Vers un modèle d'agent flexible

Sébastien Leriche — Jean-Paul Arcangeli

IRIT-UPS, Université Paul Sabatier
118 Route de Narbonne, F-31062 TOULOUSE CEDEX 9
{leriche, arcangel}@irit.fr

RÉSUMÉ. *Dans le cadre de nos travaux sur l'aide au développement d'applications réparties ouvertes, nous avons proposé un modèle d'agent mobile auto-adaptable ainsi que son implémentation au sein d'un middleware Java. Nous étudions actuellement une évolution de ce modèle afin de lui donner davantage de flexibilité. Ce papier présente le travail en cours. Nous discutons le besoin de flexibilité et la possibilité de définir différents modèles d'agents par le biais d'un méta-niveau configurable. Puis, nous présentons les éléments d'une architecture à base de micro-composants permettant l'adaptation dynamique.*

MOTS-CLÉS : *agents, composants, architecture logicielle, flexibilité, ouverture*

1. Introduction

Les évolutions du matériel conduisent de plus en plus à l'exécution d'applications dans des contextes répartis et hétérogènes (en termes de logiciel et de matériel -machines et réseau-) : par exemple sur l'Internet ou sur des grilles qui donnent accès à des quantités importantes de ressources, ou encore sur des équipements mobiles ou enfouis (PDA, PC portables, calculateurs embarqués...). Les ressources et les services y sont plus ou moins volatiles ; les conditions d'exécution varient (bande passante, disponibilité et localité des ressources) et sont partiellement ou mal connues au moment du développement. Aussi, pour offrir des services robustes et performants, ou pour en améliorer la qualité, les applications doivent pouvoir s'adapter dynamiquement. Ce besoin doit être pris en compte dès la conception et par conséquent il est nécessaire de disposer de technologies adéquates.

Le concept d'agent (voir par exemple la définition de J. Ferber (Ferber, 1995)) est particulièrement intéressant pour son niveau d'abstraction. Nous appelons *agent*

logiciel une entité autonome capable de communiquer, disposant de connaissances et d'un comportement privé, ainsi que d'une capacité d'exécution propre. Un agent (logiciel) agit pour le compte d'un tiers (un autre agent, un utilisateur) qu'il représente sans être obligatoirement connecté à lui. Comme les composants logiciels (Szyperski, 2002; Oussalah, 2005), les agents logiciels sont issus du concept d'objet et constituent des briques de base pour la construction des applications. Un *agent mobile* (Fuggetta *et al.*, 1998; Bernard *et al.*, 2002) est un agent logiciel qui peut se déplacer sur un réseau avec son code et ses données propres, mais aussi avec son état d'exécution. Le but peut être de localiser une ressource ou de rapprocher un traitement des données pour réduire la consommation de bande passante ou encore d'adapter un service distant aux besoins et aux capacités du client.

L'utilisation d'agents, éventuellement mobiles, contribue à la flexibilité des applications. Nous avons étudié et proposé des technologies de niveau modèle de programmation et *middleware* pour aider au développement d'applications réparties auto-adaptables : au niveau des agents, l'auto-adaptation repose sur la mobilité proactive ainsi que sur l'évolution fonctionnelle et opératoire. Dans cette communication, nous présentons un travail en cours dont l'objectif est d'augmenter le niveau de flexibilité intra-agent : nous cherchons à définir un modèle et une architecture d'agent flexible (ceci à partir d'un modèle d'agent que nous avons précédemment défini et qui est décrit en section 2.1).

En section 2, nous discutons les besoins en termes d'évolution d'un modèle existant et de définition de différents modèles d'agents. En section 3, nous décrivons les principes architecturaux et nous proposons quelques élements pour la mise en oeuvre. En section 4, nous discutons les limites en termes de flexibilité intra-agent et nous situons notre approche par rapport d'autres (principalement MAST de L. Vercouter). Enfin, nous concluons en section 5.

2. Evaluation des besoins

2.1. *Un premier modèle d'agent adaptable*

Les agents mobiles sont, par nature, des outils d'adaptation en contexte réparti. Afin d'accroître les possibilités d'adaptation, nous avons proposé un niveau de flexibilité intra-agent contrôlé par l'agent lui-même et nous avons défini un modèle d'agent mobile auto-adaptable dynamiquement (Leriche *et al.*, 2004a) : conformément au principe de séparation des préoccupations, le modèle repose sur la réification d'un ensemble de mécanismes d'exécution internes non fonctionnels (envoi des messages, réception, déplacement, cycle de vie…) sous la forme de micro-composants dynamiquement interchangeables. Ces micro-composants constituent un méta-niveau (cf. 3.1.1) de l'agent ; ils opèrent par délégation de son code fonctionnel (niveau de base). Ce modèle d'agent a été mis en œuvre dans le *middleware* JAVACT [1] que nous dé-

1. http://www.irit.fr/free.html

veloppons. Il a été validé expérimentalement dans le cadre de plusieurs applications : prototypes de logiciels pair à pair pour le partage et l'échange de fichiers (Leriche *et al.*, 2004b), distribution de correctifs de sécurité, messagerie instantanée mobile, jointures en base de données réparties... Différents micro-composants ont été développés et réutilisés : cryptage des communications, tolérance aux déconnexions[2], localisation d'agents mobiles...

La séparation des préoccupations en différents modules de code indépendants est une propriété essentielle qui doit être préservée : elle simplifie le développement des différents composants (indépendamment les uns des autres) et leur réutilisation.

2.2. *Vers davantage de flexibilité*

Prenons l'exemple de notre prototype de logiciel pair à pair cité précédemment. Il est implanté par un système d'agents répartis. Certains agents sont des agents d'interface (avec les pairs clients ou serveurs). Certains agents sont mobiles sur le réseau et peuvent être amenés à percevoir leur environnement physique (charge du réseau, niveau de sécurité...) ; pour s'y adapter, ils peuvent redéfinir leurs mécanismes de communication ou de déplacement. Certains agents peuvent développer une algorithmique complexe (recherche de ressource, exploitation d'une ressource...). En termes d'évolution, de décision, de perception, de réactivité et de mobilité, les différents agents qui constituent le système ont des caractéristiques et des besoins variables. Or, dans notre implantation, tous ont du être développés à partir du modèle d'agent unique offert par le *middleware*.

Plus précisément, dans la proposition décrite en 2.1 et mise en œuvre dans le *middleware* JAVACT , l'adaptation se réduit au choix initial des micro-composants et au remplacement d'un micro-composant par un autre qui offre le même service (spécifié par une interface Java). L'évolution des services opératoires est limitée par une interface fixe, et il n'est pas possible de décider de quels micro-composants sont remplaçables et de quels ne le sont pas. L'architecture de l'agent est également figée : les agents sont des *acteurs*[3] (Hewitt, 1977; Agha, 1986) répartis et mobiles et il n'est possible ni d'introduire un nouveau micro-composant ni de retirer un micro-composant inutile (par exemple, le gestionnaire de déplacement pour un agent immobile).

D'autre part, on peut aussi noter que notre expérience du développement du *middleware* JAVACT nous a conduit à réviser plusieurs fois l'architecture des agents et à modifier le *middleware*. Par exemple :

– lors de l'introduction de la mobilité, il a fallu ajouter au méta-niveau un micro-composant dédié (il interagit avec d'autres composants de méta-niveau) et donc modifier l'architecture des agents ;

2. Pour cela, il a fallu développer un composant de perception de l'environnement réseau.
3. L'acteur est un modèle concret d'agent logiciel : communication par messages asynchrones, traitement des messages en série, changement de comportement...

– pour mettre en œuvre un mécanisme de communication synchrone, nous avons dû modifier les interfaces existantes, en l'occurrence celle des micro-composants de communication ;

– nos agents n'ont pu être utilisés tels quels pour implanter des agents AMAS (Capera *et al.*, 2003) et une nouvelle architecture d'agent a été définie (Déjean, 2003).

Si les expérimentations effectuées ont permis de vérifier l'intérêt de l'architecture proposée, celle-ci reste donc limitée en terme de flexibilité : elle n'est pas minimale (si un agent n'effectue pas certaines opérations, par exemple changement de comportement ou déplacement, tous les micro-composants lui sont quand même associés) et ne permet pas de modeler et de personnaliser la structure des agents. Par exemple, il n'est pas possible de définir un agent immobile ou un type quelconque d'agent doté de mécanismes particuliers.

2.3. *Différents types d'agents*

Dans une démarche de classification, on peut être amené à distinguer les agents suivant différentes propriétés (ou composantes) indépendantes, à caractère individuel ou social :

– l'autonomie, déclinée diversement dans différents modèles d'agent et leur implantation, qui induit les notions de cycle de vie et d'activité ;

– le savoir et le savoir-faire (éventuellement offert sous forme de services) ;

– une capacité de décision ;

– l'évolutivité (apprentissage...) ;

– la possibilité d'engendrer de nouveaux agents ;

– la communication ;

– l'interaction avec l'environnement (perception...) dans lequel l'agent est situé (par exemple, un agent mobile est situé géographiquement sur un réseau de machines et interagit avec lui *via* un ensemble des services) ;

– la réactivité[4] ;

– ...

La notion de cycle de vie se retrouve dans tous les modèles sous des formes différentes, et la plupart des agents ont besoin de moyens de communication respectant certains standards (invocations de services distribués, KQML, FIPA-ACL...). Les différentes propriétés d'un agent définissent son type : par exemple, les agents BDI (Rao

4. Un agent est réactif s'il répond de manière opportune aux changements de son environnement issus de stimuli externes. Il n'a pas besoin d'une représentation symbolique élevée de la perception son environnement.

et al., 1995), les agents AMAS, les agents de déploiement (Hall *et al.*, 1997), les agents conteneurs[5], etc.

	Com. asynchrone	Création	Com. synchrone	Mobilité	Perception	Décision	Cycle de vie	...
Acteur	*	*					*	
Agent logiciel	*	*	*				*	
Agent mobile	*	*		*			*	
Agent déploiement	*			*			*	
Agent conteneur	*	*	*	*			*	
Agent réactif		*			*	*	*	
Agent BDI	*	*	*		*	*	*	
Agent AMAS	*	*	*		*	*	*	
...								

Figure 1. *Quelques agents et leurs composantes*

Au sein d'une application, on peut trouver des agents de types différents voire certaines formes d'agents hybrides possédant des composantes issues de différents types. On peut s'interroger sur l'intérêt qui y aurait (pour faciliter le développement) à pouvoir définir des types d'agents « à la carte ». Nous pensons que certains agents pourraient bénéficier de composantes issus d'autres modèles, pour obtenir par exemple des agents AMAS mobiles ou des acteurs sans changement de comportement. La réutilisabilité des composantes est un de nos objectifs.

3. Architecture flexible d'agent logiciel

Nous pensons que la flexibilité de l'architecture doit permettre non seulement le remplacement dynamique de composants, mais aussi la définition de structures d'agent personnalisées (éventuellement l'ajout de nouveaux composants à une structure existante et symétriquement le retrait). Notre idée est de :

5. L'agent mobile adaptable peut être vu comme un support de déploiement au sens de (Carzaniga *et al.*, 1998) pour les comportements (composants fonctionnels) : d'une part la mobilité d'agent permet un transport adaptatif des composants fonctionnels sur le réseau, d'autre part le méta-niveau joue le rôle de conteneur pour le composant fonctionnel : il lui fournit les services nécessaires à son exécution et permet sa configuration (choix des micro-composants) et sa reconfiguration dynamique (Arcangeli *et al.*, 2006).

– permettre la définition de différents modèles (ou types) d'agents sous forme d'un ensemble de mécanismes primitifs (des types de micro-composants) conférés à l'agent ;

– puis de configurer les agents par assemblage de composants sur ce modèle ;

– et enfin de produire une architecture minimale à partir des besoins réels exprimés au niveau de base.

3.1. *Principes architecturaux*

3.1.1. *Réflexivité*

Un système logiciel est dit réflexif lorsqu'il est capable d'inspecter et de modifier sa structure et sa sémantique, de la même manière qu'il opère pour son domaine applicatif. C'est une technique générale pour implanter des mécanismes d'adaptation dynamique (Boyer *et al.*, 2001). En effet, pour faire les bons choix de configuration pour s'adapter au contexte d'exécution, il faut connaître d'abord l'état de l'environnement, mais aussi son propre état et sa structure. On appelle *niveau de base* le niveau habituel de programmation d'une entité, et *méta-niveau* celui qui concerne sa description et dans lequel on réalisera des transformations. Cette séparation entre niveaux permet de rendre transparent au niveau de base un changement structurel ou opératoire dans le méta-niveau.

3.1.2. *Micro-composants*

Pour spécialiser le méta-niveau d'un agent et le doter de nouveaux mécanismes opératoires, une première approche consisterait à procéder par héritage. Mais, dans ce cas, il serait impossible d'adapter dynamiquement les mécanismes internes des agents. L'approche par délégation, bien plus flexible, s'impose. Le paradigme composant (modularité et mécanismes d'assemblage) nous semble un bon candidat pour supporter le codage des différentes composantes des agents identifiées plus haut.

Les agents sont alors construits par assemblage de composants (ce sont donc des composites). Nous avons proposé (Leriche *et al.*, 2004a) de placer les composants fonctionnels au niveau de base, tandis que les composants non-fonctionnels sont regroupés dans le méta-niveau. Ces derniers sont des réifications de mécanismes opératoires spécifiques de l'agent (communication, mobilité...) ; ils sont placés au méta-niveau pour faciliter leur adaptation dynamique via la réflexivité de l'architecture. Pour une identification précise des rôles et donc pour simplifier la validation des assemblages, nous proposons l'utilisation de micro-composants de granularité minimale c'est-à-dire n'offrant qu'un service opératoire unique.

3.1.3. *Architecture composant/connecteur*

Le point central de l'architecture d'agent, le *contrôleur*, est une forme de connecteur. Il permet de faire le lien entre le niveau de base et le méta-niveau ainsi qu'entre les micro-composants du méta-niveau. Il supporte également l'adaptation dynamique

par la modification des liens avec les composants. Pour obtenir la flexibilité souhaitée et que chaque micro-composant possède une interface différente, le contrôleur devrait contenir autant de connexions (ou de ports) que de types de composants possibles. Pour un ensemble donné de types de composants, il faut donc un contrôleur *ad hoc*.

3.1.4. *Minimalité*

Pour répondre au problème précédent, nous proposons de dériver du code fonctionnel de l'agent un type appelé *type réduit* (ou type d'implantation), qui correspond à un ensemble de types de composants effectivement utiles (ou encore aux mécanismes strictement nécessaires à l'exécution). A partir du type réduit identifié, nous proposons de générer automatiquement le contrôleur *ad hoc* associé.

Dans le cas le plus complexe d'un agent qui pourrait changer de comportement fonctionnel, le type réduit pourrait être représenté par un diagramme états/transitions (cf. figure 2) : les transitions correspondraient aux changements de comportement et chaque état représenterait le type réduit pour un comportement.

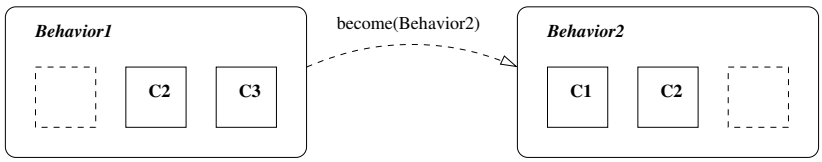

Figure 2. *Type réduit d'agent, représenté par un diagramme état/transition*

Cette minimalité de l'architecture a des avantages en terme d'efficacité à l'exécution (d'autant plus si les agents se déplacent sur le réseau), mais aussi en termes de sûreté et de sécurité car la vérification et la validation des assemblages (dépendances...) est facilitée. Par ailleurs, il n'est pas nécessaire d'expliciter les services de méta-niveau (mécanismes d'exécution) requis par le code fonctionnel puisqu'ils sont directement déduits du code (de même, dans une étape ultérieure, pour les services requis entre micro-composants).

3.2. *Mise en œuvre (en Java)*

Dans un premier temps, pour un *pool* de composants donnés, un contrôleur et une classe abstraite seront engendrés. Dans la classe abstraite, sera implantée une méthode par service de composant ; le corps de cette méthode sera en réalité une indirection vers le contrôleur qui redirigera le traitement vers le composant adéquat. En héritant de la classe abstraite, le code fonctionnel aura accès de manière transparente à tous les services du méta-niveau, sans construction syntaxique particulière.

Après compilation des parties fonctionnelles, un outil en extraira (par rétro-ingénierie) l'ensemble des services non-fonctionnels réellement nécessaires à l'exé-

cution de l'agent, donc par extension le type des micro-composants à instancier dans l'architecture (ou encore le type réduit de l'agent). Il générera ensuite le code du contrôleur minimal *ad hoc*, et pourra effectuer un ensemble de vérifications de cohérence.

Pour réaliser ces étapes, nous utiliserons le framework de manipulation de bytecode Java ASM[6] ainsi que les outils de réflexivité de Java.

4. Discussion

4.1. *Limitations*

Le problème de la « mutation » des agents, *i.e.* le changement dynamique de type, reste ouvert. Nous pensons qu'il serait préférable de ne pas autoriser la mutation si on veut obtenir des systèmes fiables et vérifiables a priori (on pourrait cependant autoriser les mutations prévues lors de la conception). L'adoption dynamique par un agent d'un micro-composant non prévu ne nous semble pas justifiée (un agent pourrait découvrir par lui-même la possibilité de « se bonifier » en changeant ses mécanismes opératoires mais, à notre sens, cela ne pourrait résulter que de capacités d'introspection beaucoup plus élévées que celles que nous envisageons aujourd'hui).

Néanmoins, il est envisageable qu'un agent puisse acquérir dynamiquement un comportement fonctionnel nouveau (dans le cas d'une adaptation fonctionnelle) qu'il aura, par exemple, reçu par message (évolution par changement de comportement dans le cas des acteurs de K. Hewitt et G. Agha). Pour que ce comportement soit effectivement exécutable, il faut que les micro-composants opératoires nécessaires soient disponibles au méta-niveau ; s'ils ne le sont pas, il faut reconfigurer le méta-niveau à partir de nouveaux micro-composants et redéfinir ainsi dynamiquement le type de l'agent.

4.2. *Travaux connexes*

Il existe de très nombreux environnements de développement orientés agents (Jade[7], Actor Foundry[8], MadKit[9], Jack[10]...), mais la majorité est spécifique à une forme particulière d'agent. Par ailleurs, les modèles de composants utilisables aujourd'hui (CCM[11], EJB[12], Fractal[13]...) sont plus adaptés à la conception de gros composants métiers qu'à la conception d'agents à base de composants de petite taille. Nous

6. http://asm.objectweb.org
7. http://jade.tilab.com
8. http://yangtze.cs.uiuc.edu/foundry
9. http://www.madkit.org
10. http://www.agent-software.com/shared/products/index.html
11. http://www.omg.org/technology/documents/formal/components.htm
12. http://java.sun.com/products/ejb
13. http://fractal.objectweb.org

comparons notre approche avec celle de MAST, qui semble très proche de notre proposition.

MAST[14] est un environnement pour le développement et le déploiement d'applications multi-agent. Il fournit des outils pour la conception d'agents par assemblage de composants génériques. MAST propose un modèle de composants spécifique (Vercouter, 2004), avec pour objectif de permettre l'ajout et le retrait de composants sans perturber le fonctionnement de l'agent, et sans intervention extérieure (autonomie de l'agent). Les composants sont assemblés sur un noyau d'agent, qui leur retransmet des évènements auxquels ils sont abonnés. Chaque composant spécifie des ensembles de rôles abstraits fournis et requis, permettant d'en faire un assemblage correct. La méthodologie pour le déploiement des agents est issue de celle des composants répartis : schémas de déploiement spécifié dans un fichier XML et outil de déploiement spécifique.

Notre approche diffère de MAST sur plusieurs points.

– Il est possible d'enlever ou d'ajouter n'importe quel composant à l'exécution dans MAST, sans perturber le fonctionnement des autres composants, mais il n'est pas possible de vérifier si les assemblages sont corrects : par exemple, on peut enlever le composant de communication mais l'agent risque de fonctionner en étant isolé... Nous souhaitons empêcher ce type de faute à l'exécution.

– Les composants de MAST doivent être identifiables dans une méthodologie spécifiques aux SMA (approche *Voyelles*), nous ne souhaitons pas limiter nos composants à un tel ensemble, par exemple nous souhaitons pouvoir modifier le composant de cycle de vie, ou introduire de nouveaux composants qui n'ont rien de spécifique aux SMA (mobilité...).

– Dans MAST, tous les composants de l'agent sont au même niveau, alors que dans notre modèle les composants non-fonctionnels sont placés dans un méta-niveau pour permettre l'introspection et l'intercession sur la structure de l'agent afin de permettre l'adaptation dynamique de l'architecture de l'agent.

– Dans notre proposition, les composants sont des micro-composants, de très petite granularité (service unique), afin de faciliter les validations sur les assemblages.

– L'un des objectifs de notre modèle d'agent est de pouvoir simplifier le développement d'applications réparties. Notre solution permet aux agents de manipuler la répartition et la mobilité, au moyen d'abstractions simples disponibles au niveau de base. Ainsi, il n'est pas nécessaire de prévoir des outils et des schémas de déploiement des agents. De même, les opérations de communication sont réalisées dans des composants de méta-niveau et il suffit d'un appel de méthode au niveau de base pour envoyer un message. La même opération est plus compliquée et moins lisible dans MAST, malgré l'utilisation d'outils.

Pour ces raisons, nous rejoignons les préoccupations de flexibilité de MAST ainsi que certains principes architecturaux (composants...), mais en proposant un modèle

14. http://www.emse.fr/~vercouter/mast

d'architecture différent, plus propice à la conception d'agents logiciels distribués de natures distinctes, plus simple pour le concepteur de l'application à base d'agents et pour lequel il sera possible de vérifier des propriétés de sûreté, notamment à propos de l'assemblage des composants.

5. Conclusion

Dans ce papier, nous présentons un travail en cours dont l'objectif est de simplifier le développement d'applications réparties ouvertes à base d'agents en fournissant un modèle de programmation adapté (basé sur la séparation des préoccupations) ainsi que les outils de niveau *middleware* correspondants. Sur le plan de la méthodologie de développement, notre proposition doit permettre au concepteur de :

– définir un modèle d'agent adapté au besoin applicatif,

– configurer un agent sur ce modèle, *via* la définition ou la réutilisation de composants logiciels (*plug-in*) de grain fin, et valider la composition,

– reconfigurer l'agent en cours d'exécution en changeant les composants.

Dans un travail précédent, nous avons déjà validé un certain nombre d'éléments. Il nous faut maintenant mettre en oeuvre la solution proposée dans sa totalité. Le développement d'un prototype d'agent flexible permettra d'évaluer expérimentalement l'apport au développement ainsi que les limites. En outre, il nous faudra proposer des solutions pour la validation des architectures d'agent, à la fois pour des raisons de sûreté et de sécurité.

6. Bibliographie

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

Arcangeli J.-P., Leriche S., Pantel M., « Construction et déploiement de systèmes d'information répartis ouverts et adaptables au moyen d'agents mobiles et de composants », *Revue des sciences et technologies de l'information, série L'Objet*, 2006. à paraître.

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, p. 771-796, 2002.

Boyer F., Charra O., « Utilisation de la réflexivité dans les plate-formes adaptables pour applications réparties », *Revue électronique sur les Réseaux et l'Informatique Répartie*, 2001.

Capera D., George J.-P., Gleizes M.-P., Glize P., « The AMAS theory for complex problem solving based on self-organizing cooperative agents », *Twelfth International Workshop on Enabling Technologies : Infrastructure for Collaborative Enterprises*, 2003.

Carzaniga A., Fuggetta A., Hall R. S., van der Hoek A., Heimbigner D., Wolf A. L., A Characterization Framework for Software Deployment Technologies, Technical Report n° CU-CS-857-98, Dept. of Computer Science, University of Colorado, April, 1998.

Déjean T., Développement d'applications multi-agents adaptatifs avec l'API JavAct, Technical report, Institut de Recherche en Informatique de Toulouse, 2003.

Ferber J., *Les systèmes multi-agents : vers une intelligence collective*, Interéditions, 1995.

Fuggetta A., Picco G., Vigna G., « Understanding Code Mobility », *IEEE Transactions on Software Engineering*, vol. 24, n° 5, p. 342-361, 1998.

Hall R., Heimbigner D., van der Hoek A., Wolf A., The Software Dock : A Distributed, Agent-based Software Deployment System, Technical Report n° CU-CS-832-97, Department of Computer Science, University of Colorado, 1997.

Hewitt C. E., « Viewing control structure as patterns of passing messages », *Journal of Artificial Intelligence*, vol. 8, n° 3, p. 323-364, 1977.

Leriche S., Arcangeli J.-P., « Une architecture pour les agents mobiles adaptables », *Actes des Journées Composants JC'2004*, 2004a.

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*, p. 29-43, 2004b.

Oussalah M., *Ingenierie des Composants : Concepts, techniques et outils*, Vuibert Informatique, 2005. Ouvrage collectif.

Rao A. S., Georgeff M. P., BDI agents : From Theory to Practice, Technical Report n° 56, Australian Artificial Intelligence Institute, Melbourne, Australia, 1995.

Szyperski C., *Component Software - Beyond Object-Oriented Programming*, Addison-Wesley / ACM Press, 2002.

Vercouter L., « MAST : Un modèle de composant pour la conception de SMA », *Actes des Journées Systèmes Multi-Agents et Composants*, 2004.