

Table des matières

Chapitre 1. Architectures Logicielles et Systèmes Multi-Agents	11
Jean-Paul ARCANGELI, Victor NOËL, Frédéric MIGEON	
1.1. Introduction	11
1.2. Systèmes multi-agents et ingénierie logicielle orientée agent	12
1.2.1. Agent	12
1.2.2. Système et interactions	13
1.2.3. Système multi-agent	14
1.2.3.1. Organisation	15
1.2.3.2. Environnement	15
1.2.4. Exemples de systèmes multi-agents	16
1.2.5. Ingénierie logicielle orientée agent	17
1.2.5.1. Processus	18
1.2.5.2. Notations et métamodèles	19
1.2.5.3. Outils	20
1.2.5.4. Bilan	21
1.3. Les systèmes multi-agents en tant que style d'architecture	21
1.3.1. Positionnement du style SMA	22
1.3.2. Caractéristiques en matière d'abstraction	22
1.3.2.1. Haut niveau d'abstraction	22
1.3.2.2. Niveau d'abstraction modulable	24
1.3.2.3. Conséquences en terme de support de mise en œuvre	24
1.3.2.4. Réutilisation et patrons de conception	25
1.3.3. Caractéristiques en matière de (dé)composition	26
1.3.3.1. Décentralisation et autonomie	26
1.3.3.2. Composition et couplage	27
1.3.3.3. Décomposition et émergence	27
1.3.4. Lien avec les exigences	28
1.3.4.1. Principales exigences couvertes	28
1.3.4.2. Vues « module » et « allocation »	29

10 Short Title

1.3.4.3. Documentation	30
1.3.4.4. Exigences non couvertes	30
1.3.5. Une famille de styles architecturaux	31
1.4. Le fossé architectural	32
1.4.1. Etat de la pratique	32
1.4.2. Analyse d'un point de vue architectural	33
1.4.2.1. Vues « macro » et « micro »	33
1.4.2.2. Séparation des préoccupations et des métiers	34
1.4.3. Bilan	36
1.5. Comment combler le fossé architectural	36
1.5.1. Limites des solutions existantes	36
1.5.2. Réalisation de l'architecture micro	37
1.5.2.1. Défis	37
1.5.2.2. Une solution à base de composants logiciels	38
1.6. Conclusion	39
1.7. Bibliographie	39

Chapitre 1

Architectures Logicielles et Systèmes Multi-Agents

1.1. Introduction

Les systèmes multi-agents (SMA) sont des systèmes logiciels dont la fonction est réalisée par une ensemble d'entités autonomes et indépendantes en interaction.

Les premiers travaux dans le domaine datent de la fin des années 80. Les SMA prennent racine en « intelligence artificielle », en particulier en « intelligence artificielle distribuée », mais aussi dans les travaux sur les paradigmes et les langages de programmation concurrente et répartie. C'est au milieu des années 90 que la communauté a commencé à s'intéresser à la question du développement des SMA et à aborder des problèmes de « génie logiciel ». Mais rares sont les travaux qui ont effectivement abordé les SMA du point de vue des architectures logicielles.

Réaliser une application en prenant la technologie SMA comme base de la solution est un choix de conception architecturale qui répond à certaines exigences relatives à la complexité, la distribution, la dynamique, l'adaptation... L'objet de ce chapitre est de présenter les SMA en tant que style d'architecture logicielle, de positionner ce style par rapport à d'autres styles connus (objets, composants logiciels, services, acteurs), d'identifier ses singularités ainsi que les problèmes particuliers que le choix de ce style pose en terme de développement, et enfin d'exposer les principes d'une solution à ces problèmes.

Chapitre rédigé par Jean-Paul ARCANGELI, Victor NOËL et Frédéric MIGEON.

Ce chapitre est organisé comme suit. La section 1.2 présente le paradigme SMA et les principaux concepts du domaine, ainsi qu'un état de l'art sur l'ingénierie logicielle orientée agent (le lecteur familier des SMA pourra sauter cette section). Les systèmes multi-agents sont ensuite analysés et positionnés en tant que style d'architecture dans la section 1.3, et les avantages et les inconvénients du style SMA en tant que support de conception architecturale sont synthétisés. La section 1.4 traite de l'existence et de la nature d'un fossé architectural entre la conception et la mise en œuvre des SMA, à l'origine de difficultés de développement et d'une perte de qualité des logiciels produits. En réponse à ce problème, les grandes lignes d'une solution sont présentées dans la section 1.5 . Pour terminer, une conclusion est donnée dans la section 1.6 .

1.2. Systèmes multi-agents et ingénierie logicielle orientée agent

Le concept d'agent est présent dans différents domaines de l'informatique et le terme « agent » est surchargé de sens. Il n'existe pas de définition unique et acceptée de tous, y compris dans la communauté « systèmes multi-agents ». Malgré tout, il est communément admis que le terme « agent » fait référence à des entités qui se caractérisent par un certain niveau d'autonomie et une capacité à agir ou à réagir à des situations, des évènements, des messages, etc. Pour G. Weiss [WEI 99], un agent est un système informatique autonome, réactif, proactif et capable de communiquer. On peut donner une autre définition inspirée de celle de J. Ferber [FER 95] : un agent est une entité (physique ou virtuelle) capable de percevoir partiellement son environnement, d'agir dans celui-ci et de communiquer avec d'autres agents, éventuellement de se reproduire, et dont le comportement tend à satisfaire un ensemble d'objectifs en fonction de ses ressources et de ses compétences propres ainsi que de la représentation partielle qu'il possède de son environnement.

Il existe également de multiples définitions de ce qu'est un système multi-agent [FER 95, WEI 99, BRI 01, WOO 09]. Par exemple, pour G. Weiss, c'est un système composé de multiples agents en interaction.

Cette section a pour objet de présenter et de discuter plus en détail ces concepts, de montrer quelques domaines d'application, puis de faire un panorama des méthodes et des techniques de développement orienté agent.

1.2.1. Agent

D'un point de vue logiciel, l'agent est une brique de base pour la conception et la programmation d'applications concurrentes et/ou réparties dans lesquelles le contrôle est fortement décentralisé. Ainsi, la conception à base d'agents peut se substituer à la conception à base d'objets par exemple, pour répondre à des besoins de modélisation de systèmes réels ou de résolution distribuée de problèmes. En terme de modèle de

programmation, l'ancêtre des agents est le modèle d'acteur de C. Hewitt [HEW 77] repris par G. Agha dans le cadre des systèmes répartis [AGH 86]. La communauté « systèmes répartis » s'est par ailleurs emparée du concept d'agent pour développer celui d'agent mobile [FUG 98].

Chaque agent a une activité spécifique définie par son comportement individuel et réalisée dans le cadre d'un « cycle de vie » qui ordonne ses opérations de perception, de décision et d'action. Les agents sont autonomes en matière de contrôle et de « sélection de l'action » [BRI 09] : la prise en compte ou non d'une situation ou d'un message est au choix de l'agent et est décidée dynamiquement en son sein, ainsi que le choix des actions qui seront réalisées. Ainsi, les agents peuvent être proactifs (pas seulement réactifs), c'est-à-dire qu'ils peuvent agir de leur propre initiative.

C'est l'intelligence artificielle classique qui fournit aux agents les moyens nécessaires en matière de prise de décision. Les agents cognitifs ou délibératifs (ou encore intelligents) sont dotés de connaissances et de moyens de raisonnement logique. Le plus souvent, ils poursuivent un but explicite, et leur capacité de raisonnement leur permet de choisir les actions à effectuer afin d'atteindre ce but. Par exemple, les agents de type BDI (*Belief, Desire, Intentions*) [RAO 95] s'appuient sur des croyances réalisables sur l'état du monde qui les entoure et sur des buts (désirs) pour produire des intentions, ces dernières étant à l'origine du choix des actions que les agents réalisent. En matière de conception, la complexité réside parfois là, dans l'intelligence de l'agent (mécanismes avancés de décision, apprentissage...), et l'application peut n'être composée que d'un petit nombre d'agents « intelligents » (voire d'un seul). Néanmoins cependant que certains mécanismes, tels que la réflexion ou le changement de comportement des acteurs, donnent aux agents des moyens d'évolution autonome tout en étant dépourvus d'intelligence.

Par ailleurs, il est important de noter que l'intelligence n'est pas une propriété obligatoire dans les SMA : les agents « réflexes » agissent et réagissent sans aucune intelligence.

1.2.2. Système et interactions

Fondamentalement, les agents opèrent et interagissent au sein d'un système dans lequel le contrôle et les données sont décentralisés.

De manière abstraite, pour agir et interagir les agents sont dotés de capteurs et d'effecteurs (appelés aussi actionneurs). L'agent perçoit (partiellement) le système qui l'entoure via ses capteurs (à travers lesquels l'information entre dans l'agent) et agit sur celui-ci via ses effecteurs (à travers lesquels l'information sort de l'agent). Capteurs et effecteurs supportent les transferts d'information en mode *push* ou *pull* selon qui est à l'initiative du transfert. Ainsi, l'agent peut être vu comme une machine

abstraite dont l'architecture interne définit comment les informations sont traitées, produites, entrent et sortent. Il existe de multiples types d'agent (la communauté SMA emploie le terme « architecture d'agent ») selon la façon de percevoir l'information, de la traiter et d'agir sur le système environnant.

Les mécanismes d'interaction contribuent également à alimenter en connaissances les prises de décision de l'agent. Celles-ci sont donc « contextualisées » : l'agent agit en fonction de la représentation qu'il a de son environnement et de l'état qu'il en perçoit. Il peut aussi recevoir un *feedback* sur ses actions qui contribue à améliorer la qualité de ses décisions futures.

Il est néanmoins fréquent que ces capacités d'interaction se réduisent à un simple mode de communication par message, point à point, unidirectionnel et asynchrone (comme dans le cas des acteurs). Ces interactions par messages peuvent cependant s'inscrire dans le cadre d'échanges complexes régis par des protocoles de communication. Pour cela, il existe des langages de communication entre agents ou ACL (*Agent Communication Language*) qui définissent des types de messages et des protocoles pour organiser les conversations entre les agents. Le plus connu de ces langages, FIPA-ACL [FIP02], a été défini et normalisé par la fondation FIPA (*Foundation for Intelligent Physical Agents*)¹.

Nous avons vu que, conceptuellement, un agent est autonome en matière d'exécution et peut déclencher lui-même ses actions. Dans un système, les agents s'exécutent donc naturellement en mode asynchrone. En pratique, la réalité de l'autonomie opérationnelle dépend de l'implémentation et de la politique d'allocation des ressources du système d'exécution (par exemple, *multi-threading*). On peut noter que la distribution du contrôle et l'asynchronisme peuvent conduire assez naturellement à des systèmes non déterministes, mais cela n'est pas obligatoire ni forcément désirable.

1.2.3. Système multi-agent

L'une des principales raisons d'être des SMA est de distribuer l'intelligence (on est ici au cœur de l'intelligence artificielle distribuée), la connaissance et le contrôle. Dans un SMA, aucun agent ne contrôle le système à lui seul et ne possède la totalité de l'information ; c'est le collectif qui prévaut sur l'individu et qui réalise la tâche pour laquelle le système a été conçu. Outre cette « intelligence collective », les SMA présentent deux caractéristiques majeures :

- L'existence d'une organisation qui régule le fonctionnement du système, et au sein de laquelle les agents sont des entités « sociales »,

1. <http://www.fipa.org>

- La présence dans le système d'un élément particulier, l'environnement, constitué d'entités qui ne sont pas des agents.

Ces deux caractéristiques se déclinent différemment dans les multiples approches qui ont été définies par la communauté SMA. Une approche définit un ensemble cohérent de concepts qui permettent d'aborder un problème et qui supportent la conception de la solution sous forme de SMA. Une approche peut reposer sur une théorie et son utilisation peut être encadrée par une méthode (modèles, processus...) comme nous le verrons dans la section 1.2.5.

1.2.3.1. *Organisation*

La façon dont les agents sont organisés au sein du système est de première importance en terme de conception et de fonctionnement. L'organisation définit un cadre pour l'interaction, la collaboration et le partage des tâches entre les agents, possiblement en utilisant des « lois sociales » qui ne sont pas sous le contrôle des agents. Elle peut prendre différentes formes selon les approches, mais toute organisation est basée sur la notion de rôle, explicite ou non, qui définit de manière abstraite le comportement (la fonction) des agents. Par exemple, le modèle AGR (Agent-Groupe-Rôle) est un modèle organisationnel à base de rôle pour les SMA dynamiques et ouverts [FER 04]. De manière générale, un agent peut jouer plusieurs rôles et un rôle peut être joué par plusieurs agents, et ces associations peuvent être réalisées dynamiquement [ODE 03].

Certaines approches définissent, pour une application donnée, une organisation qui ne change pas durant l'exécution du système, alors que d'autres s'appuient sur la capacité d'auto-organisation des agents qui sont dotés d'un comportement « social » leur permettant de changer de rôle à l'exécution. L'organisation est donc la façon de composer les différents éléments du système, et la fonctionnalité du système résulte de l'organisation des agents. C'est alors en changeant cette organisation que le système peut adapter sa fonctionnalité [DEM 95]. Ainsi, certaines approches cherchent à exploiter l'auto-organisation pour faire émerger des organisations adéquates, de sorte que le système réalise la fonctionnalité attendue [DIM 11].

1.2.3.2. *Environnement*

Dans un SMA, l'environnement est un espace partagé qui supporte l'échange d'informations : c'est le medium pour l'interaction et la coordination entre agents, à travers lequel ces derniers agissent et interagissent. Il possède sa propre dynamique et ses processus, indépendamment de la dynamique des agents. On peut distinguer l'environnement logique qui est un composant à part entière du SMA et l'environnement système qui est externe au SMA (défini par le contexte de déploiement du SMA et les utilisateurs). Pour Weyns *et al.*, l'environnement est une abstraction de première classe qui fournit aux agents les moyens de leur existence et qui sert à la fois de médiateur pour l'interaction entre les agents et l'accès aux ressources [WEY 07]. Un exemple

typique est celui des SMA « situés », pour lesquels les agents ont une position dans l'environnement : ils sont liés à une partie de celui-ci, accessible localement, qu'ils perçoivent et sur laquelle ils agissent.

L'environnement peut supporter des interactions complexes, comme par exemple la communication indirecte bio-inspirée de type « stigmergie » [GRA 59] : dans ce mode de communication, l'environnement contient des marqueurs (inspirés des phéromones des colonies de fourmis) qui peuvent être captés ou déposés par les agents.

D'une façon générale, on trouve dans l'environnement des entités, passives ou actives, qui n'ont pas d'autonomie. En particulier, certaines approches considèrent et conçoivent l'organisation comme une entité logicielle qui fait partie de l'environnement et qui est responsable de réguler les interactions entre agents.

1.2.4. Exemples de systèmes multi-agents

Il existe de nombreuses applications des SMA dans le but de résoudre des problèmes de manière distribuée (optimisation, aide à la décision...), de simuler des phénomènes complexes (sociaux ou naturels) ou de gérer des ressources et des systèmes complexes. Ces applications concernent de multiples domaines : robotique, transport, sécurité d'installations, gestion de crise, jeux vidéo, systèmes et processus industriels, systèmes embarqués, intelligence ambiante...

Parmi les exemples, on peut citer le contrôle de chaînes de fabrication (*manufacturing and control*) [SHE 06] où le problème est d'affecter des produits à fabriquer à des machines, celles-ci étant contrôlées par des opérateurs tout en respectant certaines contraintes tel qu'un ordre dans les étapes de fabrication. Cet ordre n'étant pas prévisible, les machines ne pouvant traiter qu'un produit à la fois et les opérateurs qui pilotent les machines n'étant pas disponibles en permanence pour toutes les machines, la fonction du SMA est d'optimiser ces affectations produit-machine et machine-opérateur de façon à minimiser les temps de production et à respecter des délais stricts. Une modélisation classique du problème consiste à représenter les machines, les produits et les opérateurs par des agents et à leur donner un comportement qui leur permet de s'organiser pour répondre aux demandes de fabrication. L'arrivée de nouvelles demandes de fabrication et les indisponibilités imprévues d'opérateurs ou de machines se traduisent par la création et la suppression d'agents dans le système et, par conséquent, l'adaptation et la réorganisation dynamique du système afin de satisfaire les demandes.

Un autre domaine privilégié d'application des systèmes multi-agents est la simulation dite « individu-centrée » [AMB 03] dans laquelle, pour modéliser un phénomène collectif, on se focalise sur les comportements et les interactions au niveau des individus. Chaque individu est un agent plongé dans un environnement. Le but est d'observer et d'expliquer le phénomène collectif qui se produit au niveau du système ainsi que

d'identifier les comportements individuels qui sont à l'origine du phénomène collectif et leur impact sur ce dernier.

La plupart des applications de type SMA ont été développées dans un contexte académique. Parmi les cas d'utilisation réelle des SMA dans un contexte industriel, on peut citer celui de la société Massive Software² qui utilise le paradigme des systèmes multi-agents dans le domaine de la réalité virtuelle pour offrir à ses clients les moyens de réaliser des scènes d'action dans des films : des foules de personnages sont définies à partir d'agents (un agent pour un personnage) dont chacun suit un comportement prédéfini. Un éditeur permet de définir les déplacements des agents dans l'environnement, de développer interactivement les décisions que peut prendre un agent, ceci sans aucune connaissance de programmation. Ainsi, il est possible de réaliser des animations ou des simulations dans lesquelles des milliers d'agents interagissent pour donner une scène réaliste³.

1.2.5. Ingénierie logicielle orientée agent

L'ingénierie logicielle orientée agent (AOSE, *Agent-Oriented Software Engineering*), est un terme générique qualifiant tout développement de systèmes à agents mais pas forcément de système multi-agent. Présenter les aspects méthodologiques nécessite d'aborder le processus logiciel, la notation et le méta-modèle utilisés, ainsi que les outils disponibles dont une partie importante concerne les plateformes de développement. Ici nous aborderons uniquement ce qui concerne les systèmes multi-agents.

Comme dans le génie logiciel classique, la méthode vient supporter (au sens anglais) l'approche adoptée pour le développement. Ainsi, pour une approche SMA donnée (tel que cela est exposé dans les parties 1.2.1 à 1.2.3), on peut trouver plusieurs méthodes qui en suivent les principes de manière plus ou moins fidèle. En théorie, étant données les exigences exprimées par les parties prenantes, l'équipe de développement doit déterminer quelle approche convient le mieux et la méthode pour la mettre en œuvre. En pratique, les affinités qu'une équipe de développement peut avoir avec une méthode contraignent autant le développement qu'elles ne l'accompagnent.

Les premières méthodes ont vu le jour au milieu des années 1990 mais c'est depuis les années 2000 que les recherches dans ce domaine ont été les plus intenses. Les méthodes les plus nombreuses, issues de l'ingénierie du logiciel, s'inspirent des méthodes orientées objet (cf. figure 1.1).

2. <http://www.massivesoftware.com>

3. <https://www.youtube.com/watch?v=ZwkjW4bmpYE>

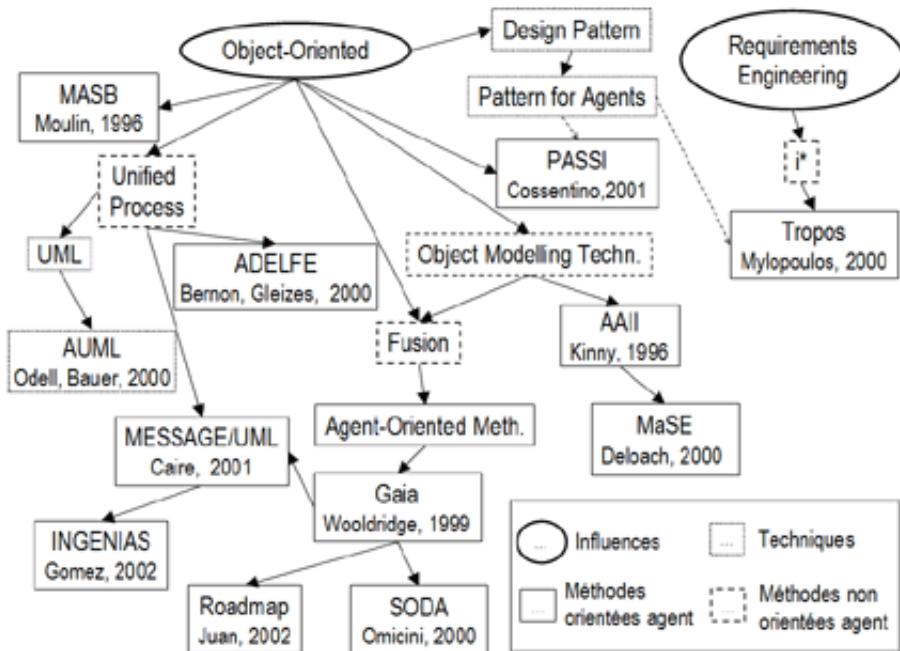


Figure 1.1. Influence des méthodes orientées objet (extrait de [GLE 08])

D'autres, que nous ne présentons pas ici, sont issues de l'ingénierie des connaissances ou bien ont été définies spécifiquement pour soutenir les développements promus par certaines approches.

1.2.5.1. Processus

Dans cette partie, nous faisons un petit tour d'horizon de quelques processus de développement parmi les plus remarquables sans prétendre à l'exhaustivité. Pour plus de références, nous conseillons l'ouvrage de Bergenti, Gleizes et Zambonelli ([BER 04]) comme point d'entrée sur les méthodes abordées dans la suite de cette partie.

Nous définissons un processus comme une succession de phases se décomposant en activités coordonnées (une activité pouvant elle-même se décomposer en étapes) qui peuvent produire et/ou consommer des artefacts logiciels sous la responsabilité de personnes assumant un rôle. La plupart des méthodes de développement de systèmes multi-agents s'inspirant de l'ingénierie orientée objet, il est naturel que les cycles de vie et processus utilisés dans la conception de tels systèmes soient des types classiquement utilisés en conception modulaire : en cascade, itératif, en V, en spirale.

De plus, ces méthodes intègrent, pour la grande majorité, les grandes phases de développement que sont l'extraction et l'analyse des besoins, la conception de l'architecture et l'implémentation, même si elles ne sont pas clairement identifiées en tant que telles. Par exemple, ADELFE et INGENIAS reposent sur le processus unifié (Unified Process, UP) et intègrent à ce processus de nouvelles étapes et définitions de travaux spécifiques au domaine multi-agent, alors que PASSI et ASPECS reposent sur un processus itérable original. Certaines méthodes ne couvrent qu'une partie du processus, comme Gaia ou SODA qui se concentrent exclusivement sur l'analyse des besoins et la conception, ou Prometheus et MaSE qui n'abordent pas l'analyse des besoins. Généralement, les méthodes ne proposent pas de processus allant jusqu'au déploiement, s'appuyant sur le principe que les modèles multi-agents sont davantage du niveau de l'analyse (notamment pour Tropos, issue de l'ingénierie des exigences) et de la conception, à l'exception de PASSI ou INGENIAS (qui couvrent tout le cycle de développement).

Outre les phases couvertes par les processus, il est également intéressant d'identifier ce que les processus proposent en terme de livrables, de gestion de la qualité et de directives de conduite. Les méthodes reposant sur le processus unifié, comme ADELFE ou INGENIAS, héritent de sa gestion des risques et de la qualité. Certaines méthodes prennent un soin particulier à la définition des processus et des artefacts logiciels, en utilisant notamment des formalismes dédiés comme SPeM⁴ (*Software Process Engineering Metamodel*). Enfin, il faut noter une forte volonté de la communauté SMA de documenter clairement l'ensemble des processus définis sous une forme standardisée comme c'est le cas avec les travaux de la FIPA autour du groupe thématique DPDF (*Design Process Documentation and Documentation*) qui a défini le standard SC00097B⁵.

1.2.5.2. Notations et métamodèles

Comme pour les processus et les outils, le large spectre des notations utilisées dans les méthodes de développement montre la grande diversité des approches. On y retrouve une vision unifiée de ces approches, issue principalement du monde objet, qui a donné lieu à l'utilisation de langages de modélisation qualifiés de génériques (GPML, *Generic Purpose Modelling Language*) comme UML2 ou SysML. Ils sont souvent utilisés dans les phases traditionnelles des processus, voire, sous la forme de profils UML le plus souvent, dans des activités plus spécifiques. Cependant, pour des activités de développement plus étroitement liées à une approche, l'utilisation de langages de modélisation spécifiques (DSML, *Domain Specific Modelling Language*), dans laquelle les concepts du domaine agent sont réifiés, devient plus courante. On peut alors trouver dans cette branche issue de l'Ingénierie Dirigée par les Modèles

4. <http://www.omg.org/spec/SPeM/>

5. <http://fipa.org/specs/fipa00097/SC00097B.pdf>

Tableau 1.1. Caractérisation des modèles de conception en termes d'abstractions architecturales

		SeSAmUML	PASSI2	ASPECs	AgentUML	A _{ML}	ADELFE	A&A	Macodo	AGR	MASQ	VOWELS	MOISE _z	Opera	MESSAGE	GAIA	CRIo	Tropos
Modèle de conception	Goal	X	X	X														X
	UML-like				X	X	X											
	BDI	X																
	Rules							X										
	Aptitudes		X					X										
Environnement	Recursion	X						X										
	Org. Struct.		X						X	X			X	X				X
	Org. Dyn.		X						X									
	Situated Entities	X				X	X			X	X							
		X																
Interaction	Messages		X	X	X	X	X					X			X	X	X	X
	Sens./Act.	X		X		X	X					X	X	X	X	X	X	X
	Role Play							X	X									

([FAV 06]) de nombreux langages de modélisation (AUML, INGENIAS, AMAS-ML pour ADELFE...).

À partir de l'étude de ces DSML, dont on peut avoir une représentation par ce qui est appelé un meta-modèle, il est possible d'analyser quels sont les concepts clefs d'une notation, et donc d'une méthode AOSE et de l'approche qu'elle promeut. On mesure alors la diversité des meta-modèles et la faible couverture que chacun offre de l'ensemble des concepts. Dans le tableau 1.1, tiré de [NOë 12], nous présentons une analyse comparative de caractéristiques de modélisation concernant l'architecture interne de l'agent, celle de l'environnement et de l'organisation du SMA, et celle des interactions. La dispersion des croix illustre à elle seule la diversité des approches.

1.2.5.3. Outils

La nature même du paradigme SMA et la richesse de ses approches rend difficile l'implantation des systèmes multi-agents directement à partir des langages de programmation standards. Le développement est alors facilité par l'utilisation d'outils et de plateformes (qui, entre autres, aident à la réutilisation). Comme pour l'utilisation des notations, celle des plateformes et outils de développement passe par une identification des concepts clés caractéristiques des approches promues. Dans la production de la communauté SMA, on trouve de très nombreuses plateformes spécifiques à une approche. Des tentatives d'unification existent mais se heurtent à la richesse du paradigme SMA : en prendre le dénominateur commun réduit le résultat à une description

trop abstraite du paradigme, et à l'inverse, l'agrégation des multiples concepts pose des problèmes de cohérence et de compatibilité.

1.2.5.4. *Bilan*

Comme nous l'avons montré dans cette section, le paradigme SMA révèle une grande richesse dont témoigne la diversité des approches définies et des plateformes qui les supportent. Pour autant, le développement de systèmes multi-agents recèle de singularités, d'usages et de difficultés qu'il nous faut encore détailler en termes architecturaux (et d'exigences associées). Culturellement, la communauté SMA ne s'est que peu interrogée sur les questions d'architecture logicielle et, qui plus est, que très récemment. On peut citer cependant les travaux de D. Weyns sur la conception et la documentation de SMA situés et de leur environnement [WEY 10]. Nous verrons dans la section suivante que le choix d'une approche particulière, les conséquences sur l'architecture à concevoir, l'effort à produire pour combler la distance entre le domaine du problème et celui de la solution sont autant de contraintes qui s'imposent au développeur d'un système multi-agent.

1.3. Les systèmes multi-agents en tant que style d'architecture

Dans la section précédente, nous avons présenté les concepts d'agent et de système multi-agent, d'interaction, d'organisation, d'environnement... Nous avons explicité la dynamique au sein des systèmes en termes sociaux et également au sein des agents en terme de cycle perception, décision et action. Cette section a pour objectif de donner une lecture et une compréhension de ces éléments d'un point de vue « architecture logicielle » et de montrer en quoi le « style SMA » se rapproche et se distingue d'autres styles architecturaux tels que les styles à objets, à composants logiciels, à base de services, à acteurs, etc. Nous comparons ici leurs propriétés structurelles, la façon d'aborder les problèmes et leurs solutions, et les exigences auxquelles les choix architecturaux répondent.

Après un positionnement du style SMA en termes architecturaux, nous l'analysons du point de vue des abstractions qui le caractérisent, puis du point de vue de la composition et des aspects structurels. Nous proposons ensuite une synthèse de ses avantages et de ses inconvénients relativement aux exigences sur le produit et sa documentation ainsi que sur le développement.

Ici, nous montrons aussi que, selon l'approche SMA choisie, le paradigme et le style sont déclinés de différentes façons. Nous en déduisons que, plus qu'un style unique d'architecture, il existe une famille de styles architecturaux de type SMA.

1.3.1. Positionnement du style SMA

Tout d'abord, il faut constater que les solutions exprimées au moyen du paradigme SMA entrent de manière quasi-exclusive dans le cadre des vues architecturales de la catégorie « composants et connecteurs » [CLE 03] : au niveau du système comme au niveau des agents, c'est la structure organisationnelle et la dynamique du système à l'exécution qui sont définis. Cependant, bien que peu explorés, les apports du style SMA aux vues des catégories « module » et « allocation » ne sont pas inexistant. Nous reviendrons sur ce point dans la section 1.3.4.2.

Si l'on suit la nomenclature de description de style utilisée dans [CLE 03], on constate que les éléments du style SMA sont variés (en fonction des approches et du problème à traiter), mais l'on retrouve généralement les grandes catégories suivantes :

- les agents ;
- les entités de l'environnement ;
- les connecteurs d'interaction (perception et action).

Il existe autant de types de connecteurs d'interaction que de mécanismes d'interaction tels que cela est décrit dans la section 1.2.5, parmi lesquels on distingue ceux pour l'interaction directe (d'agent à agent, tel que l'envoi de message asynchrone) et ceux pour l'interaction indirecte (d'agent à environnement, telle que la stigmergie, la perception située, les déplacements en environnement situé...). Nous reviendrons plus tard sur ces éléments et leur exploitation pour documenter l'architecture d'un SMA.

1.3.2. Caractéristiques en matière d'abstraction

Nous analysons ici le style SMA du point de vue des abstractions et des éléments qui le caractérisent, puis les conséquences en termes de support de mise en œuvre et les possibilités de réutilisation.

1.3.2.1. Haut niveau d'abstraction

Les paradigmes traditionnels sont fortement influencés par la machine et le support utilisé pour la mise en œuvre : des premiers langages impératifs jusqu'aux composants logiciels et aux services, les abstractions proposées ont été construites conformément aux couches plus basses déjà disponibles. A l'inverse, de par leurs racines en intelligence artificielle et comme le style de programmation « logique », les SMA se distinguent par une vision décorrélée de la machine (comme l'illustre la section 1.2.5) et un haut niveau d'abstraction dans la modélisation : ainsi, le système et ses éléments présentent des caractéristiques qu'on peut qualifier d'anthropomorphes (autonomie, interaction, socialité). Méthodologiquement parlant, la conception de tels systèmes s'intéresse donc tout particulièrement aux caractéristiques sociales, que ce soit en terme de communication, d'organisation, de rôles, etc. Cela se traduit par une

séparation claire et explicite dans le système entre les agents et l'environnement, et dans la conception des agents, entre perception et action d'une part (le volet social) et décision d'autre part (le volet autonomie).

Toute activité de conception a pour objet de traduire les concepts manipulés dans le domaine du problème en des concepts qui appartiennent au domaine de l'approche choisie pour construire la solution. Dans le cas du style SMA, la distance entre les deux domaines est faible et la conception revient le plus souvent à structurer la solution sous une forme proche du problème lui-même. Cette proximité est plus prononcée que dans le cas des objets, en particulier du fait de l'intégration de l'activité au sein des agents (qui concourt à l'autonomie et à l'anthropomorphisme) et du fait de l'exploitation de mécanismes d'interaction divers et variés au sein d'un environnement adapté aux agents. En minimisant le fossé conceptuel entre les deux domaines, la conception en est ainsi facilitée : les éléments modélisés, les agents, représentent les entités du problème et leur comportement, en particulier social, représente la façon dont la fonctionnalité du système est réalisée. Le concepteur doit cependant choisir les abstractions qui conviennent pour modéliser sa solution, et ce en fonction de son problème (par exemple les mécanismes d'interaction adaptés à la structure du problème).

Par exemple, dans le domaine des chaînes de fabrication, il est classique dans les SMA de modéliser les machines, les opérateurs et les produits par des agents et le circuit de circulation des produits par l'environnement (cf. section 1.2.4). Les affectations étant très dynamiques et changeantes, une gestion des relations sociales interne aux agents et des interactions par messages sont préférées. Ainsi, les éléments du problème (les machines, les opérateurs et les produits) font partie intégrante du processus de résolution. Inversement, dans une approche plus traditionnelle, on exprime la solution au moyen de concepts éloignés du problème mais proches d'un algorithme de résolution générique. Ce dernier n'est pas spécifique au problème : par exemple, en optimisation à base de contraintes, à partir de l'expression de contraintes sur des variables discrètes (valeurs appartenant à un ensemble fini), des algorithmes trouvent une allocation de valeurs pour ces variables. Cela demande une reformulation du problème. Par exemple, pour les chaînes de fabrication, les variables représentent des allocations produit-machine et machine-opérateur ainsi que des relations d'ordres entre ces allocations dans le temps. Les contraintes, quant à elles, expriment quelles relations temporelles sont possibles et impossibles, voire préférées, entre les valeurs de ces variables.

La contrepartie de ce haut niveau d'abstraction et de la proximité entre les concepts des domaines du problème et de la solution est l'existence d'un fossé entre la solution de conception et sa mise en œuvre. Nous reviendrons sur cette question dans la section 1.4.

1.3.2.2. *Niveau d'abstraction modulable*

La conception d'un SMA s'appuie donc sur les abstractions de haut niveau que sont les agents, l'environnement et les mécanismes d'interaction. Mais ceux-ci peuvent néanmoins prendre des formes très diverses. Certaines abstractions doivent donc faire l'objet d'un raffinement conformément à la fois à l'approche choisie et aux exigences auxquelles on veut répondre.

Ainsi, pour répondre à des exigences en matière de décentralisation, on se concentrera sur les capacités d'autonomie et de perception des agents ; pour répondre à des exigences en matière d'interopérabilité, on se concentrera sur les mécanismes d'interaction et de communication ; pour répondre à des exigences en matière d'adaptation, on se concentrera sur la dynamique de l'organisation, etc. Par exemple, dans le cadre d'une chaîne de fabrication où l'exigence première est l'allocation à la volée des tâches à des machines et à des opérateurs, on se focalisera sur la dynamique de l'organisation et surtout de la réorganisation (l'organisation réalisant ici les choix d'allocation faits par les agents à un moment donné, celle-ci évoluant à chaque évènement comme l'apparition d'une nouvelle tâche ou l'indisponibilité d'une machine ou d'un opérateur).

Cette propriété différencie le style SMA d'autres styles plus traditionnels car elle en fait, en quelque sorte, un style « ouvert » avec lequel le concepteur a la liberté (mais aussi la contrainte) de choisir quelles abstractions il veut utiliser et raffiner, et quelles abstractions il considère comme étant non architecturales dans la conception d'un SMA.

Dans certaines applications, on ne peut pas, par exemple, faire abstraction de l'ordonnancement des agents (typiquement, en simulation multi-agent) : quand il définit son système, le concepteur fait certaines hypothèses en matière d'ordonnancement (sans forcément les expliciter) ; il faut donc « descendre » au niveau de l'ordonnancement pour concevoir convenablement la solution. Mais dans de nombreuses applications, le problème de l'ordonnancement ne se pose pas. La sûreté du passage de message est un autre exemple de préoccupation qui peut être prise en compte ou non selon la nature de l'application : selon les cas, le concepteur fait ou pas l'hypothèse que les messages envoyés sont reçus de manière sûre.

En fonction du problème, le concepteur SMA choisira donc de prendre en compte ou pas certaines préoccupations, et pour cela il choisira les abstractions à raffiner.

1.3.2.3. *Conséquences en terme de support de mise en œuvre*

De manière générale, le choix d'un style architectural et d'un support de mise en œuvre associé (par exemple, un *middleware* ou un *framework*) répond à des exigences majeures et garantit certaines propriétés essentielles du produit logiciel fabriqué. En contrepartie, il constraint la solution et sa réalisation. Par exemple, le choix d'une architecture à objets constraint à exprimer toute interaction sous la forme d'un appel de méthode.

Dans le cadre du style SMA, le problème se pose de manière inverse : parce que les abstractions du style sont de haut niveau et que celles-ci subissent un raffinement lors de la conception, la contrainte réside dans le fait que les abstractions employées (typiquement, les divers mécanismes d'interaction) doivent être mises à disposition par un support de mise en œuvre adapté à l'approche et au domaine d'application. Par exemple, pour une interaction par envoi de messages, on peut utiliser directement le moyen offert par le support de mise en œuvre (c'est le cas en général), mais pour une interaction à base de stigmergie, il faut disposer d'un composant d'interaction spécifique fournissant cette abstraction.

C'est pour répondre à ce besoin que la production de plateformes orientées agent a pris une telle place dans la communauté SMA comme présenté section 1.2.5.3.

Ainsi, la conception d'un SMA et les choix faits lors de celle-ci contraignent le support de mise en œuvre. D'une manière générale, la diversité des types d'agent, des types d'environnement et des mécanismes d'interaction envisageables permettent (et imposent !) à l'architecte de définir et de séparer :

- les aspects SMA qui concernent la conception du système selon le paradigme SMA pour répondre au problème,
- et les aspects « opératoires », qui concernent les abstractions exploitées par la conception du système, c'est-à-dire comment fonctionnent les agents et les mécanismes sur lesquels ils s'appuient pour agir et interagir.

Un effort supplémentaire est donc nécessaire, plus ou moins important selon le support de mise en œuvre utilisé, et on ne peut pas espérer disposer systématiquement du support adéquat. Nous retrouvons ici l'idée du fossé entre conception et mise en œuvre évoqué précédemment et qui sera traité dans la section 1.4.

1.3.2.4. Réutilisation et patrons de conception

Les efforts de la communauté SMA pour fournir des éléments réutilisables se sont majoritairement cantonnés à la production de supports de mise en œuvre.

Des patrons ont cependant été proposés [OLU 07]. Traditionnellement, c'est en réponse à des problèmes de conception récurrents que des collections de patrons de conception et d'architecture sont élaborées : les patrons de conception permettent d'enrichir les abstractions « natives » d'un paradigme pour exprimer de nouvelles abstractions tandis que les patrons d'architecture s'appuient sur des abstractions disponibles pour modéliser des solutions organisationnelles réutilisables. Les abstractions proposées dans le cadre du style SMA étant, par nature, de haut niveau et raffinables, la majorité des patrons proposés par la communauté SMA sont d'ordre architectural. On retrouve typiquement des patrons de communication indirecte à travers différentes topologies d'environnement, des patrons d'organisation avec des rôles génériques et

des comportements à instancier selon les problèmes, des patrons s'appliquant à des classes de problèmes, etc.

En revanche, ces patrons ne répondent pas ou peu à des exigences relatives à l'organisation du développement, à la flexibilité et à l'extensibilité du logiciel produit contrairement aux patrons de conception du monde objet.

1.3.3. Caractéristiques en matière de (dé)composition

Dans cette partie, nous discutons des caractéristiques du style SMA en termes de décomposition en entités autonomes, de composition et de couplage.

1.3.3.1. Décentralisation et autonomie

Une caractéristique essentielle du style SMA est de combiner la décentralisation du contrôle et l'autonomie des agents. Au sein d'un SMA, les agents jouent des rôles et ces rôles structurent le SMA. Ainsi, l'agent est un composant à caractère social mais aussi autonome. Ceci se traduit par les propriétés suivantes :

- l'agent peut gérer lui-même ses relations avec d'autres agents ;
- l'agent a la capacité de déterminer lui-même ses actions (c'est-à-dire de faire ou de ne pas faire ce que d'autres agents attendent de lui).

Le premier point rapproche les agents des objets qui contiennent et gèrent en interne les références des objets dont ils requièrent les services, et les éloigne des composants logiciels (ou des services) pour lesquels, au contraire, la composition est externalisée. Le couplage est donc du même niveau que dans le cadre des objets mais l'autonomie a par ailleurs ses avantages que nous discutons ci-dessous. Cette gestion des relations sociales peut néanmoins être externalisée dans l'environnement comme par exemple dans les SMA situés, mais cela reste tout de même un choix de conception non imposé par le style comme dans le cas des composants.

Le second point repose sur le découplage entre la prise d'information (la réception d'un message par exemple) et le traitement de cette information (le traitement du message). Il rapproche les agents des acteurs (du modèle d'acteur de C. Hewitt) et les éloigne nettement des paradigmes et styles plus traditionnels où les fournisseurs de services n'ont aucun choix en matière de réponse à une demande. Finalement, avec le style SMA, on est bien loin de la structuration des applications des programmes en sous-programmes qui sous-tendent la plupart des approches traditionnelles. Cette véritable « inversion de contrôle » n'est pas seulement structurelle, comme elle l'est pour le patron de conception de même nom, mais réellement sémantique.

1.3.3.2. *Composition et couplage*

En matière de composition, les moyens d’interaction spécifient la nature des échanges entre agents mais ne servent pas à spécifier les services rendus par l’agent, à l’inverse des interfaces traditionnelles (plus ou moins riches) qui sont plus explicites. En effet, alors que dans des approches traditionnelles, les interactions entre éléments du système représentent des demandes en terme de fonctionnalité, dans les SMA, les agents échangent de l’information, que ce soit de façon directe ou indirecte, qui permet à l’agent de décider d’effectuer ou non des actions, c’est-à-dire rendre son service. Ce point est significatif d’un changement de paradigme et la composition des agents (pour former un système) se traduit par un « couplage sémantique » guidé par les connaissances et l’organisation sociale du travail [BRI 09]. D’un point de vue architectural, il se manifeste particulièrement par la définition et l’utilisation de connecteurs (d’interaction) plus pauvres sémantiquement que dans une approche traditionnelle à composants, mais plus diversifiés. Cette diversité résulte à la fois de la variété des types de d’informations qui peuvent être échangées, mais aussi et surtout, de la nature du problème et de la structure de ce dernier : par exemple si les éléments du problème sont situés, alors il est pertinent que les agents perçoivent et agissent (et donc échangent de l’information par ce biais) dans un environnement situé.

1.3.3.3. *Décomposition et émergence*

Cette manière de coupler sémantiquement les éléments d’un système s’accompagne d’un changement dans la façon d’aborder la conception : d’une décomposition fonctionnelle en sous-programmes, on passe à une vision plus intégrée au problème lui-même où chaque agent a une fonctionnalité locale. La fonctionnalité du système (comportement global) « émerge » alors des interactions entre agents.

Ceci est bien illustré par l’application de contrôle de chaînes de fabrication (cf. section 1.2.4). Une approche traditionnelle consiste à considérer la question de manière globale et à découper la recherche de la solution en plusieurs tâches distinctes. Par exemple, la résolution par contraintes s’appuie sur un algorithme générique qui cherche séquentiellement des valeurs valides pour les variables, vérifie les contraintes et optimise certains critères, et ce de façon itérative. Mais quand le problème change, le processus de résolution doit être repris entièrement. En revanche, dans une approche SMA, on découpe le problème en participants distincts qui cherchent collectivement la meilleure solution. C’est par leurs interactions locales, à travers un comportement social propre à chaque élément du problème – ici former des allocations deux par deux selon des critères locaux –, que le comportement global de résolution émerge et s’adapte continuellement aux changements du problème. Plus de détails sur cet exemple et sur la différence avec les approches traditionnelles peuvent se trouver dans [KAD 11].

D'un point de vue architectural, cette manière de décomposer le système est donc d'autant plus intéressante que la structure du système, en termes d'agents et d'interactions, n'est pas fixe mais évolue durant l'exécution du système : les agents jouent le rôle de garants de la validité de cette structure vis-à-vis des besoins fonctionnels. On peut considérer, en particulier dans le cadre des SMA auto-organisateurs, le système comme une architecture sans cesse en réorganisation et s'adaptant donc à l'évolution des exigences auxquelles elle répond. La conception de tels SMA basés sur l'émergence n'est bien entendu pas simple, et ce problème est sujet à de nombreux travaux de recherche [DIM 11], entre autres pour s'assurer de la qualité du logiciel réalisé.

1.3.4. *Lien avec les exigences*

La programmation par aspects et le concept de composant logiciel ont été introduits pour pallier certaines limites du style objet. Le concept d'agent y contribue aussi : alors que le concept de composant logiciel peut être vu comme une évolution du concept d'objet afin de mieux répondre à des exigences en matière de décomposition, de composition, de réutilisation et d'administration, le concept d'agent peut être vu comme une réponse en matière d'autonomie et de socialité.

1.3.4.1. *Principales exigences couvertes*

Le style SMA répond à différentes exigences relatives à la réalisation de systèmes complexes et permet une certaine maîtrise de cette complexité. Au niveau du « produit », les principales exigences auxquelles il permet de répondre sont :

- l'intégration de l'application dans des systèmes physiques et sociaux distribués (cette distribution « de fait » s'est généralisée avec l'omniprésence de réseaux de communication performants et pousse à la décentralisation des solutions et à l'autonomie) ;
- la dynamique et l'ouverture, voire le non déterminisme, de ces systèmes (contraintes auxquelles les SMA répondent par une adaptation dynamique constante et une localité des traitements) ;
- l'évolutivité et la capacité d'apprentissage, l'adaptabilité, « l'intelligence » afin d'épauler l'humain ;
- la robustesse, le passage à l'échelle et la performance (auxquelles les SMA répondent par la façon de décomposer le système, la multiplication du nombre d'agents et le potentiel de parallélisation) ;

Les SMA s'inscrivent pleinement dans l'évolution du génie logiciel dans le sens d'une conception en termes d'unités en interaction [FER 95]. Ainsi, concernant la conception et la réalisation, le style SMA répond aux exigences suivantes :

- la maîtrise de la complexité des problèmes à traiter et des applications à développer (résolution de problèmes ou simulation) avec l'absence d'un modèle global de solution qui conduit à choisir une distribution logique en tant que fondement de la

solution accompagnée par une vision locale centrée sur les agents et leurs interactions plutôt qu'une vision globale (en outre, la combinatoire – complexité structurelle – de la solution croît linéairement – pas exponentiellement – avec la taille du problème de par leur proximité structurelle) ;

- l'expressivité et le niveau d'abstraction (cf. section 1.3.2.1) qui facilitent le développement de tels systèmes naturellement concurrents, basés sur des interactions complexes entre entités anthropomorphes qui peuvent être situées dans un environnement ;
- la modifiabilité et la modularité (multiplicité des comportements des agents), en particulier dans le cadre d'un développement incrémental, fréquent dans la pratique des SMA.

Le lecteur trouvera dans [BOI 01] une énumération des principales caractéristiques des SMA et une discussion sur la manière dont elles répondent aux exigences non fonctionnelles.

1.3.4.2. *Vues « module » et « allocation »*

Le style SMA vise directement les vues architecturales de la catégorie « composants et connecteurs » comme nous l'avons indiqué en 1.3.1. Ici, nous discutons les liens qui existent avec les vues des catégories « module » et « allocation » [CLE 03].

On peut noter que le paradigme SMA répond indirectement à des exigences auxquelles répondent les vues de la catégorie « modules ». D'une part, la proximité entre le problème et la solution (cf. Section 1.3.2.1) conduit naturellement à la séparation des préoccupations et à la modularité de la solution.

Ceci concerne le système mais aussi les agents pour lesquels la séparation entre les préoccupations à caractère social et celles à caractère décisionnel est exploitable lors du développement.

En revanche, en dehors de la séparation des préoccupations, les apports du style SMA aux vues de la catégorie « module » sont inexistant en terme de structuration du code (à l'inverse des objets avec l'organisation à base de classes). Nous touchons là une difficulté majeure en matière de diffusion de cette technologie : la question de la réalisation du SMA est complètement ouverte et peu ou pas encadrée (dans la section précédente, nous avons vu que les méthodes et outils couvrent essentiellement les parties amont du développement). Il est donc nécessaire de proposer des modèles, des méthodes et des outils afin de faciliter la conception et la réalisation des SMA et, entre autres, de promouvoir la réutilisation.

Notons, par ailleurs, que la modularité et la séparation des préoccupations ont aussi un impact sur la définition de vues de la catégorie « allocation ». D'une part, l'allocation des tâches aux équipes de développement selon leurs compétences est facilitée, ce qui encourage la participation d'experts du domaine applicatif au développement. Ceci

est particulièrement vrai dans le domaine de la simulation multi-agent dans lequel les experts du domaine (appelés « thématiciens ») sont des parties prenantes et participent à la conception avec les informaticiens développeurs. D'autre part, les SMA étant par nature fortement distribués, l'allocation des ressources physiques aux éléments du SMA est généralement directe.

1.3.4.3. *Documentation*

Dans cette partie, nous discutons le lien entre le style SMA et les exigences en matière de documentation.

Comme nous l'avons indiqué dans la section 1.3.3, les connecteurs d'interaction possèdent peu de sens en terme de composition de fonctionnalité. Le couplage entre agents est principalement sémantique, alors que dans le cadre des architectures à base de composants logiciels ou de services, les descriptions de type « composants et connecteurs » expriment, à travers les interfaces, comment le système se comporte à l'exécution. Le fonctionnement du SMA se retrouve dans la description des comportements sociaux des agents (protocoles d'interaction et échanges de données, rôles...) et de la dynamique de l'environnement, en particulier en tant que « bus logiciel » médiateur des interactions. En guise d'illustration, considérons un ensemble d'agents situés dans un espace qui communiquent par envoi de messages à portée (distance) limitée. C'est bien la dynamique de l'environnement et des agents dans celui-ci, à travers leurs déplacements, qui explique comment le système fonctionne, plus que la description des liens structurels qui existent entre les agents.

Ce n'est donc pas tant l'utilisation de la nomenclature typiquement utilisées par les vues « composants et connecteurs » (éléments, relations, interfaces) qui permettent de documenter de manière précise et concise une architecture SMA et les décisions architecturales qui répondent aux exigences, mais bien la description des éléments eux-mêmes (comportements, protocoles d'interaction, dynamique de l'environnement...).

Ainsi, comme cela a été présenté dans la section 1.2.5.2, documenter un SMA passe très souvent par l'utilisation de modèles propres à une approche, plutôt que de modèles classiquement utilisés dans les architectures logicielles comme celui proposé par Clements *et al.* [CLE 03]. En revanche, comme nous allons le voir dans la section 1.4, documenter le système en termes d'agents et de leur interactions est nécessaire mais pas suffisant.

1.3.4.4. *Exigences non couvertes*

Force est de constater qu'après une vingtaine d'années de recherches portées principalement par la communauté « intelligence artificielle distribuée » (beaucoup plus que par la communauté « génie logiciel »), et malgré tous les avantages que l'on peut espérer, les technologies SMA n'ont pour l'instant pénétré l'industrie du logiciel que de manière marginale. C'est essentiellement dans les milieux académiques que les

développements ont été effectués. Dans ce contexte particulier, il y a souvent peu de parties prenantes et il est fréquent que la même personne soit à la fois le client, le développeur et l'utilisateur. Quant aux produits développés, ce sont le plus souvent des prototypes, pas des logiciels industriels à longue durée de vie. Dans ce cadre, les exigences peuvent donc être assez différentes de celles que l'on rencontre dans le développement logiciel traditionnel. Nous énumérons ci-dessous quelques limitations des technologies SMA qui peuvent expliquer en quoi certaines exigences sont mal couvertes, particulièrement en matière de conception et de réalisation :

- les architectes logiciels manquent d'expertise sur les technologies SMA, et les équipes de développement manquent de connaissance et de formation (et, pour les entreprises, le retour sur investissement en termes de productivité et de qualité reste incertain) ;
- la technologie souffre encore d'un manque de maturité et de sa diversité (et il ne peut pas exister de méthode ou de support de mise en œuvre universel) ce qui est de nature à en limiter la prise en main ;
- les outils manquent, en particulier les outils industriels, ce qui limite l'utilisabilité et la productivité ;
- la technologie SMA pose des problèmes en termes de vérification et de validation (dûs en particulier à la forte décomposition) d'où le besoin de pouvoir au moins encadrer le développement par des méthodes solides.

1.3.5. Une famille de styles architecturaux

Le style SMA recouvre un ensemble de propriétés et de caractéristiques diverses que nous avons discutées dans cette section. Nous avons cependant vu que son utilisation dans le cadre d'un développement demandait un certain nombre de raffinements en fonction de l'approche choisie, du domaine métier, de choix de conception (par exemple, types d'agents et mécanismes d'interaction nécessaires). Ces raffinements conduisent à la définition de styles d'architecture plus concrets, exploitables en pratique pour un problème donné. Plus qu'un style unique d'architecture, c'est donc une famille de styles architecturaux de type SMA qui existe.

Ce point est lié à un autre que nous avons déjà souligné : il existe un fossé entre la conception et la réalisation des SMA qui doit être comblé de telle sorte que les SMA puissent être produits dans le cadre de processus de développement sûrs et efficaces. À défaut, c'est un surcroît de travail qui est demandé au développeur pour une moindre qualité du logiciel produit. Ceci s'ajoute aux limites discutées dans la partie précédente. Dans la section suivante, nous analysons la nature de ce fossé, puis nous décrivons les grandes lignes d'une proposition de solution dans la section 1.5.

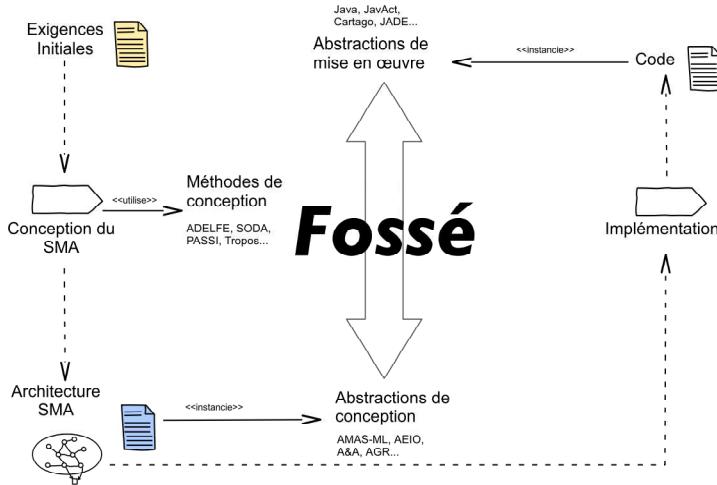


Figure 1.2. Le fossé architectural dans les SMA

1.4. Le fossé architectural

Dans le développement des SMA, on retrouve classiquement les phases de conception et d'implémentation. Mais, ici, comme nous l'avons déjà évoqué, le passage de l'une à l'autre n'est pas immédiat : la distance entre les concepts sur lesquels repose la conception et ceux disponibles pour l'implémentation (c'est-à-dire ceux qui sont fournis par le support de mise en œuvre) crée un fossé qui doit être comblé. Ce problème a souvent été abordé et il est présenté, par exemple, dans [BER 05]. Pour A. Molesini *et al.* [MOL 07], il réside dans le fait que les méta-modèles de SMA visent l'expression de solutions à base d'agents, alors que les meta-modèles des supports de développement s'inspirent généralement des concepts du monde « objet ». Pour notre part, dans la section précédente, nous avons souligné le haut niveau d'abstraction du style SMA, le besoin de raffinement de ce style et l'absence (dans le cas général) de support qui couvre les besoins précis de l'approche et de l'application.

L'objet de cette section est de mettre en évidence et de discuter la nature de ce fossé architectural (cf. figure 1.2). Nous décrivons tout d'abord l'état de la pratique, puis nous analysons la question du fossé d'un point de vue architectural.

1.4.1. Etat de la pratique

Une observation des pratiques montre que le développement des SMA repose, le plus souvent, sur l'un des enchaînements de tâches suivants :

- Choisir une approche SMA, concevoir le système, puis choisir un support de mise en œuvre qui convient à l'approche choisie et, finalement, planter le système.
- Choisir un support de mise en œuvre en amont de la conception, puis choisir une approche SMA compatible avec ce support, réaliser la conception en tenant compte des contraintes qui résultent de ces choix, puis planter le système.
- Choisir une approche SMA, concevoir le système, puis l'implanter de manière *ad hoc* sans utiliser un support de mise en œuvre adapté, ce qui revient en quelque sorte à développer (en partie) un support de mise en œuvre pour répondre aux besoins de l'approche choisie.

Dans les deux premiers cas, il s'avère qu'un effort est nécessaire pour adapter le support à la conception (réaliser des développements au niveau du support pour modifier les abstractions disponibles ou en ajouter) ou adapter la conception au support (ce qui conduit à déformer la conception pour la faire correspondre au support et ainsi à s'éloigner des intentions premières du concepteur du SMA). De manière générale, même si certaines abstractions de conception peuvent se retrouver directement dans le support de mise en œuvre (par exemple, un moyen de programmer la décision à base de règles, ou un mécanisme d'interaction de haut niveau), les supports existants ne peuvent pas répondre précisément à toutes les exigences induites par l'approche et les choix de conception (ceci dès que l'on veut produire de véritables logiciels de qualité et pas de simples prototypes).

Le troisième cas se rencontre également, par exemple dans le domaine de la simulation multi-agent où la conception est fortement contrainte par le domaine applicatif et ses abstractions spécifiques (typiquement, on peut avoir à modéliser des interactions réelles donc complexes entre individus). On est alors conduit à développer des solutions *ad hoc*, ce qui est coûteux et relativement inefficace dès que ce travail n'est pas encadré.

1.4.2. Analyse d'un point de vue architectural

1.4.2.1. Vues « macro » et « micro »

Comme pour tout développement logiciel, différentes exigences sont formulées initialement puis pendant le développement d'un SMA, et les exigences initiales expriment les caractéristiques et les propriétés de la solution et de sa réalisation.

En s'appuyant sur une approche et un certain niveau d'abstraction, la conception du SMA répond dans un premier temps à un sous-ensemble de ces exigences : habituellement, celles-ci concernent les fonctionnalités du système et le domaine métier, mais aussi ses capacités d'adaptation, son extensibilité, la performance... Plus généralement, il s'agit des exigences auxquelles le style SMA et l'approche choisie permettent de répondre naturellement (cf. section 1.3.4). Leur prise en compte conduit

à des choix architecturaux que nous qualifions de « macro ». Ainsi, la vue architecturale « macro » de la solution définit les types d'agent, leurs comportements, leurs interactions, les éléments de l'environnement...

Néanmoins, le travail de conception architecturale ne s'arrête pas là. D'une part, il existe des exigences initiales non prises en compte à ce stade. Celles-ci peuvent, par exemple, être relatives à l'interface avec l'utilisateur ou avec le système physique auquel s'intègre le SMA, ou à différentes exigences liées à l'organisation du développement (par exemple, la testabilité). D'autre part, la conception de niveau macro produit de nouvelles exigences en matière de réalisation. Ces dernières peuvent, par exemple, concerter la réalisation de moyens d'interaction particuliers, la dynamique d'exécution et le fonctionnement des agents ou de l'environnement, etc. Ces exigences expriment les préoccupations à caractère « opératoire ». Elles proviennent de choix de conception, de l'approche ou de la méthode choisies. Par exemple, certaines approches imposent aux agents une prise de décision à base de règles ou un modèle d'organisation particulier ce qui contraint le support de mise en œuvre à fournir ces abstractions.

Toutes ces exigences doivent être prises en compte après que la vue macro ait été explicitée. Puisqu'il n'est pas possible, en général, de disposer d'un support de développement adéquat, cette prise en compte conduit à une nouvelle étape de conception architecturale que nous qualifions de « micro », donc à des choix qui concernent, non pas le SMA dans son ensemble, mais certains mécanismes spécifiques sur lesquels le fonctionnement du SMA repose. Ainsi, on donne une sémantique opérationnelle aux abstractions utilisées au niveau macro. La vue architecturale « micro » de la solution définit comment se décomposent certains éléments du système, principalement les agents, donc leur architecture interne et ses connexions avec les autres éléments (par exemple, les connexions entre agent et environnement). La vue micro peut être considérée comme une sorte de raffinement de la vue macro. La conception de cette architecture de niveau micro, participe à la production d'un support pour la mise en œuvre du SMA conçu et spécialisé pour celui-ci mais dont la réalisation peut se faire indépendamment de celle du SMA.

Les catégories de vue macro et micro reflètent des points de vue différents sur l'architecture et la prise en compte d'exigences de niveaux différents, et leur différenciation reflète la manière dont s'enchâînent des activités dans la conception de SMA (cf. figure 1.3). Les vues macro et micro ne remplacent pas, mais complémentent, les autres vues architecturales.

1.4.2.2. *Séparation des préoccupations et des métiers*

La distinction entre les vues macro et micro reflète la séparation entre les préoccupations liées au SMA et les préoccupations opératoires.

La production des vues macro et micro fait appel à des compétences différentes. La vue macro est produite par le concepteur du SMA, celui-ci ayant des compétences

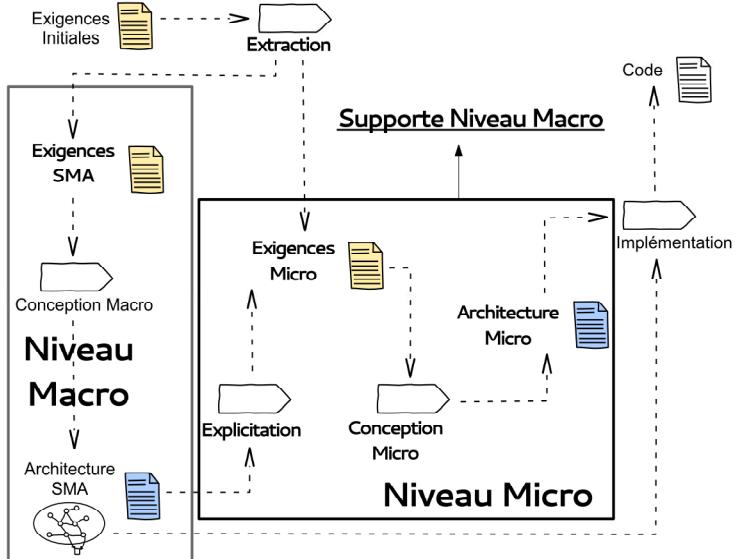


Figure 1.3. Les différents niveaux de conception architecturale dans les SMA

dans le domaine des systèmes multi-agents, des approches et des méthodes de développement. La vue micro est produite par un concepteur ayant des compétences en matière de protocole de communication, de synchronisation, d'ordonnancement, de middleware... Au final, l'implémentation doit réaliser à la fois l'architecture macro et l'architecture micro mais il est important de ne pas reléguer à la phase d'implémentation les choix architecturaux de niveau micro.

Dans la pratique, il est fréquent qu'une même personne réalise le développement du SMA dans son intégralité (conception et implémentation), au risque de mélanger la réalisation du niveau macro (au moyen d'un support de mise en œuvre) et celle du niveau micro (c'est-à-dire le support sur lequel s'appuie le niveau macro), et ainsi de ne plus séparer les préoccupations. Alors, la prise en compte des exigences ne suit plus un ordre de priorité bien défini, ce qui nuit à la qualité de l'architecture produite. Les limites, en termes de compétences des développeurs, peuvent aussi être à l'origine d'une perte de qualité du produit. Cet état de fait n'est pas propre au développement de SMA mais il est particulièrement prégnant dans ce cadre à cause de la pluralité des concepts que regroupe le paradigme SMA et qui donne la grande diversité des outils que nous avons présentée dans la section 1.2.5.

1.4.3. *Bilan*

En tant que famille de styles architecturaux, qui possèdent des points communs mais aussi d'importantes différences, les SMA conduisent le développeur à raffiner les abstractions utilisées afin de les faire correspondre à l'approche utilisée et au problème. Il doit donc travailler sur l'architecture, non pas du système, mais de ce qui va fournir les abstractions sur lesquelles repose la conception macro, c'est-à-dire l'architecture de niveau micro. Les principales abstractions concernées sont les connecteurs d'interaction et les types d'agent, ces derniers servant de passerelles entre la conception macro qui les utilise et la conception micro qui les réalise. La conception architecturale de niveau micro vise donc la production d'un support architectural sur lequel s'appuie l'architecture macro, et qui, une fois implémenté, est utilisable pour implémenter la conception macro.

Pour une meilleure qualité des SMA produits, il faut donc organiser le développement en séparant le développement du SMA lui-même et celui d'un support spécialisé pour sa mise en œuvre. De cette façon, on prend mieux en compte des exigences qui, dans la pratique, sont souvent implicites ou prises en compte trop tard. Ceci met aussi en évidence l'existence de deux rôles distincts dans le développement, qui demandent des connaissances et des compétences différentes.

1.5. Comment combler le fossé architectural

Le propos de cette section est de présenter, dans ses grandes lignes, une solution pour le développement de SMA, élaborée afin d'atteindre les objectifs annoncés dans la section 1.4.3. En outre, l'objectif est de faciliter la réutilisation de manière à satisfaire les exigences en matière de productivité et de qualité de la solution produite. Or, la spécialisation du support de mise en œuvre va à l'encontre d'une éventuelle réutilisation de celui-ci. Dans cette section, nous verrons que la réutilisation peut être favorisée dans le cadre d'une solution de conception architecturale à base de composants logiciels. Pour plus de détails sur cette solution, le lecteur peut se reporter au mémoire de thèse de l'un des auteurs [NOë 12].

1.5.1. *Limites des solutions existantes*

Comme cela a été présenté dans la partie 1.2.5, il existe de nombreuses réponses à la question du fossé soulevée ici, bien que cette question n'ait pas été effectivement formulée en ces termes. Il s'agit essentiellement de supports orientés agent qui servent à implémenter la conception macro, ceux-ci étant plus ou moins flexibles pour ce qui concerne l'adaptation des abstractions fournies, et d'approches orientées modèles avec lesquelles l'implémentation est générée à partir d'une description de la conception macro. Ces propositions tentent de répondre au problème du fossé par un

style SMA unificateur, c'est-à-dire avec un raffinement particulier au sein de la famille des styles SMA. Malheureusement, comme nous l'avons souligné précédemment, il existe une telle diversité de types d'agent, de types d'environnement et de mécanismes d'interaction, qu'il n'est pas possible de définir des abstractions qui soient raffinables pour tout type de problème et de conception macro.

Lorsque l'on se restreint à une approche voire à un domaine, ces solutions proposent des abstractions adéquates, mais d'un point de vue architectural, de nombreuses exigences sont spécifiques à l'application à développer, qu'il s'agisse d'exigences initiales ou d'exigences extraites de la conception. Ces exigences ont des impacts sur les abstractions nécessaires à l'implémentation (par exemple l'exécution, la connexion à des systèmes externes ou à des interfaces graphiques spécifiques, etc.), ce qui conduit à « bricoler » les supports de mise en œuvre pour implémenter complètement l'application.

1.5.2. Réalisation de l'architecture micro

De l'analyse faite précédemment, nous déduisons que le comblement du fossé est une question architecturale et non une question d'implémentation. En conséquence, plutôt que de tenter d'adapter des abstractions existantes, nous proposons donc de fabriquer les abstractions répondant aux exigences micro et permettant d'implémenter la conception macro. De notre point de vue, c'est la seule manière d'aborder le problème de façon satisfaisante.

Cette fabrication doit être encadrée et doit permettre la conception et la réutilisation d'éléments récurrents de solution, en particulier les mécanismes d'interaction, et la définition des éléments propres à chaque application, en particulier les types d'agents. L'agent lui-même est un système (au sein d'un système multi-agent) qui doit faire l'objet d'une conception concernant, non seulement ses aspects fonctionnels, mais aussi ses aspects opératoires.

En matière de conception, cette proposition permet de séparer le rôle de concepteur de la vue macro de celui de concepteur de la vue micro. Et en matière d'implémentation, le rôle d'implémenteur du SMA est séparé du rôle d'implémenteur des mécanismes opératoires du logiciel.

1.5.2.1. Défis

À partir de notre expérience dans le développement de SMA, que ce soit au moyen de supports de mise en œuvre génériques (donc non directement adaptés à la conception macro) ou seulement d'un langage de programmation traditionnel, et à partir de l'étude des supports existants, nous avons identifié les défis suivants pour encadrer au mieux le développement de l'architecture micro :

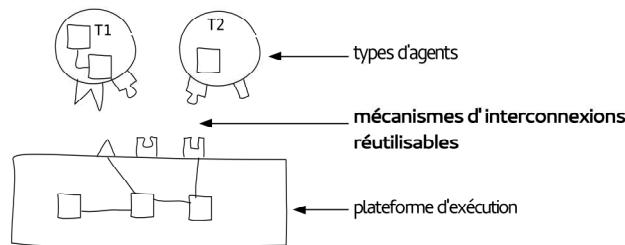


Figure 1.4. Approche proposée

- permettre la définition, l’implémentation et la réutilisation de mécanismes pour interconnecter à l’exécution les agents à leur plateforme d’exécution (alors que l’environnement est un concept qui se rapporte à la vue macro, la plateforme d’exécution se rapporte à la vue micro et réalise l’environnement et les aspects opératoires) ;
- permettre la définition de types d’agent qui exploitent ces mécanismes et qui possèdent des architectures internes ;
- permettre la création d’instances de ces types d’agents à l’exécution et leur connexion dynamique à la plateforme.

Le cœur du problème se situe au niveau de la définition de mécanismes d’interconnexion (et non pas uniquement d’interaction, par exemple interfaces graphiques, cadencement du système, etc) réutilisables et composables pour définir des types d’agents adaptés à l’application et utilisables pour la conception macro.

1.5.2.2. Une solution à base de composants logiciels

Les supports de mise en œuvre existants ont une architecture dans laquelle les agents sont connectés à une plateforme d’exécution. Répondre aux défis présentés précédemment, c’est faciliter la réalisation de tels supports, adaptés à la mise en œuvre de l’application SMA visée.

Dans la pratique, nous proposons de fabriquer l’architecture micro à l’aide d’abstractions similaires à celles disponibles dans les architectures à composants et la programmation orientée composants. Nos travaux nous ont amenés à définir un nouveau type de composant qui permet de construire des mécanismes d’interconnexion composés de deux parties : une liée à la plateforme et une liée aux agents. Ainsi, les types d’agents sont définis comme une composition des parties agent de ces mécanismes d’interconnexion et de composants traditionnels. Il en est de même pour la plateforme d’exécution qui contient les parties plateforme des mécanismes utilisés par les types d’agents (cf. figure 1.4).

Ces abstractions permettent de fabriquer des mécanismes spécifiques à une application, tels que des interfaces graphiques ou des représentations de l'environnement, mais aussi des mécanismes opératoires potentiellement réutilisables, tels que des mécanismes d'interaction ou d'exécution, de connexion à l'environnement système du SMA, etc. Elles permettent aussi d'automatiser, lors de la création d'un agent, l'instanciation des différentes parties de l'agent et leur connexions à la plateforme, ce qui simplifie le développement.

Ainsi, définir l'architecture micro d'un SMA revient à :

- définir la plateforme d'exécution au moyen de composants et de mécanismes d'interconnexion mis à disposition des agents (spécifiques à l'application ou réutilisés) ;
- définir les types d'agents au moyen de composants (spécifiques à l'application ou réutilisés) et en utilisant les mécanismes d'interconnexion de la plateforme.

Il ne reste ensuite qu'à implémenter les composants et mécanismes d'interconnexions spécifiques à l'application, définir comment initialiser le SMA (création des agents en particulier) et le système est prêt à être exécuté.

Cette solution a été mise en œuvre en pratique, et expérimentée dans le cadre de plusieurs projets de recherche. Concrètement, elle repose sur un modèle de composant (SpeAD, *Species-based Architectural Design*), un langage de description d'architecture (SpeADL, *Species-based Architectural Description Language*) et une méthode de conception (SpEARAF, *Species to Engineer Architectures for Agent Frameworks*) qui facilite et guide la conception et l'implémentation de SMA en suivant ces modèles [NOë 12]. Un outil, MAY⁶, qui permet d'utiliser SpeADL avec Java, est mis à disposition des utilisateurs sous forme d'un plugin Eclipse, ainsi qu'une collection de composants réutilisables pour divers mécanismes d'interconnexions et pour les architectures internes des agents et de l'environnement.

1.6. Conclusion

1.7. Bibliographie

[AGH 86] AGHA G., Ed., *Actors : A Model of Concurrent Computation in Distributed Systems*, MIT Press, Cambridge, MA, USA, 1986.

[AMB 03] AMBLARD F., Comprendre le fonctionnement de simulations sociales individus-centrées - Application à des modèles de dynamiques d'opinions, Thèse de doctorat, Université Blaise Pascal, Clermont-Ferrand, France, 2003.

6. <http://www.irit.fr/MAY>

- [BER 04] BERGENTI F., GLEIZES M.-P., ZAMBONELLI F., Eds., *Methodologies and Software Engineering for Agent Systems*, Klüwer Academic Press, 2004.
- [BER 05] BERNON C., COSENTINO M., PAVÓN J., « An overview of current trends in european AOSE research », *Informatica*, vol. 29, p. 379–390, 2005.
- [BOI 01] BOISSIER O., GITTON S., GLIZE P., « Caractéristiques des systèmes et des applications multi-agents », BRIOT J.-P., DEMAZEAU Y., Eds., *Principes et architectures des systèmes multi-agents*, Chapitre 1, Hermès, Paris, France, 2001.
- [BRI 01] BRIOT J.-P., DEMAZEAU Y., Eds., *Principes et architectures des systèmes multi-agents*, Traité IC2, Informatique et systèmes d'information, Hermès, Paris, France, 2001.
- [BRI 09] BRIOT J.-P., « Composants logiciels et systèmes multi-agents », EL FALLAH SEGH-ROUCHNI A., BRIOT J.-P., Eds., *Technologies des systèmes multi-agents et applications industrielles*, Chapitre 5, Lavoisier, Paris, France, 2009.
- [CLE 03] CLEMENTS P., BACHMANN F., BASS L., GARLAN D., IVERS J., LITTLE R., NORD R., STAFFORD J., *Documenting Software Architectures : Views and Beyond*, Addison-Wesley, Boston, MA, USA, 2nd édition, 2003.
- [DEM 95] DEMAZEAU Y., « From interactions to collective behaviour in agent-based systems », *Proceedings of the First European conference on cognitive science*, Saint Malo, France, p. 117–132, April 1995.
- [DIM 11] DI MARZO SERUGENDO G., GLEIZES M.-P., KARAGEORGOS A., *Self-organising Software - From Natural to Artificial Adaptation*, Natural Computing Series, Springer, <http://www.springerlink.com>, octobre 2011.
- [FAV 06] FAVRE J.-M., ESTABLIER J., BLAY-FORNARINO M., Eds., *L'ingénierie dirigée par les modèles : au-delà du MDA*, Hermes-Lavoisier, Cachan, France, février 2006.
- [FER 95] FERBER J., *Les systèmes multi-agents - Vers une intelligence collective*, InterEditions, 1995.
- [FER 04] FERBER J., GUTKNECHT O., MICHEL F., « From Agents to Organizations : An Organizational View of Multi-agent Systems », GIORGINI P., MÜLLER J. P., ODELL J., Eds., *Agent-Oriented Software Engineering IV, Proceedings of Int. Workshop AOSE 2003*, Lecture Notes in Computer Science ,2935, Springer, p. 214–230, 2004.
- [FIP02] FIPA, FIPA ACL Message Structure Specification, 2002, FIPA document SC00061G : www.fipa.org/specs/fipa00061/SC00061G.html.
- [FUG 98] FUGGETTA A., PICCO G. P., VIGNA G., « Understanding Code Mobility », *IEEE Transactions on Software Engineering*, vol. 24, n° 5, p. 342-361, 1998.
- [GLE 08] GLEIZES M.-P., BERNON C., MIGEON F., PICARD G., « Méthodes de développement de systèmes multi-agents », *Génie Logiciel*, vol. 86, p. 2–7, GL & IS, septembre 2008.
- [GRA 59] GRASSÉ P.-P., « La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. la théorie de la stigmergie : Essai d'interprétation du comportement des termites constructeurs », *Insectes Sociaux*, vol. 6, n° 1, p. 41–80, 1959.

- [HEW 77] HEWITT C., « Viewing Control Structures as Patterns of Passing Messages », *Artificial Intelligence*, vol. 8, n° 3, p. 323-364, 1977.
- [KAD 11] KADDOUM E., Optimisation sous contraintes de problèmes distribués par auto-organisation coopérative, PhD thesis, Université Paul Sabatier, Toulouse, France, 2011.
- [MOL 07] MOLESINI A., DENTI E., OMICINI A., « From AOSE Methodologies to MAS Infrastructures : The SODA Case Study », ARTIKIS A., O'HARE G., STATHIS K., VOURROS G., Eds., *8th International Workshop “Engineering Societies in the Agents World” (ESAW'07)*, Athens, Greece, NCSR “Demokritos”, p. 283–298, 10 2007, Workshop Notes.
- [NOË 12] NOËL V., Component-based Software Architectures and Multi-Agent Systems : Mutual and Complementary Contributions for Supporting Software Development, PhD thesis, Université Paul Sabatier, Toulouse, France, 2012.
- [ODE 03] ODELL J., VAN DYKE PARUNAK H., FLEISCHER M., « The Role of Roles », *Journal of Object Technology*, vol. 2, n° 1, p. 39–51, 2003.
- [OLU 07] OLUYOMI A., KARUNASEKERA S., STERLING L., « A comprehensive view of agent-oriented patterns », *Autonomous Agents and Multi-Agent Systems*, vol. 15, p. 337–377, Springer Netherlands, 2007.
- [RAO 95] RAO A. S., GEORGEFF M. P., « BDI-agents : from theory to practice », *Proceedings of the First International Conference on Multiagent Systems*, p. 312-319, 1995.
- [SHE 06] SHEN W., HAO Q., YOON H. J., NORRIE D. H., « Applications of agent-based systems in intelligent manufacturing : An updated review », *Advanced Engineering Informatics*, vol. 20, n° 4, p. 415–431, 2006.
- [WEI 99] WEISS G., Ed., *Multiagent systems, a Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge, MA, USA, 1999.
- [WEY 07] WEYNNS D., OMICINI A., ODELL J., « Environment as a first-class abstraction in multiagent systems », *Autonomous Agents and Multi-Agent Systems*, vol. 14, n° 1, p. 5-30, 2007.
- [WEY 10] WEYNNS D., Ed., *Architecture-Based Design of Multi-Agent Systems*, Springer, Berlin, Heidelberg, 2010.
- [WOO 09] WOOLDRIDGE M., Ed., *An Introduction to Multiagent Systems*, John Wiley & sons, Chichester, England, 2nd édition, 2009.

