Assemblage Automatique de Composants pour la Construction d'Agents avec MADCAR

G. Grondin[*,] — N. Bouraqadi[*] — L. Vercouter[**]**

[*] *Dépt. GIP, Ecole des Mines de Douai*
941, rue Charles Bourseul – B.P. 10838, 59508 Douai Cedex, France
{grondin,bouraqadi}@ensm-douai.fr

[**] *Dépt. G2I, Ecole des Mines de Saint-Étienne*
158 cours Fauriel, 42023 Saint-Étienne cedex 02, France
vercouter@emse.fr

RÉSUMÉ. *Dans cet article, nous proposons d'utiliser le modèle* MADCAR *pour faciliter la conception d'agents à base de composants.* MADCAR *est un modèle abstrait pour construire des moteurs d'assemblage dynamique et automatique d'applications à base de composants. Ici, l'application à modéliser est un agent capable de se ré-assembler à la suite d'un changement contextuel ou à la demande du concepteur. Dans cet agent, nous séparons son comportement normal (matérialisé par un assemblage de composants) de son comportement d'adaptation. Pour ce faire, nous intégrons à cet agent un moteur permettant de déclencher, décider et réaliser des ré-assemblages de composants. Le processus d'assemblage est piloté par une politique d'assemblage explicite, précise et ouverte, qui doit être spécifiée par le concepteur de l'agent.*

ABSTRACT. *In this article, we suggest using the* MADCAR *model to facilitate the design of component-based agents.* MADCAR *is an abstract model for building engines that dynamically and automatically (re-)assemble component-based applications. We use such an engine to define a model of agent able to be re-assembled following contextual changes or the designer's requests. One of the main advantages of* MADCAR *lies in the specification of the assemblies with both flexibility and precision. The assembling process is controlled by an explicit policy of assembling. Last, by using* MADCAR*, our agent model aims at being open so as to be usable in various application domains.*

MOTS-CLÉS : *Conception d'agents, Adaptation d'agents, Assemblage automatique.*

KEYWORDS: *Agent design, Agent adaptation, Automatic assembling.*

1. Introduction

Dans les travaux sur la conception d'agents, l'approche à base de composants est avant tout utilisée comme un cadre pour la spécification du comportement de l'agent. En effet, les composants permettent d'implémenter des capacités qui sont souvent requises chez les agents (communication, perception, planification,...). Mais, trop souvent, l'assemblage des composants doit se faire manuellement et les possibilités d'évolution de cet assemblage ne sont pas prévues.

Un modèle d'agents à base de composants et des mécanismes de ré-assemblage doivent être définis pour permettre la conception d'agents susceptibles d'évoluer régulièrement. Trois propriétés sont notamment recherchées pour ce mécanisme : l'ouverture, l'adaptabilité et l'autonomie. L'**ouverture** fait référence à la possibilité d'ajouter des composants même s'il n'était pas initialement prévu de les ajouter (i.e. après la phase de conception). L'**adaptabilité** fait référence à la capacité de modifier le fonctionnement d'un agent pour prendre en compte certains changements notables dans son contexte. Enfin, la propriété d'**autonomie** fait référence à la faculté des agents à fonctionner sans intervention humaine, en particulier en ce qui concerne les ré-assemblages.

Dans cet article, nous proposons de construire des agents en utilisant le modèle MADCAR[1]. MADCAR est un modèle abstrait qui a pour objectif la reconfiguration dynamique et automatique d'applications à base de composants. Nous voulons utiliser MADCAR pour automatiser la construction d'agents à base de composants grâce à ses mécanismes de (ré-)assemblage. C'est pourquoi, nous proposons un modèle d'agents où le comportement d'adaptation des agents est séparé de leur comportement normal (implémenté par un assemblage de composants). Nous intégrons à chaque agent un moteur conçu avec MADCAR, permettant de déclencher, décider et réaliser des ré-assemblages de composants. Le processus d'assemblage de ce moteur est piloté par une politique d'assemblage, faiblement couplée aux composants de l'agent, et qui doit être spécifiée par le concepteur de l'agent. En résumé, nous proposons un modèle d'agents auto-adaptables, et dont le comportement d'adaptation peut être spécifié de façon à la fois flexible et précise.

La section 2 décrit les besoins pour l'aide à la conception d'agents en se basant sur un environnement de développement existant : MAST. La section 3 présente de manière succincte le modèle MADCAR, et son utilisation pour l'assemblage automatique d'agents. La section 4 illustre le processus de ré-assemblage de MADCAR dans un exemple visant à adapter un agent. Enfin, la section 5 récapitule les principaux apports de MADCAR pour l'aide à la conception d'agents.

---

1. M*odel for* A*utomatic and* D*ynamic* C*omponent* A*ssembly* R*econfiguration.*

2. Le point sur la construction d'agents par assemblage de composants

Des travaux relativement récents se sont intéressés à la construction d'agents par assemblage de composants. La programmation par composants est en effet intéressante pour ce type de développement car les agents présentent souvent des capacités proches d'une application à une autre (communication, perception, planification,...). Dans ce cas, la construction d'un agent consiste à *assembler des composants pré-existants qui sont chacun chargé d'implémenter une partie bien spécifique du comportement de l'agent*. En d'autres termes, un agent est une entité logicielle contenant un assemblage de composants qui matérialise son comportement[2]. Le développement d'applications multi-agents bénéficierait grandement de l'utilisation d'une bibliothèque de composants implantant ces capacités partagées. Plusieurs environnements de développement ont ainsi adopté une approche par composants (Brazier *et al.*, 2002; Occello *et al.*, 2002; Ricordel *et al.*, 2002; Vercouter, 2004).

Nous allons mettre en évidence des possibilités d'amélioration de l'approche par composants pour concevoir des agents en discutant des avantages et des limites d'un de ces travaux. L'environnement de développement MAST (*Multi-Agent System Toolkit*) intègre un modèle de composant spécifique permettant la modification automatique d'un assemblage (Vercouter, 2004).

Un des principaux avantages de MAST est de permettre la connexion automatique et flexible des composants qui constituent un agent. En effet, les composants d'un agent ne sont pas directement connectés entre eux mais à un noyau commun qui joue un rôle de bus logiciel pour assurer la communication entre les composants. Le noyau reçoit des événements émis par un composant et se base sur une description sémantique de chaque composant pour déterminer le ou les composant(s) à qui il transmet l'événement. Cette flexibilité du mode d'interaction facilite le rôle du concepteur de l'agent lorsque des composants sont ajoutés ou supprimés, car il n'est pas nécessaire de connecter explicitement les entrées/sorties des composants. Ces mécanismes montrent l'importance de l'**automatisation** du processus d'assemblage.

Cependant, ces spécificités qui facilitent la conception d'agents dans MAST ne sont pas sans poser un certain nombre d'inconvénients. En effet, les difficultés qui sont épargnées au concepteur de l'agent se trouvent reportées chez le concepteur des composants. Ce dernier doit fournir aux composants une riche description de leurs capacités d'assemblage notamment sur la base des événements qu'ils prennent en charge. Une partie des informations d'assemblage est ainsi déportée dans la description du composant, ce qui dénote un manque d'**abstraction** alors que les mécanismes d'assemblage doivent pouvoir être spécifiés sans que cela affecte directement les composants.

Par ailleurs, il est important de souligner le manque de contrôle de l'assemblage de composants dans MAST. Les connexions entre composants n'étant pas concrètement construites mais déduites des interfaces des composants, il n'est pas évident de prévoir

2. Remarquons qu'un agent contenant des composants ne doit pas forcément être un composant.

l'impact de chaque modification de l'assemblage. Ainsi, la flexibilité de l'assemblage ne doit pas se faire au détriment du fonctionnement de l'agent. La problématique d'assemblage est transversale aux composants de l'agent car elle peut concerner l'ensemble des composants. Cela montre que le processus d'assemblage doit être géré de manière **explicite** afin de pouvoir spécifier les conditions d'assemblage avec précision, quel que soit le sous-ensemble de composants concernés.

3. Construction et adaptation d'agents avec MADCAR

MADCAR est un modèle abstrait de moteurs d'assemblage dynamique et automatique d'applications à base de composants. Nous montrons dans cette section comment il peut être exploité pour la construction et l'adaptation d'agents. Les seules hypothèses que nous faisons sur les *modèles de composants utilisables dans* MADCAR sont la spécification explicite, pour chaque composant, de leurs interfaces fournies et requises ainsi que de leurs attributs [3].

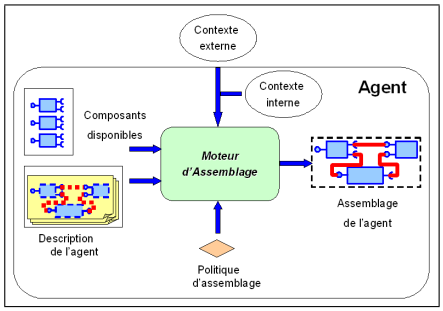


Figure 1. *Structure d'un agent avec* MADCAR

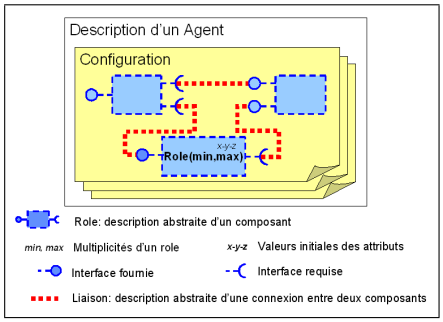


Figure 2. *Description d'un agent avec* MADCAR

3.1. *Structure d'un agent avec* MADCAR

Un moteur d'assemblage basé sur MADCAR permet de construire ou d'adapter un logiciel à partir de quatre entrées : un *ensemble de composants* à assembler, une *description du logiciel* qui représente la spécification de l'assemblage, une *politique d'assemblage* qui dirige les décisions d'assemblage et un *contexte* dont les changements déclenchent le ré-assemblage. La figure 1 représente un tel moteur avec ses différentes entrées/sortie dans le cas où le logiciel à (ré-)assembler est un agent. Un tel agent est constitué du moteur d'assemblage, de ses différentes entrées, ainsi que de l'assemblage résultat. Notez que le contexte est scindé en deux parties : l'une est in-

3. Le mariage entre notre modèle d'agents et le concept de « composant composite » est en cours d'étude.

terne et représente l'état de l'agent et l'autre est externe et représente l'état du monde dans lequel évolue l'agent.

Le choix de placer le moteur d'assemblage et ses différentes entrées à l'intérieur de l'agent est motivé par la préservation de la propriété d'autonomie. En effet, l'agent reste « maître » de sa propre structure et de ses évolutions.

3.2. *Description d'un agent avec* MADCAR

La figure 2 représente une description d'un agent avec MADCAR. Cette description regroupe un ensemble de **configurations** alternatives. Une configuration représente un assemblage de rôles. Un **rôle** est une description abstraite de composants, constituée d'un ensemble de *contrats* (Meyer, 1992; Beugnard *et al.*, 1999). Ces contrats doivent au moins spécifier (1) un ensemble d'interfaces (fournies ou requises) symbolisant ses éventuelles interactions avec les autres rôles, (2) un ensemble d'attributs dont les valeurs permettent d'initialiser les composants, et (3) deux multiplicités. Les *multiplicités* d'un rôle représentent le nombre minimal (min où $min \geq 0$) et le nombre maximal (max où $max \geq min$) de composants qui peuvent remplir ce rôle simultanément.

Les configurations de MADCAR diffèrent du concept de configuration utilisé dans les ADLs[4] (Fuxman, 2000; Medvidovic *et al.*, 1997). En effet, chacune de nos configurations décrit un *ensemble d'assemblages possibles* structurellement équivalents puisqu'ils sont basés sur le même graphe de rôles. Le concept de rôle dans MADCAR permet le découplage total entre l'architecture de l'agent (décrite par la configuration en cours) et les composants qui la composent. De plus, la multiplicité des rôles offre la possibilité d'avoir un nombre plus ou moins grand de composants utilisés dans des assemblages issus de la même configuration.

3.3. *Processus de (re-)assemblage de* MADCAR

Un moteur basé sur MADCAR peut aussi bien être utilisé pour créer automatiquement un assemblage à partir de composants déconnectés, que pour adapter dynamiquement un assemblage existant. En effet, le même processus est réalisé dans les deux cas. Ce processus se décompose en cinq étapes successives.

1) Le (ré-)assemblage est déclenché lorsque des événements pertinents sont détectés. C'est le cas par exemple lors de l'ajout de configurations et/ou de composants.

2) Le moteur identifie les compatibilités entre les composants présents et les rôles contenus dans les configurations fournies : il s'agit de déterminer pour chaque composant s'il peut satisfaire les contrats définis dans les rôles.

4. *Architecture Description Languages.*

3) Le moteur sélectionne une configuration selon la politique d'assemblage et les compatibilités identifiées dans l'étape précédente.

4) Le moteur sélectionne les composants à connecter selon la configuration choisie et sur la base de la politique d'assemblage.

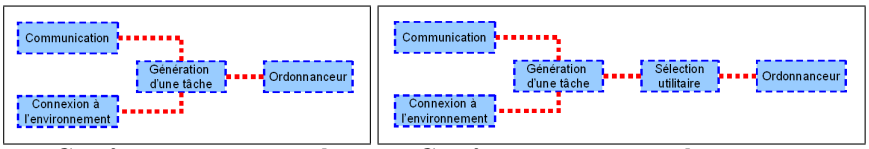
5) Enfin, le moteur réalise l'assemblage des composants sélectionnés selon la configuration choisie.

4. Exemple de ré-assemblage d'agent

Dans cette section, nous illustrons le processus d'assemblage décrit dans la section 3.3 par un scénario de ré-assemblage d'agents. Ce scénario est basé sur une application inspirée de celle décrite dans (Bourne *et al.*, 2000) et déjà utilisée dans (Vercouter, 2004) pour la conception d'agents par assemblage de composants.

4.1. *Cadre du scénario*

L'application est constituée d'une grille sur laquelle sont positionnés plusieurs agents qui accomplissent des tâches qui sont réparties sur la grille. Chaque tâche est caractérisée par une durée d'exécution, une échéance et une récompense. Certaines tâches, dites coopératives, ne peuvent être exécutées que lorsqu'un nombre d'agents suffisant pour la réaliser sont présents sur sa case. Dans ce cas, ces agents se partagent équitablement la récompense. Les agents qui souhaitent exécuter une tâche coopérative peuvent communiquer avec d'autres agents pour les inviter à les rejoindre.



$Config_1$: « Agent Social » $Config_2$: « Agent Social Autonome »

Figure 3. *Description d'un agent comportant deux configurations*

Dans ce scénario, nous voulons concevoir un agent qui a pour objectif de maximiser la somme des récompenses qu'il obtient en exécutant les tâches. Considérons l'agent qui est décrit par les *deux configurations* illustrées sur la figure 3. Chaque configuration représente un comportement possible de l'agent. En l'occurrence, ces deux configurations traduisent deux stratégies pour obtenir des récompenses. La configuration **Agent Social** désigne un agent qui interagit librement avec les autres agents et accepte toutes leurs demandes tant qu'il peut les assumer. Dans cette configuration, le rôle `Communication` permet à l'agent de communiquer avec les autres agents. Le rôle `Connexion à l'environnement` permet à l'agent d'agir sur son environnement et de percevoir les tâches et les autres agents. Le rôle `Génération d'une tâche` permet à l'agent de réifier une tâche à partir des messages ou des percepts

qu'il reçoit. Cette tâche est envoyée vers l'ordonnanceur. Le rôle `Ordonnanceur` gère les demandes d'ajout de tâches : la tâche est acceptée si et seulement si l'agent peut la réaliser avant son échéance sans remettre en cause les autres tâches. Par ailleurs la configuration **Agent Social Autonome** désigne un agent qui choisit les demandes à accepter selon son intérêt propre. Ce choix peut amener l'agent à ne pas accomplir une tâche qu'il s'était précédemment engagé à effectuer, au profit d'une nouvelle tâche qui améliore son bénéfice. Dans la configuration, cette différence de comportement se traduit par le rôle *Sélection Utilitaire* qui reçoit les demandes d'ajout de tâches et modifie l'ordonnanceur pour maximiser les bénéfices de l'agent.

Nous supposons qu'il y a un mécanisme qui calcule régulièrement le taux de satisfaction de l'agent en fonction des récompenses qu'il a obtenues jusqu'à présent. Par hypothèse, la valeur de ce taux de satisfaction fait partie de l'*état de l'agent* et va être utilisée pour déclencher les ré-assemblages. Ainsi, nous allons considérer cinq paliers de satisfaction : 0%, 20%, 40%, 60% et 80%. Dans cet exemple illustratif, nous définissons une *politique d'assemblage* assez simpliste qui se résume à changer alternativement de configuration lorsque le taux de satisfaction de l'agent passe au palier inférieur. Même si la configuration « Agent Social Autonome » est a priori plus rentable que la configuration « Agent Social » en terme de récompense, l'agent a intérêt à ne pas être constamment « Agent Social Autonome ». En effet, pour maximiser ses gains, un agent social autonome peut rompre ses engagements vis-à-vis des autres agents, en abandonnant une tâche coopérative avant qu'elle ne soit réalisée, au profit d'une tâche plus rentable. C'est la raison pour laquelle les autres agents peuvent devenir méfiants envers lui et ne plus lui proposer de tâches coopératives s'il les abandonne trop souvent. Par contre, un agent social ne rompt jamais ses engagements. Il gagne facilement la confiance des autre agents, mais ses gains sont aléatoires. C'est pourquoi il est utile de changer de configuration de temps en temps.

4.2. *Ré-assemblage de l'agent*

Ce scénario décrit l'adaptation automatique d'un agent conçu avec MADCAR, i.e un agent contenant un moteur d'assemblage. Plus particulièrement, nous allons détailler les étapes d'un ré-assemblage d'agent *avec changement de configuration*[5]. Nous supposons que la configuration courante de notre agent est « Agent Social Autonome ». De plus, l'agent contient un ensemble de composants qui sont assemblés conformément à cette configuration : *messenger*, *perceptor*, *task manager*, *scheduler* et *maximizer*.

5. Par opposition à un ré-assemblage *sans changement de configuration* comme lors d'un remplacement du composant de communication de l'agent, afin de se conformer à un nouveau protocole de communication.

4.2.1. *Déclenchement du ré-assemblage*

L'agent perçoit que son taux de satisfaction vient de passer en dessous du palier de 60%. Donc, le processus d'assemblage est déclenché automatiquement conformément à la politique d'assemblage.

4.2.2. *Identification des compatibilités*

Le moteur d'assemblage de l'agent compare deux-à-deux les composants présents dans l'agent avec les rôles des deux configurations fournies. Il est résulte la matrice de compatibilités du tableau 1. Cette matrice est simpliste pour cet exemple mais MADCAR considère qu'il pourrait exister plusieurs composants pouvant jouer un même rôle. De même, il est possible qu'un composant soit compatible avec plusieurs rôles.

	Roles / Components	messenger	perceptor	task manager	scheduler	maximizer
Config 1	Communication (1,1)	X				
	Connexion à l'environnement (1,1)		X			
	Génération de tâche (1,1)			X		
	Ordonnanceur (1,1)				X	
Config 2	Communication (1,1)	X				
	Connexion à l'environnement (1,1)		X			
	Génération de tâche (1,1)			X		
	Ordonnanceur (1,1)				X	
	Sélection utilitaire (1,1)					X

Tableau 1. *Exemple de matrice de compatibilités entre des rôles et des composants*

4.2.3. *Sélection d'une configuration*

D'après la matrice de compatibilités, chacun des rôles spécifiés peut être rempli par un composant différent. En d'autres termes, les deux configurations de l'agent sont éligibles. Le moteur d'assemblage de l'agent doit choisir l'une d'entre elles en fonction de la politique d'assemblage. Celle-ci stipule que l'agent doit forcément changer de configuration. C'est donc la configuration « Agent Social » qui est choisie.

4.2.4. *Sélection d'un ensemble de composants*

La politique d'assemblage de cet agent ne spécifie aucune contrainte sur la façon de choisir les composants. Donc, n'importe quel ensemble de composants capable de remplir la configuration choisie est valide. Les composants sélectionnés sont : *messenger, perceptor, task manager* et *scheduler.*

4.2.5. *Assemblage d'un ensemble de composants*

Les composants sélectionnés sont finalement assemblés selon la configuration « Agent Social ». Parmi les mécanismes qui sont mis en œuvre pendant cette étape de réalisation de l'assemblage, nous pouvons citer la sauvegarde et la restauration de l'état des composants à remplacer, ainsi que la connexion des composants entre eux selon ce qui est décrit dans la nouvelle configuration. Notamment, l'ensemble des

tâches pour lesquelles l'agent s'est engagé doivent être gardées en mémoire lorsque le composant qui remplit le rôle `Ordonnanceur` doit être remplacé. Dans cet exemple, la plupart des composants du précédent assemblage sont conservés sans être réinitialisés. Il reste juste à connecter directement le composant *task manager* au composant *scheduler* : le composant *maximizer* ne sera pas utilisé jusqu'au prochain ré-assemblage.

Dans ce scénario, nous n'avons pas expliqué l'ensemble des mécanismes permettant de garantir la cohérence de l'agent pendant la réalisation de l'assemblage, notamment lorsque les ré-assemblages doivent être procédés dynamiquement. Ces mécanismes doivent faire l'objet d'une étude plus approfondie.

5. Conclusion et perspectives

Dans cet article, nous avons proposé d'utiliser le modèle MADCAR pour réaliser des agents *ouverts*, *adaptables* et *autonomes*. MADCAR est un modèle de *moteurs d'assemblage automatique et dynamique de composants*. Notre proposition consiste à intégrer à chaque agent un moteur d'assemblage basé sur MADCAR, une politique d'assemblage ainsi que l'ensemble des composants à assembler et la description de l'agent. La description de l'agent correspond à un ensemble de configurations valides. Chaque configuration est un assemblage de rôles, où un rôle est une description abstraite de composants constituée d'un ensemble de *contrats*.

Une des principales caractéristiques de MADCAR est l'utilisation des mêmes mécanismes d'assemblage que ce soit pour construire un agent à base de composants ou pour l'adapter. L'adaptation d'un agent consiste à ré-assembler les composants de l'agent, en particulier après un changement de configuration. Notons que notre modèle peut prendre en charge des adaptations imprévues lors de la conception de l'agent puisque l'ensemble des configurations, l'ensemble de composants et la politique d'assemblage peuvent être changés dynamiquement. C'est pourquoi les agents conçus avec MADCAR sont ouverts. L'autonomie de l'agent est basée sur d'une part, un moteur d'assemblage général et d'autre part, une politique d'assemblage particulière. La politique d'assemblage permet de piloter le processus d'assemblage mis en œuvre par le moteur, depuis le déclenchement d'un assemblage jusqu'à sa réalisation. Par ailleurs, la principale tâche du concepteur d'agent est de spécifier des configurations et une politique d'assemblage pour son agent. Les concepts de rôle et de configuration que nous avons définis permettent de spécifier des assemblages en gardant un couplage faible avec les composants de l'agent. Cela présente un avantage pour la réutilisabilité des configurations. Par exemple, un agent pourrait récupérer une configuration depuis un autre agent et l'utiliser à son compte lorsqu'il contient un ensemble de composants qui peut remplir cette configuration.

En perspective, nous comptons fournir un formalisme de haut-niveau pour exprimer facilement des configurations et des politiques d'assemblages dans MADCAR, et aussi pour modéliser le contexte d'une application. Ce formalisme doit nous permettre de spécifier le comportement d'agents auto-adaptables. Actuellement, nous travaillons

sur une projection de MADCAR sur le modèle de composants Fractal (Bruneton *et al.*, 2002). Nous avons choisi de travailler avec ce modèle de composants hiérarchiques car nous envisageons d'étendre l'utilisation de MADCAR au ré-assemblage de composants composites. Enfin, nous devons étudier de manière spécifique la prise en compte de la dynamicité lors de l'étape de réalisation du processus d'assemblage défini dans MADCAR. En effet, les adaptations dynamiques d'applications posent des problèmes bien connus mais difficiles à résoudre de manière automatique (Segal *et al.*, 1993; Hicks, 2001).

6. Bibliographie

Beugnard A., Jezequel J.-M., Plouzeau N., Watkins D., « Making Components Contract Aware », *Computer*, vol. 32, n° 7, p. 38-45, 1999.

Bourne R., Excelente-Toledo C., Jennings N., « Run-time selection of coordination mechanisms in multi-agent systems », *14th European Conf. on Artificial Intelligence (ECAI-2000)*, Berlin , Germany, p. 348-352, 2000.

Brazier F. M. T., Jonker C. M., Treur J., « Principles of component-based design of intelligent agents », *Data Knowl. Eng.*, vol. 41, n° 1, p. 1-27, 2002.

Bruneton E., Coupaye T., Stefani J., « Recursive and dynamic software composition with sharing », *WCOP'02–Proceedings of the 7th ECOOP International Workshop on Component-Oriented Programming*, Malaga, Spain, Jun, 2002.

Fuxman A. D., A Survey of Architecture Description Languages, Technical Report n° CSRG-407, Department of Computer Science, University of Toronto, Canada, 2000.

Hicks M., Dynamic Software Updating, PhD thesis, Department of Computer and Information Science, University of Pennsylvania, August, 2001.

Medvidovic N., Taylor R. N., « A framework for classifying and comparing architecture description languages », *ESEC '97/FSE-5 : Proceedings of the 6th European conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of software engineering*, Springer-Verlag New York, Inc., New York, NY, USA, p. 60-76, 1997.

Meyer B., « Applying "Design by Contract" », *Computer*, vol. 25, n° 10, p. 40-51, 1992.

Occello M., Baeijs C., Demazeau Y., Koning J.-L., « MASK : An AEIO Toolbox to Develop Multi-Agent Systems », *Knowledge Engineering and Agent Technology, IOS Series on Frontiers in AI and Applications*, Amsterdam, The Netherlands, 2002.

Ricordel P.-M., Demazeau Y., « La plate-forme VOLCANO : modularité et réutilisabilité pour les systèmes multi-agents », *Numéro spécial sur les plates-formes de développement SMA. Revue Technique et Science Informatiques (TSI)*, 2002.

Segal M. E., Frieder O., « On-the-Fly Program Modification : Systems for Dynamic Updating », *IEEE Softw.*, vol. 10, n° 2, p. 53-65, 1993.

Vercouter L., « MAST : Un modèle de composants pour la conception de SMA », *Journées Multi-Agents et Composants, JMAC 2004*, Paris, France, 23-23 novembre, 2004.