'envoi de messages adapté. L'appel de la méthode *send()* de ce composant place le message en attente dans une file, puis retourne à l'appelant. Un thread actif depuis la création du composant s'occupe de vérifier périodiquement si la file est non vide, auquel cas il tente l'envoi (via le composant *Send* standard) de chaque message. En cas de succès, le message est retiré de la file.

4.3.2. *Envoi de messages cryptés*

Le besoin de communications cryptées existe dans certains environnements, où le support n'est pas sécurisé et où les risques d'interception sont importants. Par exemple lors de l'utilisation d'un réseau sans fil 'standard', il est envisageable que les informations échangées entre la base et le récepteur soient écoutées.

Le micro-composant *SendCryptCt* encapsule le micro-composant de base d'envoi de message, son travail se réduisant à la sérialisation puis au cryptage du message. Quelques algorithmes de cryptage symétriques courants ont été mis en œuvre.

4.4. *Mise en œuvre d'autres applications*

Afin de vérifier la réutilisabilité du *framework* et de certains composants comme ceux de localisation ou de téléchargement, nous nous intéressons ici à la construction de quelques applications autres que JAVANE, et nous identifions les composants nécessaires.

Le modèle d'interaction défini dans l'exemple précédent (localisation-recherche locale-accès aux ressources-exploitation des résultats) peut être appliqué à la recherche et à l'exploitation de services sur le réseau en mode client-serveur. L'accès aux ressources consiste à demander l'exécution du service en lui fournissant les paramètres d'appel, et l'exploitation des résultats consiste à retourner ceux-ci au client initiateur de la requête.

4.4.1. *Déploiement automatique de logiciels*

Le déploiement automatique de logiciels s'appuie sur des mécanismes de recherche sur le réseau et des opérations effectués sur des sites distants du site initiateur. Le tableau ci-dessous identifie le rôle des différents composants.

	Scénario "push"	**Scénario "pull"**
Recherche Globale	Localisation des utilisateurs connus	Localisation de distributeurs ou de producteurs pour un composant donné
Recherche Locale	Recherche et vérification de la version d'un module	Recherche d'un composant logiciel
Accès aux ressources	Directement via le système de fichiers	Base de données, Web Service…
Exploitation	Déploiement : copie physique et configuration du composant à déployer	Téléchargement d'un composant, et notification à l'agent de RG des dépendances (composants requis)

4.4.2. *Le dossier médical personnalisé*

Considérons les différents éléments nécessaires à la mise en place de l'architecture informatique répartie pour le dossier médical personnalisé.

La sécurité repose d'une part sur l'authentification des patients et des personnels de santé pour le contrôle d'accès aux différentes ressources, d'autre part sur le cryptage des communications et des données des agents pour éviter la divulgation d'informations lors des échanges. Il est donc nécessaire de disposer de composants de communication cryptée et de migration cryptée. Le contrôle d'accès est par contre du niveau applicatif et peut être géré avec les solutions classiques de chiffrement symétrique ou asymétrique. Les différentes clés peuvent être contenues dans la carte Vitale du patient et dans le terminal portable des personnels de santé.

L'accès à travers différentes formes de réseaux et le mode déconnecté peuvent être gérés par les composants décrits précédemment.

La recherche d'information (dossier médical du patient, archives des personnels, références externes dans les centres universitaires, chez les fournisseurs d'équipements ou de médicaments…) peut être traitée par les solutions évoquées précédemment.

L'exploitation a posteriori de résultats d'examen nécessite la migration d'un agent vers l'appareil d'examen qui contient les données et ses spécifications techniques. Le composant d'exploitation des données évoqués précédemment répond parfaitement au besoin.

5. Conclusion

Notre contribution repose sur l'association des technologies **agent** (autonomie, mobilité, adaptation) et **composant** pour la construction et le déploiement de systèmes d'information répartis robustes, ouverts et adaptables. Notre *framework* permet une séparation claire entre les différents aspects fonctionnels (algorithmes de recherche et de traitement des données) et opératoires (communications cryptées...) ainsi que les aspects liés au déploiement du système de recherche. Le code métier, isolé dans les composants, est exécuté au sein d'agents qui fournissent l'implantation non fonctionnelle (leur conteneur).

D'une part, l'encapsulation des composants par des agents mobiles simplifie leur déploiement par délégation de la gestion de la mobilité et de l'installation au *middleware* (sans demander l'intervention du client ou du serveur), et permet l'adaptation dynamique du système à l'état des ressources matérielles et logicielles (en particulier aux évolutions des serveurs et de la structure du réseau qui, par nature, ne peuvent être contrôlées par les clients). D'autre part les propriétés d'adaptation des agents et leur nature proactive donne la flexibilité nécessaire à la construction des applications. En particulier, les agents mobiles permettent la découverte personnalisée et adaptative d'informations et de services tout en limitant le trafic sur le réseau. Le déplacement de composants spécifiques au client permet la spécialisation et la réactivité de la recherche et contribue à sa robustesse : recherche personnalisée, codage des résultats, routage dynamique des agents, découverte d'information et de serveurs, etc.

La séparation des préocupations associée à la réutilisation du *framework* et des composants, apporte une diminution des coûts de développement. Les expériences de conception et d'implémentation des exemples présentés ont permis de valider ces avantages. L'utilisation de JAVACT comme *middleware* à base d'agents mobiles adaptables, associée à des micro-composants spécifiques permet de concevoir des applications de façon plus simple et plus sûre qu'avec des techniques plus classiques.

Ces caractéristiques semblent intéressantes dans des domaines autres que les systèmes pair à pair. Dans celui de la **recherche d'information distribuée**, notre approche semble complémentaire aux solutions à base d'agents et de systèmes multi-agent plus classiques [FAN 99, CAZ 02] et compatible avec des stratégies de recherche sophistiquées. Dans le domaine de la **grille de calcul**, la possibilité de rechercher des composants, de vérifier leurs dépendances et éventuellement de télécharger récursivement les composants requis semble intéressante. Plus généralement, nous considérons les agents mobiles comme un outil privilégié pour la recherche et le déploiement de services et de logiciels sur les grilles ; nous allons mener des travaux dans cette voie. Pratiquement, nous allons expérimenter une application similaire à JAVANE issue de notre *framework* sur la plateforme de grille expérimentale française GRID'5000[29].

29. http://www.grid5000.org

Pour améliorer la sûreté des systèmes et en particulier dans le cadre critique des logiciels pour la gestion du dossier médical personnalisé, il est nécessaire d'introduire des mécanismes de validation de l'assemblage (dynamique) des composants de l'application comme du *middleware*. En complément, la définition de politiques de sécurité adaptées aux applications reste à étudier.

6. Bibliographie

[AGH 86] AGHA G., *Actors : a model of concurrent computation in distributed systems*, M.I.T. Press, Cambridge, Ma., 1986.

[ARC 04] ARCANGELI J.-P., HENNEBERT V., LERICHE S., MIGEON F., PANTEL M., « JAVACT 0.5.0 : principes, installation, utilisation et développement d'applications », rapport n° IRIT/2004-5-R, 2004, IRIT.

[BAB 02] BABAOGLU O., MELING H., MONTRESOR A., « Anthill : A Framework for the Development of Agent-Based Peer-to-Peer Systems », rapport n° UBLCS-2001-09, 2002, Dept. of Computer Science, Univ. of Bologna, Italy.

[BEL 04] BELLEFEMINE F., CAIRE G., TRUCCO T., RIMASSA G., « JADE programmer's guide (JADE3.2) », rapport, 2004, TILab S.p.A.

[BER 02] BERNARD G., ISMAIL L., « Apport des agents mobiles à l'exécution répartie », *Revue des sciences et technologies de l'information, série Techniques et science informatiques*, vol. 21, n° 6, 2002, p. 771-796, Hermès Science Publications, Lavoisier.

[BRE 99] BREWINGTON B., GRAY R., MOIZUMI K., KOTZ D., CYBENKO G., RUS D., « Mobile Agents in Distributed Information Retrieval », *Intelligent Information Agents*, Springer-Verlag, 1999.

[BRI 04] BRIOT J.-P., « Agents et composants : une dualité à explorer », transparents de présentation aux Journées Multi-Agents et Composants, JMAC 2004, novembre 2004, http ://csl.ensm-douai.fr/MAAC/3.

[CAZ 02] CAZALENS S., DESMONTILS E., JACQUIN C., LAMARRE P., « De la gestion locale à la recherche distribuée dans des sources d'informations et de connaissances », *Revue des sciences et technologies de l'information, série L'Objet*, vol. 8, n° 4, 2002, p. 47-69, Hermès Science Publications, Lavoisier.

[CHE 94] CHESS D., HARRISON C., KERSHENBAUM A., « Mobile Agents : Are They a Good Idea ? », rapport, 1994, IBM Research Division, New York.

[CLA 00] CLARKE I., SANDBERG O., WILEY B., HONG T., « Freenet : A Distributed Anonymous Information Storage and Retrieval System », *Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, International Computer Science Institute, 2000.

[DEC04] *DECOR'2004, Actes de la 1re Conférence Francophone sur le Déploiement et la (Re)Configuration de Logiciels*, octobre 2004, ISBN-2-7261-1276-5.

[FAN 99] FAN Y., GAUCH S., « Adaptive Agents for Information Gathering from Multiple Sources », *AAAI Symposium on Agents in Cyberspace*, 1999, p. 40-46.

[FUG 98] FUGGETTA A., PICCO G., VIGNA G., « Understanding Code Mobility », *IEEE Transactions on Software Engineering*, vol. 24, n° 5, 1998, p. 342-361.

[HAM 00] HAMMER J., FIEDLER J., « Using Mobile Crawlers to Search the Web Efficiently », *Int. Journal of Computer and Information Science*, vol. 1, n° 1, 2000, p. 36-58.

[HAN 04] HANDURUKAND S., KERMARREC A.-M., FESSANT F. L., MASSOULIÉ L., « Exploiting Semantic Clustering in the eDonkey P2P network », *SIGOPS European Workshop*, 2004, p. 109-114.

[KOS 01] KOSCH H., DOLLER M., BOSZORMENYI L., « Content-based Indexing and Retrieval supported by Mobile Agent Technology », *Second International Workshop on Multimedia Databases and Image Communication*, 2001, p. 152-166.

[LER 04a] LERICHE S., ARCANGELI J.-P., « Une architecture pour les agents mobiles adaptables », *Actes des Journées Composants JC'2004*, 2004.

[LER 04b] LERICHE S., ARCANGELI J.-P., PANTEL M., « Agents mobiles adaptables pour les systèmes d'information pair à pair hétérogènes et répartis », *Actes des NOuvelles TEchnologies de la REpartition*, 2004, p. 29-43.

[MAR 02] MARVIE R., PELLEGRINI M.-C., « Modèles de composants, un état de l'art », *Revue des sciences et technologies de l'information, série L'Objet*, vol. 8, n° 3, 2002, p. 61-89, Hermès Science Publications, Lavoisier.

[MIG 99] MIGEON F., « Etude et implantation de mécanismes réflexifs dans un langage concurrent », PhD thesis, Université Paul Sabatier, Toulouse, 1999.

[MIL 02] MILOJICIC D., KALOGERAKI V., LUKOSE R., NAGARAJA K., PRUYNE J., RICHARD B., ROLLINS S., XU Z., « Peer-to-Peer Computing », rapport n° HPL-2002-57, 2002, HP Laboratories Palo Alto.

[PAP 00] PAPASTAVROU S., SAMARAS G., PITOURA E., « Mobile Agents for World Wide Web Distributed Database Access », *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, n° 5, 2000, p. 802-820.

[RAT 01] RATNASAMY S., FRANCIS P., HANDLEY M., KARP R., SHENKER S., « A scalable content-addressable network », *Proceedings of the ACM SIGCOMM '01 Conference*, 2001, p. 161-172.

[SCH 01] SCHNEIER B., *Cryptographie appliquée, 2e édition*, Vuibert, Paris, 2001, ISBN : 2-7117-8676-5.

[STO 01] STOICA I., MORRIS R., KARGER D., KAASHOEK M. F., BALAKRISHNAN H., « Chord : A Scalable Peer-to-peer Lookup Service for Internet Applications », *Proceedings of the ACM SIGCOMM '01 Conference*, 2001, p. 149-160.

[SZY 02] SZYPERSKI C., *Component Software - Beyond Object-Oriented Programming*, Addison-Wesley / ACM Press, 2002.

[THA 01] THATI P., CHANG P.-H., AGHA G., « Crawlets : Agents for High Performance Web Search Engines », *Mobile Agents 2001, LNCS 2240*, Springer-Verlag, 2001, p. 119-134.

[TSC 99] TSCHUDIN C., « Mobile Agent Security », *Intelligent Information Agents*, Springer-Verlag, 1999.