Chapitre 2

Méthodes orientées agent et multi-agent

2.1. Introduction

Les systèmes multi-agents (SMA) ont montré leur pertinence pour la conception d'applications distribuées (logiquement ou physiquement), complexes et robustes. Le concept d'agent est aujourd'hui plus qu'une technologie efficace, il représente un nouveau paradigme pour le développement de logiciels dans lesquels l'agent est un logiciel autonome qui a un objectif, évolue dans un environnement et interagit avec d'autres agents au moyen de langages et de protocoles (voir le chapitre 1 « Introduction aux systèmes multi-agents »). Souvent, l'agent est considéré comme un objet « intelligent » ou comme un niveau d'abstraction au-dessus des objets et des composants (voir le chapitre 5 « Composants logiciels et systèmes multi-agents »). Les méthodes de développement orientées objet – au vu des différences entre les objets et les agents – ne sont pas directement applicables au développement de SMA. Il est alors devenu nécessaire d'étendre ou de développer de nouveaux modèles, de nouvelles méthodologies et de nouveaux outils adaptés au concept d'agent.

Après la révolution de la conception et de la programmation orientée objet, nous sommes donc à l'aube d'une nouvelle révolution qui serait celle de la conception et de la programmation orientée agent/interaction/organisation. Le constat concernant la conception de SMA, dans les années 1998-1999, est que le développement de SMA est coûteux en temps à cause de leur complexité, mais aussi par le fait que pour chaque application, il faut créer tout le SMA adéquat. En effet, la majorité des applications existantes en SMA ont été développées de manière *ad hoc* [TRE 98]. Ce foisonnement a conduit en parallèle à de multiples propositions de modèles d'agents, notamment par

---

Chapitre rédigé par Carole **Bernon**, Marie-Pierre **Gleizes** et Gauthier **Picard**.

le rapprochement effectué avec l'approche objet. De là, plusieurs formalismes sont apparus, chacun mettant en avant une représentation de l'agent et de son système. En 1999, se fait donc sentir le besoin de fournir des modèles, des méthodologies, des plates-formes pour faciliter la prise en compte de la complexité des systèmes à concevoir [JEN 99] et pour aider les concepteurs qui ne sont pas nécessairement spécialistes des SMA. Parallèlement au développement des méthodes, des modèles ont été définis pour aider les concepteurs tels que AGR [FER 98] ou BDI [RAO 95]. Pour le développement, les concepteurs disposent de langages orientés agent dont les précurseurs ont été Agent0 et son évolution PLACA, ou LALO et METATEM. Des approches impératives (comme JAL[1]), déclaratives (comme CLAIM [ELF 03]) ou hybrides (comme 3APL[2]) ont vu le jour depuis [BOR 06], liées ou non à des plates-formes de développement dont les plus célèbres sont Madkit[3] ou JADE[4]. Bien entendu, l'utilisation de langages de programmation classiques (C++, Java, etc.) reste envisageable.

Les grandes familles de travaux concernant les méthodes orientées agent étendent les concepts soit des méthodes orientées objet, soit des méthodes issues de l'intelligence artificielle et plus particulièrement de l'ingénierie des connaissances, soit utilisent les deux [IGL 99]. Les premiers travaux sur les méthodologies sont ceux sur AAII [KIN 96], sur Cassiopée [COL 96] ou sur DESIRE [BRA 97]. En 2000, cet axe des SMA se confirme et les aspects ingénierie des SMA représentent alors le thème de groupes de travail au niveau national, avec le groupe Architecture et société d'agents (ASA) de l'Association française d'intelligence artificielle (PRC-GDRI3-AFIA) ou européen avec les groupes Methodologies and Software Engineering for Agent Systems (MSEAS) d'AgentLink II (2000-2003) et Agent-oriented Sotware Engineering (AOSE) d'AgentLink III (2003-2005) ainsi que le thème de conférences ou de workshops comme AOIS (Agent-oriented Information Systems), AOM (Agent-oriented Methodologies), AOSE (Agent-oriented Software Engineering), ESOA (Engineering Self-Organising Applications), ou les JFIADSMA (1999 et 2000). De nombreuses méthodologies sont alors conçues [BER 04, HEN 05], comme, par exemple, ADELFE [PIC 04], Gaia [CER 04a], INGENIAS [GOM 02], MaSE [DEL 01], PASSI [COS 02], Prometheus [PAD 03], Tropos [CAS 01] et Voyelles [DEM 01].

Contrairement aux méthodes orientées objet, poussées et portées par des industriels, les méthodes orientées agents sont développées et essentiellement utilisées dans le monde académique même si la motivation première de leurs concepteurs était de promouvoir la programmation et le développement orientés agent dans un contexte industriel. Toutefois, bien que n'étant pas moteurs, les industriels sont aussi intéressés par une approche d'ingénierie basée sur leurs exigences qui prennent en compte tout

1. http://www.agent-software.com.
2. http://www.cs.uu.nl/3apl/.
3. http://www.madkit.org.
4. http://jade.tilab.com.

le cycle de vie du logiciel [BUS 98] comme en témoignent AML (*agent modeling language*) et ADEM (*agent development methodology*), le processus de développement associé, de Whitestein Technologies[5] [CER 04b]. Actuellement, on peut constater que de nombreuses méthodologies ont été développées [BER 04, HEN 05] et que les travaux s'orientent vers la conception de métamodèles et vers la prise en compte des phases de développement de test et de déploiement [GAR 03]. En effet, l'organisme FIPA[6] (Foundation for Intelligent Physical Agents) œuvre dans le sens d'une standardisation d'une méthode au travers de la définition d'un métamodèle [ODE 04b]. Les premiers travaux sur la standardisation dans le domaine des agents a eu pour objectif de faire communiquer des agents développés par différents concepteurs. Pour mettre en œuvre cette interopérabilité, le langage ACL (*agent communication language*) est devenu le langage standard de communication entre les agents. Une architecture et des protocoles ont aussi été développés par la FIPA pour prendre en charge les agents communicants.

L'objectif de ce chapitre est de donner un panorama des principales méthodes de conception de SMA et de la standardisation des techniques dans ce domaine. Les méthodes seront décrites en suivant un même plan pour faciliter leur comparaison. Des prospectives à court, moyen et long termes seront ensuite exprimées avant de conclure ce chapitre.

2.2. Les méthodes de conception de systèmes multi-agents

Une méthodologie doit permettre de faciliter le processus d'ingénierie des systèmes. [BOO 92] donne la définition suivante d'une méthodologie : « Une méthodologie est un ensemble de méthodes appliquées tout au long du cycle de développement d'un logiciel, ces méthodes étant unifiées par une certaine approche philosophique générale. » Pour [SHE 01], une méthodologie est un ensemble de guides qui couvrent tout le cycle de vie du développement d'un logiciel, ils sont à la fois techniques et gèrent le projet. Une méthode est définie comme « un processus rigoureux permettant de générer un ensemble de modèles qui décrivent divers aspects d'un logiciel en cours de développement en utilisant une certaine notation bien définie ». Une méthode de développement de systèmes multi-agents est constituée d'un processus, de notations et d'outils pour prendre en charge ce processus et ces notations et/ou pour aider le développeur ; on parle alors de CAME (*computer aided method engineering*).

Les méthodes de conception orientée agent visent donc à construire des systèmes où les compétences sont attribuées à des entités logicielles autonomes, qui interagissent dans un environnement commun et peuvent avoir la faculté de naviguer dans

5. http://www.whitestein.com.

6. http://www.fipa.org.

les réseaux pour accomplir des tâches relevant de leurs compétences définies au moment de la conception du système. Ce qui est complexe tient au fait qu'il n'est pas possible de définir à l'avance les séquences temporelles des différentes interactions entre les agents, bien qu'il soit possible de définir les règles qui régiront les interactions entre les différents agents en fonction de leurs modèles d'accointances, de leurs compétences et de leurs perceptions.

L'objectif d'une méthode est de guider un utilisateur tout au long du processus de développement, au niveau du développement lui-même mais aussi de la gestion et de la qualité. Il doit permettre de décrire la sémantique sans les détails d'implémentation et de fournir un moyen de vérifier, valider et tester les fonctionnalités du système. Il est actuellement largement admis que les phases principales d'une méthode sont les suivantes :

– l'*analyse des besoins*, appelée aussi spécification dans certaines méthodes, est l'étape au cours de laquelle ce que doit faire le système est exprimé du point de vue des utilisateurs. Cette étape peut aussi être divisée en deux sous-étapes :

- les *besoins préliminaires* représentent un travail d'intercompréhension et/ou une description consensuelle du problème du cahier des charges entre clients, utilisateurs et concepteurs sur ce que doit être et ce que doit faire le système, ses limites et ses contraintes. Cette phase doit permettre de transformer le cahier des charges ou des besoins exprimés par le client en un cahier des charges consensuel entre client et fournisseur,

- les *besoins finaux* fournissent un modèle de l'environnement. Les objectifs sont de définir une vue du système, d'organiser et de gérer les besoins (fonctionnels ou non) et leurs priorités. De manière pratique, à ce stade, le concepteur doit définir le système étudié et modéliser l'environnement du système ;

– l'*analyse*, appelée aussi conception de l'architecture, doit permettre de fournir la description des besoins de l'utilisateur en termes liés au domaine et à ce que le système doit faire. Elle décrit le problème à résoudre par le système s'il est clairement identifié. En effet, les applications dédiées aux systèmes multi-agents ne possèdent pas toutes une tâche clairement définie. Dans le premier cas, les produits de l'analyse sont utilisés pour définir les fonctions du système. Par fonction, on entend les comportements visibles et testables de l'extérieur du système. Dans ce dernier cas, la phase d'analyse doit permettre une description des entités réelles intervenant dans l'application. La phase d'analyse définit l'architecture du système. Dans le contexte SMA, les étapes essentielles sont l'identification des agents qui interviennent dans le domaine d'application et la définition de leurs interactions ;

– la *conception*, appelée aussi conception détaillée, a pour objectif de définir les entités et les mécanismes qui donnent le comportement défini par la phase d'analyse et l'architecture détaillée du système. Elle permet de régler les problèmes liés à l'implémentation. La conception consiste à dire comment le système va être construit à partir de l'architecture définie à la phase d'analyse. Cette phase peut démarrer dès qu'un modèle de ce que doit faire le système a été établi. Ce modèle va être enrichi

progressivement au cours de la phase de conception. A l'issue de la phase de conception, un modèle de l'architecture logicielle du système doit être réalisé et la conception détaillée fournit les composants logiciels à coder. La conception doit définir une architecture détaillée pour le système en termes de paquetages, de sous-systèmes, d'objets et d'agents ;

– le *développement* correspond aux phases classiques de codage, de test et d'intégration d'un logiciel. L'objectif de cette phase est de produire un code compilable ou interprétable. Au cours de cette opération, des outils sont fournis pour générer du code. Dans cette phase, la vérification et la validation du logiciel sont à réaliser. La vérification consiste à s'assurer qu'il n'y a plus d'erreurs durant l'exécution du logiciel alors que la validation consiste à vérifier que les besoins du client, exprimés lors de la phase d'analyse des besoins, sont satisfaits ;

– la phase de *maintenance* consiste soit à corriger d'éventuelles erreurs, soit à ajouter des fonctionnalités demandées par l'utilisateur.

2.3. Comparaison de méthodes

La problématique de la comparaison de méthodes a été souvent soulevée dans la littérature pour aider le concepteur à choisir la méthode la plus adaptée au problème donné ou pour exhiber les différences essentielles entre les différentes méthodes [HEN 05, SUD 04]. Dans ce sens, le TFG AOSE d'AgentLink III propose un questionnaire[7] pour l'évaluation des méthodes orientées agent et [ELA 06] propose une approche statistique pour évaluer leurs différences.

[DAM 03, SHE 01, STU 03] comparent les méthodes suivant quatre ou cinq axes : les concepts manipulés, les notations utilisées (et les techniques de modélisation), le processus de développement et la pragmatique. [CER 02] se focalisent seulement sur les concepts de modélisation des agents (les rôles, les buts, l'organisation, etc.), mais proposent une évaluation chiffrée par une somme pondérée des différentes prises en compte des concepts par les méthodes. Cette évaluation nécessite, par contre, une grande expertise dans toutes les méthodes et un grand nombre d'évaluateurs afin de déterminer au mieux les coefficients. [BER 02] ont de leur côté comparé des méthodes suivant huit axes : l'étendue de la couverture du processus, la spécialisation de la méthode à une application, l'architecture d'agent sous-jacente, l'utilisation de notations existantes, le domaine d'application (dynamique ou non), le modèle de rôle, le modèle de l'environnement et l'identification des agents.

La comparaison qui va suivre sera faite suivant les quatre axes de [DAM 03], mais en gardant en sous-catégorie les points de [BER 02], qui semblent mieux exhiber les caractéristiques propres aux agents.

7. http://www.pa.icar.cur.it/~cossentino/al3tf3/docs/questionnaire.doc.

2.3.1. *Les concepts-clés et les propriétés*

Afin d'évaluer une méthode, il est intéressant de vérifier quels concepts agents et multi-agents sont pris en compte [CER 02, DAM 03, SHE 01, STU 03]. Pour plus d'information, [BOI 04] recensent de manière exhaustive les caractéristiques propres aux systèmes multi-agents et à leurs applications. Citons par exemple adaptation, autonomie, but, concurrence, croyance, désir, distribution, environnement, groupe, intention, interaction, norme, organisation, ouverture, proactivité, protocole, réactivité, rôle ou tâche. Cet axe de comparaison va donc aussi s'attacher à évaluer les problèmes auxquels la méthode peut répondre.

2.3.2. *Langages de modélisation et notations*

Les langages de modélisation agent sont souvent graphiques, inspirés de l'ingénierie objet, des besoins ou des connaissances. Ils comportent des symboles, une syntaxe et une sémantique propres. Là encore, l'évaluation s'établit sur l'analyse des propriétés des langages utilisés [ARL 04, CER 02, DAM 03, SHE 01] :

– *accessibilité* : le langage est-il facile à apprendre et à utiliser ? Demande-t-il un long temps d'apprentissage ?

– *complexité* : est-il possible de visualiser la complexité du système à plusieurs niveaux d'abstraction ?

– *consistance* : est-il possible de vérifier la consistance d'une spécification ?

– *exécution* : une fonctionnalité de prototypage permettant des exécutions préliminaires existe-t-elle ?

– *expressivité* : tous les aspects de la conception de l'application sont-ils pris en compte par la notation ?

– *modularité* : la modification des spécifications en amont implique-t-elle une remise en cause de la totalité des spécifications ?

– *portabilité* : le langage est-il indépendant de quelconques choix de développement (langage de programmation par exemple, ou architecture) ?

– *précision* : le langage ne doit pas être ambigu. Des définitions précises sont-elles établies ?

– *raffinage* : le concepteur est-il guidé pour raffiner les modèles utilisés ?

– *traçabilité* : est-il facile de modifier des spécifications passées, de tracer ses changements sans avoir de répercussion sur les spécifications en cours ?

Tous ces critères sont à prendre en compte lors du choix d'une méthode utilisant un langage dédié ou bien lorsqu'un développeur veut lui-même définir son propre langage de modélisation.

2.3.3. *Le processus de développement*

Les cycles de vie et processus utilisés dans la conception de systèmes multi-agents sont de types classiquement utilisés en conception modulaire : cascades, itératifs, en V, en spirales. Or, toutes les méthodes ne couvrent pas tout ce cycle, et il est nécessaire de le prendre en compte lors de l'évaluation d'une méthode. Pour les méthodes proposées nous allons aussi analyser :

– *le contexte de développement*, c'est-à-dire le processus est-il adéquat pour le développement de nouvelles applications, la réingénierie (*reengineering*), le prototypage, la réutilisation, pour faire appel à des schémas de conception préexistants, utiliser des bibliothèques dédiées, faire du *reverse engineering* ?

– *les livrables* sont-ils bien identifiés et associés à des étapes précises ?

– *la vérification et la validation* sont-elles faciles et rapides, voire automatisables ?

– *la gestion de la qualité* est-elle prise en compte ?

– *les directives de conduite de projet* sont-elles claires ?

2.3.4. *La pragmatique*

[DAM 03, SHE 01] proposent de s'intéresser à l'aspect pragmatique de la méthode, c'est-à-dire la gestion et les questions techniques. Ceci inclut les supports mis en place pour l'utilisation de la méthode (livres, textes en ligne, ou logiciels d'aide au suivi), le prix à payer pour choisir une nouvelle méthode, ou le besoin d'expertise. Le critère technique porte plutôt sur la spécialisation des méthodes à des domaines particuliers (gestion de collectifs humains, données distribuées, résolution de problèmes, simulations, etc.).

2.4. Principales méthodes existantes

Dans cette section, nous allons présenter les principales méthodes orientées agent ordonnées alphabétiquement. Chaque méthode sera décrite brièvement puis nous examinerons les quatre axes de comparaison. Ces méthodes, pour la plupart, sont disponibles en ligne, les outils associés étant le plus souvent libres.

2.4.1. *ADELFE*

ADELFE (*atelier de développement de logiciels à fonctionnalité émergente*) est dédiée à des systèmes multi-agents très spécifiques : auto-organisateurs par coopération [PIC 04]. Ceci implique que certains concepts ne soient pas abordés explicitement, ou bien que leur manipulation ne soit pas immédiate. Par exemple, les concepts

de groupes ou d'organisations ne sont pas utilisés en tant que concepts de modélisation, mais plutôt comme des observations, donc utilisés *a posteriori*.

Les concepts manipulés par ADELFE sont directement issus des AMAS (*adaptive multi-agent systems*) [GEO 03a]. En effet, l'*adaptation* du système est obtenue par *interactions* coopératives entre les agents *autonomes* ayant des *buts* locaux. Ces derniers sont intégrés dans le module de compétence des agents. De même, les *croyances* peuvent être vues comme des représentations – qui sont forcément locales et relatives. L'autonomie d'un agent est assurée par l'encapsulation des compétences et des représentations dans les modules de l'agent, ainsi que par l'accès privé aux décisions de l'agent qui garantit la proactivité. La réactivité des agents est obtenue par l'utilisation de machines à états finis pour transcrire les comportements dynamiques et sociaux. Les interactions sont en effet prises en compte dans le module d'interaction – *via* les actions et les perceptions. L'adaptation d'un système auto-organisateur peut aussi passer par l'ajout et la suppression d'agents, et donc par l'*ouverture* du système. Enfin, l'*environnement* est très clairement cité comme concept majeur. C'est grâce à la pression qu'il exerce sur le système, que les agents vont s'organiser et fournir la bonne fonction.

ADELFE utilise UML (*unified modeling language*) et AUML (*agent UML*). Les agents sont vus comme des agents stéréotypés possédant des attributs et opérations stéréotypés en fonction des modules. L'utilisation de ces stéréotypes est réglée et en limite la mauvaise utilisation (en fonction de l'outil de conception utilisé, bien sûr). UML et AUML ont les désavantages suivants, dont hérite ADELFE : aucune gestion de la cohérence ou de la vérification d'erreurs n'est possible, et ces notations manquent de précision lors de la modélisation et d'expressivité dans la représentation de raisonnements logiques. La prise en compte de la complexité ne passe que par la structuration en paquetages, modèles et sous-systèmes, mais n'est pas aussi efficace que dans Gaia ou MaSE.

ADELFE s'appuie sur le RUP (*rational unified process*) [JAC 99]. Cependant, ADELFE n'intègre la notion d'agent que dans les travaux d'analyse et de conception. Par contre, la gestion des tests, *via* des scénarios issus des protocoles de communication, et donc simulables, est envisageable lors du prototypage rapide. L'analyse de ces tests est faite à partir des diagrammes de séquences générés en cours de simulation. Le processus d'ADELFE a été défini de manière claire, en termes d'activités, d'étapes, de participants et de documents, mais la gestion de la qualité est identique à celle proposée par le RUP et ne tient pas compte de la problématique agent. La gestion de projet et de suivi du processus est facilitée par l'utilisation d'AdelfeToolkit qui guide le concepteur tout au long du processus de développement. Enfin, comme le RUP, le processus d'ADELFE est adéquat pour n'importe quel *contexte* de développement : création de logiciels, reverse engineering, prototypage ou réutilisation.

Ce processus se divise en cinq définitions de travaux : les besoins préliminaires, les besoins finaux, l'analyse, la conception et le codage. Chaque ensemble de travaux ainsi que ses artefacts sont définis en utilisant la notation issue de SPEM (*software process engineering metamodel*) permettant de définir des processus de manière simple et compréhensible pour les utilisateurs d'UML. Certaines activités ont été ajoutées au RUP pour l'adapter à cette technologie :

1) durant l'expression des besoins finaux :

- *caractérisation de l'environnement* à partir des cas d'utilisation et des diagramme d'activités,
- *identification des échecs à la coopération* qui seront les déclencheurs de la réorganisation du système ;

2) durant l'analyse :

- *vérification de l'adéquation aux AMAS* afin de souligner la nécessité ou non d'une telle approche,
- *identification des agents impliqués dans le système à construire* à partir du domaine,
- *étude des relations entre ces agents* par protocoles AUML ;

3) durant la conception :

- *étude des langages d'interaction* par diagrammes de protocoles ou de machines à états,
- *conception complète de ces agents* par la définition des compétences, des aptitudes, un langage d'interaction, des représentations du monde et règles d'auto-organisation coopérative,
- *prototypage rapide* à partir de machines à états et de pseudo-code.

La description complète des activités s'arrête actuellement à la fin de la conception. Bien que les développements actuels se portent sur les phases préliminaires, des ponts vers l'implémentation reposant sur la transformation de modèles ont été développés : des protocoles en machines à états puis en code Java, et des classes de conception en code Java [OTT 06]. Des ressources diverses ont été mises à la disposition des utilisateurs *via* un site web [8] décrivant le processus et un exemple d'application à la conception d'un logiciel de gestion d'emploi du temps adaptatif, et permettant l'accès aux logiciels d'ADELFE. C'est une méthode générique dans le sens où aucun domaine d'application n'est réellement privilégié. Cependant, il est possible que les AMAS ne soient pas nécessaires au développement d'une application donnée, vus les fondements systémiques de la méthode. Dans de telles circonstances, l'utilisateur est invité à se tourner vers une autre méthode de développement, plus adaptée. Cependant, les retours restent mitigés lorsque des utilisateurs, concepteurs d'ADELFE exclus, tentent d'appliquer ADELFE. Certes, le processus est bien compris, mais le manque de clarté des phénomènes auto-organisateurs à mettre en place reste un problème majeur.

8. http://www.irit.fr/ADELFE.

2.4.2. *Gaia*

Gaia est une extension des approches d'ingénierie logicielle classique [WOO 00]. C'est une méthode de la seconde génération, par opposition à la première génération que formaient AAII ou DESIRE. Elle est donc plus complète et bénéficie, de plus, d'une large reconnaissance dans le domaine multi-agent. Gaia se veut être générale et applicable à n'importe quel domaine, et compréhensible par la distinction entre macro-niveau et micro-niveau. Les agents modélisés, pouvant être hétérogènes, sont des systèmes computationnels à gros grain, qui vont essayer de maximiser une mesure de qualité globale. Toutefois, Gaia ne prend pas en compte les systèmes admettant de réels conflits. L'organisation (d'une centaine d'agents environ) est clairement statique dans le temps, de même que les services offerts par les agents.

Gaia manipule six modèles d'analyse et de conception différents :

– le *modèle de rôle*, qui identifie les différents rôles devant être joués par les agents du système ;

– le *modèle d'interaction* qui définit les protocoles de communication entre agents ;

– le *modèle d'agent* qui attribue les rôles aux différents agents du système ;

– le *modèle de service* qui définit, comme son nom l'indique, les différents *services* offerts par le système, et les agents tributaires ;

– le *modèle d'organisation* qui définit la structure de l'*organisation* grâce à des graphes orientés représentant les voies de communication entre agents ;

– le *modèle environnemental* qui décrit les différentes ressources accessibles caractérisées par les types d'actions que les agents peuvent entreprendre pour les modifier.

Gaia rend facile la manipulation d'un grand nombre de concepts multi-agents, grâce à ces modèles, ce qui a certainement fait sa popularité. La propriété d'autonomie de contrôle ou de rôle des agents est exprimée par le fait qu'un rôle encapsule sa fonctionnalité, qui est interne et non affectée par l'environnement. La réactivité et la proactivité sont, elles, plus ou moins exprimées grâce aux propriétés de vivacité des responsabilités. On peut regretter que l'environnement ne soit défini que comme une liste de variables caractérisées en lecture ou écriture. Enfin, les notions sociales sont abordées dans le modèle organisationnel d'accointances.

Gaia ne fournit pas de notation graphique à proprement parler. En effet, le modèle de rôles et les protocoles ne sont décrits que par des tables. Ceci n'enlève en rien la précision obtenue grâce, notamment, aux propriétés de sûreté ou de vivacité. Les autres modèles sont aussi clairement accessibles. Les rôles étant clairement distincts, la modularité fait partie des avantages de cette approche. Par contre, comme aucune

présentation hiérarchique n'est disponible, la gestion de la complexité organisationnelle ou fonctionnelle n'est pas prise en compte et limite donc les systèmes développés grâce à Gaia, à des organisations simples et rigides. Récemment, l'adaptation dans Gaia a été étudiée mais uniquement par le biais de l'utilisation de bibliothèques d'organisations [CER 05].

Dans Gaia, trois phases sont principalement abordées : l'*analyse*, la *conception architecturale* et la *conception détaillée*. Lors de l'analyse, le système se voit divisé en sous-organisations, dont vont découler les modèles d'environnement, de rôle et d'interaction (préliminaires). A partir de ces modèles, l'analyse doit spécifier les règles organisationnelles (permissions, vivacité et sûreté) de dépendances entre rôles. La conception architecturale doit aboutir au raffinage des modèles de rôle et d'interaction par l'analyse des structures organisationnelles. Enfin, lors de la conception détaillée, les modèles d'agents (détermination des types et instances d'agents) et de services sont implantés. Ce processus est donc très limité et se focalise uniquement sur les premières phases de conception. Néanmoins, les livrables et différents modèles à fournir au cours du processus sont très clairement définis.

Les ressources disponibles sur Gaia sont assez limitées. De plus, Gaia nécessite de posséder une solide maîtrise de la logique temporelle. Ceci la rend plutôt difficile à adopter pour des développeurs. Gaia est limitée aux applications à agents à forte granularité, peu nombreux, et avec une organisation statique, ce qui rend le passage à l'échelle difficile. Malgré quelques travaux sur une spécification de directives de passage à la plate-forme JADE, Gaia ne propose aucune étude d'implémentation, ce qui a, bien sûr, l'avantage de laisser le développeur libre d'utiliser n'importe quel langage de programmation [MOR 02].

2.4.3. *INGENIAS*

INGENIAS [GOM 02] est la continuité du projet MESSAGE (*methodology for engineering systems of software agents*) qui étend la notation UML et s'intègre dans l'USDP (*unified software development process*) dont le RUP est une instanciation [JAC 99].

Dans INGENIAS, les concepts agents sont manipulés dans des vues (ou modèles) :

– la *vue de l'organisation* montre les entités concrètes dans le système et son environnement, et la relation de haut niveau comme l'agrégation, le pouvoir ou les accointances (qui implique la définition d'interactions) ;

– la *vue des buts/tâches* montre les buts, les tâches, les situations et les relations entre eux. Les dépendances temporelles peuvent être définies, ainsi que les dépendances d'ordre compositionnel ;

– la *vue des agents/rôles* associe les rôles aux agents, en spécifiant les événements déclenchant l'attribution de rôles aux agents et les ressources exploitées ;

– la *vue des interactions* définit, pour chaque interaction agent/rôle, l'initiateur, les collaborateurs, les « motivateurs », les informations échangées, les événements déclenchant les interactions, et les autres effets possibles comme l'attribution de nouveaux rôles par exemple ;

– la *vue du domaine* est une vue classique en analyse orientée objet qui décrit le domaine d'application du système en termes d'entités et de relations entre elles.

Comme dans Gaia, l'autonomie des agents est assurée par l'encapsulation des buts. La réactivité des agents est obtenue grâce à la spécification d'événements et d'actions dans les modèles d'agent/rôle et d'interaction. De même, la proactivité est reliée aux buts. Les concepts sociaux propres aux systèmes multi-agents sont exprimés, telles, par exemple, les notions d'organisation ou de collaboration.

INGENIAS propose une extension d'UML, non pas en utilisant les stéréotypes (comme dans AUML par exemple), mais en modifiant le métamodèle UML par profilage. INGENIAS hérite donc des avantages et des inconvénients d'UML, qui n'est pas un langage formel et qui perd donc en précision, mais gagne en modularité, exécutabilité (avec AUML et certains travaux sur le prototypage rapide), en analysabilité (grâce aux outils UML). Par contre, UML n'est pas le plus adéquat pour représenter les connaissances ou les raisonnements logiques.

Le processus d'INGENIAS consiste en la transformation et le raffinage des vues précédemment présentées tout au long du déroulement de l'USDP. Initialement, pour un développement donné, toutes les vues existent, et sont améliorées à chaque phase du processus. Tout le cycle de développement est couvert, des besoins à la maintenance, et ses étapes sont bien définies et couplées aux artefacts et livrables à produire. Cependant, INGENIAS ne fait que modifier les activités d'analyse et de conception pour prendre en compte la problématique agent, sans se soucier des éventuels besoins technologiques nécessaires lors de l'implémentation. INGENIAS est fortement dirigée vers Java et la plate-forme JADE. Des outils, comme IDK (*INGENIAS development kit*) sont disponibles[9], ce qui facilite les phases d'analyse et conception. La validation et la vérification, dans INGENIAS, reposent sur la théorie des activités, et un outil de vérification est inclu dans IDK.

2.4.4. *MaSE*

MaSE (*multiagent software engineering*) est un exemple d'approche complète de développement de systèmes multi-agents de l'analyse au déploiement, avec de nombreux modèles graphiques et une approche logique [DEL 01].

9. http://grasia.fdi.ucm.es/ingenias.

MaSE manipule neufs modèles très proches de ceux utilisés en conception orientée objet :

– la *hiérarchie de buts* est obtenue en décomposant le *but* principal du système en sous-buts ;

– les *cas d'utilisation* qui permettent notamment d'identifier les *rôles* joués par le système (et donc ses agents) ;

– les *diagrammes de séquences* sont définis pour chaque cas d'utilisation identifié ;

– les *rôles* qui sont vus comme le moyen d'atteindre les buts fixés au système ;

– les *tâches concurrentes* qui décrivent les moyens pour les rôles d'atteindre les buts par des automates à états finis (ou bien des réseaux de Petri) ;

– les *classes d'agents* qui sont créées à partir des rôles identifiés dont les architectures ne sont pas imposées (BDI ou autres) ;

– les *conversations* qui définissent les protocoles de coordination entre deux agents par deux automates à états finis ;

– l'*architecture* qui consiste en un ensemble de composants qui peuvent être des classes ou des ensembles de classes ;

– les *diagrammes de déploiement* permettent de spécifier la place des agents sur les plates-formes utilisées et les liens de discussion nécessaires entre les plates-formes.

Dans MaSE, l'autonomie des agents est, comme dans Gaia, assurée par l'encapsulation des fonctionnalités dans les rôles. De même, la proactivité est exprimée par les tâches, spécifiées par des automates, qui sont attribuées aux rôles. Par contre, la réactivité n'est pas clairement abordée. En effet, il n'y a pas de lien direct entre les événements et les actions ; mais il est possible de le définir en utilisant des machines à états. Les propriétés sociales (groupe, organisation) ne sont pas non plus clairement définies, contrairement aux normes (règles organisationnelles) ou protocoles (conversations).

Les nombreux modèles de MaSE utilisent des notations directement inspirées de la conception objet. La hiérarchie de but est simplement définie grâce à un arbre dont les nœuds sont les buts et les liens représentent les décompositions. Les autres modèles sont exprimés en UML. MaSE retire donc tous les avantages d'une telle notation graphique : accessibilité (mais avec le désavantage de la synchronisation nécessaire lors de la transformation de modèles), analysabilité, gestion de la complexité par décomposition des buts et des classes (mais pas pour les tâches et les rôles), expressivité (mis à part les flots de données et la représentation des connaissances comme dans DESIRE). Les diagrammes d'agents de MaSE permettent une certaine réutilisabilité, mais pas pour les tâches, les rôles ni les protocoles. Là où l'on pourrait s'attendre à un

défaut d'exécutabilité, dû à la filiation UML, MaSE fournit un outil, *agentTool*[10], qui permet une génération de code partielle.

Le processus de MaSE est très clair, et se décompose en deux phases :

1) l'*analyse* qui consiste à :

a) identifier les rôles,

b) identifier les cas d'utilisations et créer les diagrammes de séquences,

c) transformer les buts en ensembles de rôles : créer un modèle de rôle puis définir un modèle de tâches concurrentes ;

2) la *conception* qui a pour rôle de :

d) assigner les rôles à des classes d'agents spécifiques, et identifier les conversations,

e) construire les conversations,

f) définir les classes internes d'agents,

g) définir la structure finale du système.

Ce processus est valide pour n'importe quel contexte de développement (nouvelles applications, *re-engineering*, ou conception avec réutilisation). MaSE a le net avantage de couvrir la totalité du cycle de vie du système, malgré le manque de gestion de qualité. De nombreux articles et rapports sont dédiés à MaSE, ainsi, et grâce à agentTool, il devient facile de l'utiliser pour développer un système multi-agent. De plus, malgré la large couverture du processus, MaSE n'impose l'utilisation d'aucun langage de programmation particulier et s'applique à n'importe quel domaine. Cependant, le concepteur devra certainement maîtriser la logique temporelle afin d'exploiter au mieux le potentiel des modèles proposés. De plus, les architectures développées sont statiques, fermées et ne permettent pas de définir des sociétés à grand nombre d'agents. Toutefois, O-MaSE, une version étendue de MaSE [DEL 05], définit un métamodèle permettant aux agents d'adapter leur organisation à l'exécution.

2.4.5. *PASSI*

PASSI (*process for agent societies specification and implementation*) est une méthode pas à pas intégrant des modèles de conception et concepts provenant de l'ingénierie orientée objet et de l'intelligence artificielle en utilisant UML [COS 01].

Le métamodèle d'agent de PASSI manipule beaucoup de concepts communs aux agents et y ajoute les notions provenant des contraintes FIPA pour être adapté à FIPA-OS ou JADE [BER 03]. Ces notions sont encapsulées dans cinq modèles : le modèle des besoins, le modèle de société d'agent, le modèle d'implémentation d'agent, le modèle de codage et le modèle de déploiement. L'autonomie des agents est liée aux

10. http://macr.cis.ksu.edu/projects/agentTool/agentool.htm.

différents rôles qui leur sont attribués. La réactivité des agents est définie dans un ensemble de modèles comportementaux, ainsi que la proactivité. Les aspects sociaux sont intégrés dans le modèle de société d'agents.

PASSI réutilise fortement UML. Par exemple, le modèle des besoins est exprimé grâce à des diagrammes de cas d'utilisation, de paquetages stéréotypés, de séquences (associés aux cas d'utilisation) et d'activité. Le modèle de société d'agents utilise des diagrammes de classes et de séquences. Les modèles d'implémentation d'agent et de codage utilise des diagrammes de classes et d'activités classiques. Enfin, le modèle de déploiement utilise des diagrammes de déploiement UML. L'utilisation d'UML, comme pour ADELFE ou INGENIAS par exemple, implique le manque de formalisme (et donc de précision), et les connaissances ne peuvent être représentées qu'en utilisant des diagrammes de classes.

Le processus de PASSI est un processus pas-à-pas, mais réitérable. Il est composé de cinq phases, composées d'étapes, correspondant aux spécifications des cinq modèles présentés ci-dessus :

1) les *besoins* :
   a) *description du domaine* avec cas d'utilisation,
   b) *identification des agents* avec regroupement en paquetages,
   c) *identification des rôles* avec des diagrammes de séquences,
   d) *spécification des tâches* ;

2) la *définition de la société d'agents* :
   e) *identification des rôles*,
   f) *description de l'ontologie* avec des diagrammes de classes appelés diagrammes d'ontologies,
   g) *description des rôles* grâce à des diagrammes de classes et de paquetages,
   h) *description des protocoles* avec des diagrammes de séquences ou de protocoles AUML ;

3) l'*implémentation des agents* :
   i) *définition de la structure des agents*,
   j) *définition du comportement des agents* avec diagrammes d'activités ;

4) le *codage* :
   k) *réutilisation du code*,
   l) *complétion du code* ;

5) le *déploiement* :
   m) *configuration du déploiement*.

Des besoins au déploiement, les étapes de PASSI sont très clairement définies ainsi que les modèles à fournir tout au long du processus. PASSI est une méthode pouvant s'appliquer à n'importe quel domaine, mais est plutôt orientée agents mobiles, avec

notamment les aspects respectueux des spécifications FIPA. Le manque de formalisme, cependant, rend les phases de test peu efficaces.

PASSI est riche en ressources, et possède notamment un site[11] expliquant le processus ainsi que tous les modèles. Aucune expertise particulière n'est recommandée bien que la connaissance des spécifications FIPA soit préférable. PASSI n'est reliée à aucun langage particulier, bien que l'outil PTK (*PASSI toolkit*) soit orienté Java afin d'être compatible avec JADE et FIPA-OS. Tous les diagrammes et stéréotypes spécifiques sont intégrés dans une extension au logiciel Rational Rose [COS 02].

2.4.6. *Prometheus*

[PAD 03] ont développé Prometheus comme une méthode complète (processus, notations et outils) pour des agents BDI, bien qu'aujourd'hui elle ait évolué et se dise indépendante de toute architecture. Prometheus est le fruit d'une expérience de programmation avec la plate-forme JACK, ce qui implique certains choix de conception.

On retrouve dans Prometheus les mêmes concepts que dans Gaia ou MaSE. Les agents sont à gros grains, BDI et fortement *proactifs*. L'autonomie des agents est assurée par l'encapsulation des buts et des plans. Bien que la notion de groupe soit avancée, les notions de rôles sont inexistantes, et leur gestion non abordée. Par contre la sûreté est assurée lors de regroupements (en termes de droits d'accès). Une notion intéressante, et peu présente dans les autres méthodes, est la notion d'*incident* pouvant survenir en cours de fonctionnement. Cette notion reste cependant floue, et semble correspondre à la notion d'exception.

Prometheus définit des notations dédiées et réutilise UML et AUML. Nous pouvons distinguer sept notations spécifiques. Des *descripteurs textuels* permettent de spécifier des fonctionnalités du système, des scénarios, ou des agents à partir de champs textuels (comme les buts, les actions, les interactions possibles). Les *diagrammes de couplage de données* permettent de regrouper les fonctionnalités du système aux données produites en termes de production, de modification ou d'exploitation. Les *diagrammes d'accointances d'agent* décrivent les interconnexions entre agents de différents types. C'est un diagramme de classes UML simplifié où les agents sont représentés par des classes (sans compartiment) et les connexions sont représentées par des associations binaires avec multiplicité. Les *diagrammes de vue du système* relient les agents, les événements et les objets partagés dans un même modèle en utilisant une notation dédiée. Les *diagrammes d'interaction* montrent les interactions entre agents. Prometheus exploite les diagrammes de séquences UML et AUML pour modéliser les protocoles. Les *diagrammes de vue des agents* sont similaires aux diagrammes de vue du système mais montrent les interactions entre capacités au sein

11. http://mozart.csai.unipa.it/passi/.

d'un agent. Chaque message, perception ou action identifié au niveau du système doit être pris en compte à ce niveau. Les *diagrammes de vue des capacités* montrent les interactions et contraintes sur les actions et événements entre les plans et les capacités. Ces notations sont claires, sémantiquement bien définies, et donc faciles à utiliser et à apprendre. Basées sur le paradigme objet, la modularité est prise en compte et la gestion de la complexité est rendue possible par la décomposition des agents en activités et des activités en sous-activités. Comme Prometheus est couplée à JACK, le langage Java est préférable pour l'implémentation.

Le processus de Prometheus, ainsi que les modèles produits, se décompose en trois phases majeures :

1) la *spécification du système* voit l'identification des buts et sous-buts du système (sous forme de listes) par le développement des scénarios de cas d'utilisation. A ce niveau, il est alors possible d'identifier les interfaces avec l'environnement (actions et perceptions). Les fonctionnalités du système sont regroupées dans un descripteur préliminaire de capacités, ce qui nécessite, en parallèle, la définition des données de travail associées grâce à des diagrammes de couplage de données ;

2) la *conception architecturale* groupe les fonctionnalités identifiées à la phase précédente. Ces fonctionnalités permettent d'identifier des agents auxquels les affecter. Les interactions entre agents, données et perceptions sont alors spécifiées dans un diagramme de vue du système. Un diagramme de couplage est aussi utilisé pour spécifier les relations entre données et fonctionnalités regroupées. Les interactions sont aussi décrites grâce à des diagrammes de séquences ;

3) la *conception détaillée* commence par la vue interne des agents, qui est spécifique à l'architecture choisie (principalement BDI, si l'on veut bénéficier des outils proposés). Il convient alors de préciser ou de raffiner toutes les composantes internes aux agents – capacités, plans, croyances et données – dans des diagrammes de vue d'agent et de capacités.

Prometheus couvre une grande partie du cycle de développement, même si les besoins ne sont que peu abordés. Les phases de test, validation, déploiement et maintenance sont totalement absentes, ainsi que la gestion de qualité et de conduite de projet. Néanmoins, Prometheus convient à des applications orientées utilisateurs, comme Tropos, et non à la résolution de problèmes ou à la simulation. Des ressources sur Prometheus sont disponibles depuis peu. Prometheus est associée à deux outils : JDE et PDT. JDE (*JACK development environment*) permet d'éditer les modèles proposés pour les porter vers la plate-forme JACK. Seules quelques vérifications de consistance sont faites sur certains modèles. PDT[12] (*Prometheus design tool*) manipule les descripteurs et vérifie partiellement les interactions entre objets du modèle.

12. http://www.cs.rmit.edu.au/agents/pdt.

2.4.7. *Tropos*

Tropos est une méthode de développement fondée sur les concepts utilisés en ingénierie des besoins et adopte une approche transformationnelle, dans le sens où, à chaque étape, les modèles vont être raffinés de manière itérative par ajout ou suppression d'éléments ou relations dans les modèles [GIO 05].

Les concepts manipulés par Tropos sont : l'*acteur* (BDI de préférence), le *rôle*, la *position*, le *but* (*soft* ou *hard*), le *plan*, la *ressource* et la *dépendance* (avec *depender* et *dependee*). D'autres notions sont manipulées comme les *capacités* d'un acteur à définir, choisir et exécuter un plan pour remplir un but et les *croyances* qui sont une représentation de la connaissance d'un acteur sur le monde. Le but est de définir les obligations des acteurs envers les autres acteurs.

Tropos définit une modélisation selon cinq points de vue complémentaires :

– *social* : quels sont les différents acteurs et que font-ils ? Quelles sont leurs obligations et leurs capacités ?

– *intentionnel* : quels sont les buts adéquats et comment sont-ils reliés ? Comment sont-ils remplis et par qui des dépendances sont-elles demandées ?

– *communicationnel* : comment les acteurs dialoguent-ils et comment interagissent-ils ?

– *orienté procédé (processus)* : quels sont les processus (business/computer) adéquats ?

– *orienté objet* : quels sont les classes et objets adéquats et quels sont leurs liens ?

Tropos utilise principalement $i^\star$ pour modéliser les systèmes. Cette notation provient de l'ingénierie des besoins. L'utilisation de diagrammes d'états/transitions est nécessaire à la modélisation des capacités (diagrammes de capacités) et des plans (diagrammes de plan). Les interactions entre agents sont spécifiées grâce à des diagrammes de protocoles AUML. Une approche plus formelle, appelée *Formal Tropos*, propose des descriptions textuelles et logiques d'expression des besoins et des possibilités de traduction automatique vers $i^\star$ [PER 03]. Ceci ajoute de la précision aux notations et langages. Les modèles de Tropos sont simples, accessibles et très expressifs. La gestion de la complexité passe néanmoins par plusieurs modèles. Tropos a développé un outil de génération de prototypes vers la plate-forme JACK, ce qui ne réduit pas ses capacités de portabilité, bien que l'architecture agent soit imposée (agents BDI). Enfin, de nombreux outils permettent d'analyser les spécifications.

Tropos, comme INGENIAS, a l'avantage de couvrir une grande partie du cycle de développement logiciel multi-agent. Cinq phases sont clairement définies :

1) l'*analyse des besoins initiaux* qui produit un modèle de dépendances stratégiques (acteurs, buts et dépendances) et un modèle de raisonnement stratégique (moyen d'atteindre les buts collectivement) ;

2) l'*analyse des besoins finaux* fournit une description du système étudié dans son environnement opérationnel et modélise le système en tant qu'ensemble d'acteurs possédant des dépendances ;

3) la *conception architecturale* qui définit l'architecture globale du système en termes de sous-systèmes interconnectés par des flots de données et de contrôle ;

4) la *conception détaillée* qui définit chaque composant architectural en termes d'entrées, de sorties, de contrôle et qui permet de détailler la communication entre acteurs et le comportement des acteurs en différents niveaux ;

5) l'*implantation* fournit une architecture d'agents BDI sous la plate-forme JACK.

Tropos est adéquat dans n'importe quel contexte et couvre la totalité du cycle de développement, même la vérification et la validation sont abordées de manière formelle. Par contre, les activités et les livrables associés au sein des phases ne sont pas si clairement définis (la lecture de nombreux articles est nécessaire). Contrairement à INGENIAS, et au RUP en général, Tropos ne touche pas aux aspects qualité et gestion de projet. Ici encore – architecture BDI oblige – des notions de logique temporelle peuvent être nécessaires. Tropos est plutôt conçue pour des systèmes de *e*-Business ou de gestion partagée des connaissances. En effet, aucune réelle identification des agents n'est précisée. Ainsi, les acteurs de l'environnement seront toujours représentés par des agents. Des applications sans acteurs humains – du type simulation ou résolution de problèmes – sont proscrites.

2.4.8. *L'approche Voyelles*

Voyelles est une approche de haut niveau (peu de directives techniques sont données), difficile à classer en tant que méthode, mais très souvent citée car reposant sur des principes purement multi-agents [DEM 03]. Nous l'exposons ici car elle propose un cadre d'analyse très usité dans la communauté multi-agent en France, notamment. Elle repose sur la décomposition de la vue d'un système suivant cinq dimensions (ou voyelles) : Agent, Environnement, Interaction, Organisation et Utilisateur.

Cette approche permet un jeu d'écriture par associativité, afin d'exprimer le type d'approche choisie pour le développement d'un système, ou bien un domaine d'application. Par exemple, Demazeau attribue à l'informatique une approche (((A + E) + I) + O) dans laquelle les agents et l'environnement sont les priorités de développement (approche centrée agent/environnement), alors que les sciences de l'économie se caractériseraient plutôt comme (((O + I) + E) + A). Les concepts comme l'autonomie, la réactivité, la proactivité ou les aspects sociaux ne sont pas « techniquement » expliqués, mais présupposés.

Voyelles laisse les concepteurs libres d'utiliser les formalismes, les notations ou langages de leur choix pour spécifier chaque lettre du système. Cependant quelques

exemples permettent de les guider. Pour modéliser les agents, on peut par exemple utiliser des logiques formelles, des arbres, des graphes ou des automates. La modélisation de l'environnement peut elle aussi passer par des formalismes très différents comme par l'expression de différentes configurations d'arbres, telle que l'organisation. Comme dans de nombreuses autres méthodes, les protocoles exprimés par des diagrammes de séquences ou des machines à états sont un moyen d'exprimer des interactions.

Voyelles peut se coupler à un processus de développement logiciel classique dans lequel les concepts Voyelles se greffent à tout moment :

1) l'*analyse* porte sur le domaine d'application et le type du problème, et consiste en la décomposition du problème en voyelles, c'est-à-dire les différentes composantes du système ;

2) la *conception* correspond à l'identification des modèles à utiliser pour chacune des voyelles [OCC 01] :

- ASTRO pour les agents hybrides ou PACORG pour les agents réactifs,
- les langages à protocoles (IL) ou les forces de PACO pour les interactions,
- les organisations statiques de RESO ;

3) la *programmation* (ou implémentation) consiste en l'instanciation des modèles, en utilisant des plates-formes et des langages choisis.

Ce processus est complet, mais comme son niveau d'abstraction est élevé, aucune des phases n'est réellement détaillée. Par conséquent, les livrables ne sont pas décrits, car leur nature peut différer d'une application à une autre. La vérification et la validation sont prises en compte, avec par exemple l'outil AGIP de vérification de protocoles. La gestion de projet et de qualité sont absentes dans Voyelles.

Voyelles est certainement l'approche méthodologique qui se distingue le plus des autres, et leur est difficilement comparable, du moins à partir des critères choisis. Ici, la priorité est la liberté totale de choix – ce qui est toutefois problématique lorsque l'on veut enseigner les moyens de concevoir des systèmes, ce qui est le but, au final, d'une méthode de développement. Le niveau de détail est trop faible pour l'utiliser sans connaissance approfondie de l'approche. De plus, les ressources sont quasi inexistantes. Le seul moyen de savoir quels modèles utiliser, comment décomposer son problème, ou quels outils choisir est de parcourir les travaux (la plupart étant des thèses) ayant utilisé la méthode. La plate-forme MASK (*Multi-agent system kernel*) est développée pour fournir des outils de développement Voyelles et permet l'ajout de modèles. Elle fonctionne sous UNIX et les librairies pour les boîtes à outils sont écrites en C, C++ et Java. Les agents sont distribués en utilisant les communications UNIX (TCP/IP), le système Xenoops ou le WEB. Cependant elle reste à l'état de prototype et son portage reste difficile. Enfin, le développement et le déploiement des systèmes peuvent être assurés par la plate-forme Volcano [RIC 01].

2.5. Comparaison des méthodes présentées

Le tableau 2.1 montre une synthèse de la comparaison des méthodes présentées dans la section 2.3. Seule l'approche Voyelles n'est pas représentée dans ce comparatif, pour les raisons citées plus haut.

2.5.1. *Lecture suivant l'axe des méthodes*

Lire le tableau suivant l'axe des méthodes (de gauche à droite) nous indique si une notion méthodologique donnée est largement prise en compte par les méthodes orientées agent existantes. On peut constater que les interactions ou les notions de rôles sont communément abordées. Par contre, des notions comme l'adaptation, l'ouverture ou l'environnement sont très souvent absentes. L'autonomie reste à « une moyenne de + » seulement, compte tenu de la partialité de cette propriété, qui ne repose souvent que sur des présupposés ou des encapsulations (artificielles) de capacités cognitives.

Concernant les notations, cette lecture reste difficile car aucune notation ne semble universelle. Seules les méthodes proposant plusieurs types de notations ou langages, sont polyvalentes, comme MaSE ou Tropos. Deux courants principaux se dégagent : les notations de type UML qui gagnent en modularité, analysabilité et complexité, et les notations et langages formels plus précis et vérifiables.

Toutes les méthodes affichées se concentrent principalement sur les phases d'analyse et de conception. Tropos est un cas d'exception concernant les besoins ; ceci s'explique par sa filiation à l'ingénierie des besoins avec $i^\star$. Les autres exceptions sont des méthodes dont le processus s'inspire de processus orientés objet éprouvés, comme le RUP avec ADELFE ou l'USDP avec INGENIAS. Cette limitation à ces deux phases peut s'expliquer de deux façons :

– la technologie agent n'en est qu'à ses débuts et certains concepts, comme l'autonomie ou la socialité, restent difficiles à implanter ;

– le concept d'agent n'a peut-être aucune spécificité par rapport à l'objet en ce qui concerne l'implantation.

La première hypothèse est assez convaincante. En effet, ceci implique aussi le manque de formalisme et donc de validation. Peu de définitions communément admises existent. Même le concept d'agent est sujet à polémique. La seconde hypothèse semble par contre discutable. En effet, implanter des agents comme des objets nécessite de se munir de règles de transformation spécifiques. De plus, le domaine de la programmation orientée agent, qui propose de nouveaux paradigmes de programmation, non forcément inspirés de l'objet, souligne la nécessité de se munir de nouveaux langages de programmation afin de palier les difficultés des concepts agents et multi-agents.

	ADELFE	Gaia	MaSE	INGENIAS	PASSI	Prometheus	Tropos	Voyelles
Adaptation	++	±	+	±	±	-	-	+
Autonomie	+	+	+	+	+	+	+	±
Buts	+	±	+ +	+ +	±	+ +	+ +	±
Croyances	+	±	+	-	+	+ +	+ +	+
Désirs	-	±	+ +	+	+ +	+ +	+	+
Environnement	+	±	-	-	±	+	±	+
Groupe	±	±	-	±	+	+	±	+
Intention	-	±	±	±	±	+ +	+ +	+
Interaction/Message	++	+	+ +	+ +	+ +	+ +	+	+ +
Norme	±	+	+ +	±	-	±	+	+
Organisation	+	+ +	±	+ +	+	+	+	+ +
Ouverture	++	- -	±	±	±	-	- -	+
Proactivité	+	+	+	+	+	+	+	±
Réactivité	++	+	-	+	+	±	+	±
Rôle	+	+ +	+ +	+ +	+ +	-	+ +	+
Tâche	±	+	+ +	+ +	+ +	+	±	+
Accessibilité	+	±	±	+	+	+	+	?
Analysabilité	+	- -	+	+	+	+ +	+ +	?
Complexité	+	- -	±	+	+	+	-	±
Consistance	+	+	±	±	+	+ +	+ +	?
Exécution	++	- -	+ +	+	+ +	+	+ +	+
Expressivité	+	±	±	±	±	+	±	-
Modularité	++	+ +	±	+ +	+ +	+	+	?
Portabilité	±	±	±	±	+	±	-	+
Précision	-	+ +	+ +	±	±	+	+ +	- -
Raffinage	+	+	+ +	+	+	+	+ +	?
Traçabilité	+	+	-	+	+	+	±	?
Contexte	Tous	Tous	Tous	Tous	Tous	Tous	Tous	Tous
Besoins	+	-	- -	+	+	-	+ +	-
Analyse	++	+ +	+ +	+ +	+ +	+ +	+ +	+ +
Conception	++	+ +	+ +	+ +	+ +	+ +	+	+ +
Implémentation	+	- -	+	+	+	+	+	+
Test	+	-	±	+	±	-	±	+
Déploiement	±	- -	- -	±	+ +	- -	- -	+
Maintenance	±	- -	- -	±	±	- -	- -	- -
Délivrables	++	+ +	+ +	+ +	+ +	+	-	- -
Gestion de la qualité	+	- -	- -	+	- -	- -	- -	- -
Gestion de projet	++	- -	- -	+	- -	- -	- -	- -
Ressources	++	±	+	+ +	+ +	+	±	-
Expertise requise	Plutôt Oui	Oui	Non	Non	Non	Oui	Oui	Non
Langage spécifique	Non	Non	Non	Non	Non	Non (Java)	Non	Non
Domaine d'application	Non	Gros Grains	Non	Non	FIPA	Utilisateur	Utilisateur	Non
Scalabilité	Oui	Oui	Non	Oui	Oui	Oui	Non	Oui

Légende : (++) pour les propriétés pleinement et explicitement prises en charge ; (+) pour les propriétés prises en charge de manière indirecte ; (±) pour des propriétés potentiellement prises en charge ; (−) pour des propriétés non prises en charge ; (−−) pour des propriétés explicitement non prises en charge.

Tableau 2.1. *Synthèse de la comparaison des différentes méthodes*

Enfin, les méthodes orientées agent manquent cruellement de ressources. Rares sont celles proposant des tutoriels, des pages internets ou des livres synthétiques.

2.5.2. *Lecture suivant l'axe des notions méthodologiques orientées agent*

Lire le tableau 2.1 suivant l'axe des notions méthodologiques orientées agent/multi-agent nous indique si une méthode donnée aborde tous les concepts et propriétés, possède des notations et langages efficaces, propose un cycle de développement complet et clair, et enfin si cette méthode est facilement exploitable. Globalement, un premier constat est qu'aucune méthode n'est « parfaite » dans ce sens. Chaque méthode possède ses spécificités en termes d'architecture, de formalisme ou de modèles. Ces différences sont souvent dues aux filiations des méthodes – ingénierie des connaissances, des besoins, orientée objet, ou simulation.

Toutefois, certaines méthodes sont plus générales – ou inversement plus spécialisées – que d'autres. Par exemple, Tropos sera plutôt utilisée pour des systèmes orientés ergonomie et utilisateurs ou systèmes d'information, compte tenu de la richesse de son analyse des besoins. Mais ceci implique aussi une architecture BDI d'agents à gros grain. A l'opposé, INGENIAS, s'intégrant dans le USDP, est assez générale et convient à tout type de contexte. PASSI, bien qu'ayant fait le choix de la FIPA, propose un processus et des notations riches et expressives.

2.6. Les standardisations

Les organismes de standardisation de technologies concernées par les agents et multi-agents sont : l'OMG (*objet management group*) qui comprend un groupe spécialisé sur les agents et qui a notamment créé un standard pour les agents mobiles, MASIF ou MAF (*mobile agent system interoperability facility*) [GRO 00], le GGF[13] (Global Grid Forum) qui s'intéresse aux standards associés au calcul sur la grille, le W3C[14] (World Wide Web Consortium) qui développe des standards et des guides pour le web, OASIS[15] (Organization for the Advancement of Structured Information Standards) qui standardise le format de fichiers ouverts basés sur XML et la FIPA qui définit des standards pour l'interopérabilité et l'interaction entre agents.

Depuis 1996, la FIPA est sans conteste l'organisme moteur principal pour la standardisation de la technologie agent et multi-agent. Ses objectifs sont de promouvoir l'utilisation de ces techniques par des industriels notamment, et d'assurer l'interopérabilité entre les agents, entre les systèmes multi-agents et entre les systèmes à agents et

13. http://www.gridforum.org.

14. http://www.w3c.org.

15. http://www.oasis-open.org.

les autres technologies. Les principaux standards produits sont accessibles en ligne[16]. Ils concernent cinq thèmes à savoir : la communication entre agents, le transport des messages entre agents, la gestion des agents, l'architecture abstraite d'un agent et des applications. La majorité des travaux réalisés sont centrés sur la communication entre agents et ont fourni les standards concernant l'échange de messages avec les spécifications des messages ACL et la définition de protocoles. Les principaux développements issus de ces spécifications sont en accès libre et concernent des plates-formes comme ZEUS [NWA 99], JADE et l'API JAS[17] (*Java agent services*). Les comités techniques ayant pour rôle l'écriture de spécifications se sont ensuite intéressés à quatre axes :

– les spécifications concernant *l'interopérabilité*. Elles permettent à des plates-formes agents conformes aux normes FIPA ou à des fragments de plates-formes d'interopérer ;

– les spécifications concernant *la sécurité dans les systèmes multi-agents*. Les travaux ont pris en compte plusieurs niveaux de sécurité dans les systèmes comme au niveau des échanges de messages, de l'attaque du système par un agent mal intentionné, etc., et ont révisé le modèle de sécurité FIPA pour développer un service de sécurité basé sur les agents ;

– la *définition d'un cadre sémantique*. L'étude des échanges entre les agents pour étendre le cadre sémantique actuel avec des notions telles que le contrat, la confiance, etc. a été réalisée. Un nouveau cadre sémantique a été établi pour assurer et vérifier la conformité avec les spécifications FIPA ;

– l'*ingénierie de systèmes orientée agent*, avec des travaux sur la réalisation de méthodes de conception de systèmes multi-agents et sur la modélisation de ces systèmes. Le groupe sur la modélisation a travaillé sur la réalisation d'un métamodèle de l'agent conforme aux normes FIPA et sur AUML, l'extension d'UML pour la prise en compte des protocoles d'interaction et plus généralement des spécificités des agents. Le groupe sur les méthodologies s'est orienté vers le découpage des méthodes existantes en fragments, permettant au concepteur de définir la méthode la plus adaptée à son application.

Depuis juin 2005, la FIPA est officiellement le onzième comité des standards de l'IEEE (Institute of Electrical and Electronics Engineers) Computer Society[18]. Les problématiques prises en compte actuellement sont de définir des standards :

– pour l'interopérabilité entre les services web et les agents. L'objectif est de combler le manque d'interaction entre les agents et les services web pour que les agents puissent utiliser les services web et *vice versa* ;

16. http://www.fipa.org/specifications/.

17. http://www.sourceforge.net/projects/jas.

18. http://www.computer.org.

– pour les agents mobiles. Le but est d'améliorer et d'étendre les spécifications existantes et de développer des implémentations de protocoles sous la forme de composants logiciels pour des boîtes à outils destinées aux agents ;

– pour les communications avec l'être humain. L'objectif est d'étendre le langage ACL à la communication avec des humains et de produire des standards IEEE ;

– pour les applications P2P et les agents nomades. Le but est de définir des spécifications pour des agents nomades, capables de s'exécuter sur différents types de périphériques au sein d'un réseau pair à pair.

2.7. Prospective pour l'ingénierie des logiciels orientés agent

2.7.1. *La période actuelle 2007-2008*

Un frein à l'utilisation des systèmes basés sur les agents dans l'industrie réside dans le manque de standards dans le domaine malgré les efforts déjà fournis comme nous venons de le souligner dans la section précédente. Même si certaines méthodes orientées agent reposent sur des standards de fait tels que UML ou le RUP, leur diffusion dans le monde industriel est limitée ; ceci tient essentiellement au fait que les concepts agent et multi-agent doivent encore être normalisés, qu'aucune méthode n'émerge parmi celles existantes et qu'aucune n'est vraiment universelle. La réponse du groupe de travail « Methodology » de la FIPA ou du TFG AOSE d'AgentLink III était de clarifier les notions agent et multi-agent manipulées dans les méthodes. Cette clarification passant par l'établissement d'un métamodèle des systèmes multi-agents unificateur des méthodes [BER 03, ODE 04a] permettant une composition de méthodes. En définissant et décrivant des fragments des méthodes existantes, à l'aide d'une notation telle que le SPEM par exemple, un ingénieur a la capacité de les assembler au gré de ses besoins pour constituer une méthode adaptée à son domaine d'applications. L'idée du peuplement d'une base de données de fragments de méthodes a été repris dans divers travaux [COS 04, ELA 06, GAR 04, HEN 05] et constitue une perspective de recherche à court terme dans le domaine de l'ingénierie orientée agent.

Actuellement, les premières phases du cycle de vie du logiciel sont abordées par toutes les méthodes, par contre, peu traitent les phases de développement et de déploiement. Il semble donc que des efforts doivent être réalisés sur ces phases-là avec la définition d'outils facilitant, d'une part, le codage et les tests et, d'autre part, la mise en place de ces systèmes dans des environnements réels avec des problèmes de passage à l'échelle. Ainsi, il est nécessaire de disposer de langages de programmation, de plates-formes et d'environnements de développement [BOR 06].

L'offre en termes de plates-formes est assez conséquente [DAU 06] pour que les concepteurs trouvent une plate-forme adaptée à l'application à développer. Les plates-formes cibles des applications peuvent donc être diverses, ce qui explique que suivre

l'approche MDA (*model driven architecture*) de l'OMG constitue un axe de recherche de plus en plus suivi en ingénierie logicielle orientée agent [COS 05, GIO 05, OTT 06, ROU 07, SAN 05]. Ceci permettrait de déployer un système multi-agent sur différentes plates-formes. Intégrer la transformation de modèles dans les méthodes est le but de travaux tels ceux de [GER 02, KAS 05, THI 03] et doit être poursuivi. D'un autre côté, le « living design » proposé par [GEO 03b] et dont l'idée est reprise dans PASSI est d'utiliser la simulation pour aider au développement des systèmes multi-agents et plus particulièrement à la conception du comportement de l'agent [BER 07, LEM 07]. Toujours pour faciliter le rôle des concepteurs, des outils d'observation, de débogage et de validation de ces systèmes complexes sont à concevoir, et ce, en lien avec les travaux déjà réalisés en simulation multi-agent. De même, ces outils devraient être intégrés dans un environnement de développement.

A court terme, les méthodes de développement, les langages et les outils devraient avoir atteint un degré de maturité et les systèmes développés devraient suivre des standards.

2.7.2. *A moyen terme : vers 2018*

La grande majorité des systèmes multi-agents actuels sont composés d'agents homogènes (c'est-à-dire ayant une architecture identique) et sont développés par le même concepteur. La communauté scientifique d'AgentLink III [LUC 05] prévoit que de plus en plus, ces systèmes seront conçus par des développeurs différents et les agents seront hétérogènes. Cela signifie que soit des systèmes existants auront à interagir, soit les systèmes seront ouverts en acceptant la création ou la suppression d'agents. La conception classique d'un système de bout en bout par une seule équipe de développement va donc rapidement devenir inadéquate et il faudra trouver de nouvelles méthodes de développement mettant en avant la modularité et la réutilisabilité et permettant de prendre en compte l'interopérabilité, l'ouverture, la dynamique, la robustesse et l'adaptation au sein de ces systèmes. Il est évident que les standards auront un rôle essentiel pour prendre en compte cette hétérogénéité et permettre aux agents d'interagir. Les méthodes devront participer à la validation de ces systèmes comportant un grand nombre d'agents. Ce passage à l'échelle représente un réel défi. Il faudra aussi fournir de nouveaux moyens de contrôler ces systèmes dont le concepteur ne connaîtra pas, *a priori*, l'environnement d'exécution au moment de la conception.

A moyen terme, les méthodes de conception orientées agents devraient donc favoriser la conception de systèmes multi-agents interopérables, adaptatifs et à grande échelle.

2.7.3. *A long terme : au-delà de 10 ans*

Devant la complexité croissante des systèmes à concevoir et des environnements dynamiques dans lesquels ils auront à évoluer, les agents, ainsi que les systèmes seront dotés de capacités d'adaptation et d'apprentissage. De plus, les facilités d'interaction entre des entités hétérogènes devraient permettre à ces systèmes de s'autoconstruire, et de s'automaintenir. Cette autoconstruction peut être réalisée à plusieurs niveaux : au niveau du programme en ayant une conception automatique du programme par auto-organisation d'instructions comme dans les travaux de [GEO 05], au niveau de l'agent en faisant appel à de nouveaux composants (voir le chapitre 5 « Composants logiciels et systèmes multi-agents ») pour augmenter ou changer ses compétences, au niveau du système en créant ou supprimant des agents pour répondre à un nouveau besoin : réparation, nouvelle fonction à réaliser, etc.

A long terme, la méthode de conception pourrait être vue comme un ensemble d'outils intégrés dans un logiciel minimal lui permettant de s'autoconstruire par couplage avec son environnement.

2.8. Conclusion

Comme ce chapitre l'a présenté, de nombreuses méthodes orientées agent existent actuellement, chacune rendant compte de la vision de ses concepteurs, sur les concepts agents ou multi-agents. Dans un souci d'unification, à court terme, les travaux sur le développement de systèmes multi-agents se focalisent sur trois grands axes : le développement de standards, d'outils d'aide à la conception et sur le développement des dernières phases du cycle de développement de logiciels à savoir les phases d'implémentation, validation, de déploiement et de maintenance. A moyen terme, un système multi-agent ne sera pas forcément conçu par le même concepteur et les agents d'un même système ne seront plus homogènes. Les méthodes de conception devront donc évoluer vers la prise en compte de cette hétérogénéité. La manière de concevoir, d'implémenter, de déployer et de maintenir ces systèmes va changer par rapport à la conception actuelle de ces logiciels. En effet, pour des domaines d'applications comme les services web, les systèmes ambiants, les concepteurs n'auront plus à concevoir les systèmes à partir de rien et dans leur globalité mais ils devront interagir et intégrer un fragment de logiciel (agent) dans un système existant. A long terme, l'autonomie et les capacités d'interaction du système avec son environnement permettent d'imaginer des systèmes qui s'autoconstruisent ce qui entraînera une révolution des méthodes de développement.

Pour assurer ces futurs à l'ingénierie logicielle orientée agent, il est, bien entendu, important que les chercheurs fassent des propositions mais il est aussi fondamental que ces actions soient portées par des industriels afin que la technologie agent trouve une place analogue à celle de l'objet dans le monde industriel.

2.9. Bibliographie

[ARL 04] **Arlabosse F.**, **Gleizes M.-P.**, **Occello M.**, « Méthodes de Conception », *Systèmes Multi-Agents*, vol. 29 de *Arago*, Editions Tec & Doc, p. 137-171, 2004.

[BER 02] **Bernon C.**, **Gleizes M.-P.**, **Picard G.**, **Glize P.**, « The ADELFE Methodology For an Intranet System Design », **Giorgini P.**, **Lespérance Y.**, **Wagner G.**, **Yu E.** (dir.), *Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002)*, vol. 57, Toronto, Canada, CAiSE'02, CEUR Workshop Proceedings, mai 2002.

[BER 03] **Bernon C.**, **Cossentino M.**, **Gleizes M.-P.**, **Turci P.**, **Zambonelli F.**, « A Study of Some Multi-Agent Meta-Models », **Giorgini P.**, **Müller J. P.**, **Odell J.** (dir.), *Agent-Oriented Software Engineering IV, 5th International Workshop, AOSE 2004*, New York, 19 juillet 2003, p. 113-130, 2003.

[BER 04] **Bergenti F.**, **Gleizes M.-P.**, **Zambonelli F.** (dir.), *Methodologies and Software Engineering for Agent Systems*, Kluwer Publishing, 2004.

[BER 07] **Bernon C.**, **Gleizes M.-P.**, **Picard G.**, « Enhancing Self-Organising Emergent Systems Design with Simulation », *Seventh International Workshop on Engineering Societies in the Agents World (ESAW'06), Dublin, Ireland from the 6th-8th September, 2006*, Springer, p. 284-299, 2007.

[BOI 04] **Boissier O.**, **Gitton S.**, **Glize P.**, « Caractéristiques des systèmes et des applications », *Systèmes Multi-Agents*, vol. 29 de *Arago*, Editions Tec & Doc, p. 25-54, 2004.

[BOO 92] **Booch G.**, *Conception orientée objets et applications*, Addison-Wesley, 1992.

[BOR 06] **Bordini R.**, **Braubach L.**, **Dastani M.**, **El Fallah Seghrouchni A.**, **Gomez-Sanz J.**, **Leite J.**, **O'Hare G.**, **Pokahr A.**, **Ricci A.**, « A Survey of Programming Languages and Platforms for Multi-Agent Systems », *Informatica, An International Journal of Computing and Informatics*, vol. 30, n° 1, p. 33-44, 2006.

[BRA 97] **Brazier F.**, **Dunin-Keplicz B.**, **Jennings N.**, **Treur J.**, « DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework », *International Journal of Cooperative Information Systems*, vol. 6, n° 1, p. 64-94, World Scientific, 1997.

[BUS 98] **Bussman S.**, Position Paper on Agent-Based Software Engineering, Contribution to Methodologies/software engineering SIG, Rapport, First SIG Meeting – AgentLink, 1998.

[CAS 01] **Castro J.**, **Kolp M.**, **Mylopoulos J.**, « A Requirements-driven Development Methodology », **Dittrich K.**, **Geppert A.**, **Norrie M.** (dir.), *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, vol. 2068 de *LNCS*, Springer, p. 108-123, 2001.

[CER 02] **Cernuzzi L.**, **Rossi G.**, « On The Evaluation Of Agent Oriented Methodologies », **Debenham J.**, **Henderson-Sellers B.**, **Jennings N.**, **Odell J.** (dir.), *Proceedings of the OOPSLA 02 - Workshop on Agent-Oriented Methodologies*, COTAR, p. 21-30, 2002.

[CER 04a] **Cernuzzi L.**, **Juan T.**, **Sterling L.**, **Zambonelli F.**, « The Gaia Methodology: Basic Concepts and Extensions », **Bergenti F.**, **Gleizes M.-P.**, **Zambonelli F.** (dir.), *Methodologies and Software Engineering for Agent Systems*, Kluwer Publishing, p. 69-88, 2004.

[CER 04b] **Cervenka R.**, **Trencanský I.**, **Calisti M.**, **Greenwood D. A. P.**, « AML: Agent Modeling Language Toward Industry-Grade Agent-Based Modeling », *Agent Oriented Software Engineering*, p. 31-46, 2004.

[CER 05] **Cernuzzi L.**, **Zambonelli F.**, « Dealing with Adaptive Multiagent Systems Organizations in the Gaia Methodology », *6th International Workshop on Agent-Oriented Software Engineering (AOSE-05) at AAMAS'05, July 25-26, 2005,* Utrecht, Pays-Bas, Springer, 2005.

[COL 96] **Collinot A.**, **Drogoul A.**, « Agent Oriented Design of a Soccer Robot Team », **Tokoro M.** Ed., *Second International Conference on Multi-Agent Systems (ICMAS'96), Nara, Japan*, AAAI Press, p. 41-47, 1996.

[COS 01] **Cossentino M.**, « Different Perspectives in Designing Multi-Agent System », *Designing Multi-Agent System, AgeS'02 (Agent Technology and Software Engineering) Workshop at NodE'02*, 2001.

[COS 02] **Cossentino M.**, **Potts C.**, « A CASE tool supported methodology for the design of multi-agent systems », *The Proceedings of the International Conference on Software Engineering Research and Practice (SERP'02)*, 2002.

[COS 04] **Cossentino M.**, **Seidita V.**, « Composition of a New Process to Meet Agile Needs Using Method Engineering », *Software Engineering for Multi-Agent Systems III, Research Issues and Practical Applications*, p. 36-51, 2004.

[COS 05] **Cossentino M.**, « From Requirements to Code with the PASSI Methodology », **Henderson-Sellers B.**, **Giorgini P.** (dir.), *Agent Oriented Methodologies*, Idea Group, p. 79-106, 2005.

[DAM 03] **Dam K. H.**, **Winikoff M.**, « Comparing Agent-Oriented Methodologies », *Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2003), Melbourne, Australia, at AAMAS'03*, vol. 3030 de *LNCS*, Springer, p. 78-93, 2003.

[DAU 06] **Daures N.**, Étude de la simulation de systèmes multiagents pour la conception vivante d'agents dans la méthode ADELFE, Rapport de Master 2 Recherche IARCL, Université Toulouse III, 2006.

[DEL 01] **Deloach S.**, **Wood M.**, **Sparkman C.**, « Multiagent Systems Engineering », *International Journal of Software Engineering and Knowledge Engineering*, vol. 11, n° 3, p. 231-258, World Scientific, 2001.

[DEL 05] **DeLoach S. A.**, « Engineering Organization-Based Multiagent Systems », *SELMAS*, p. 109-125, 2005.

[DEM 01] **Demazeau Y.**, VOYELLES, Mémoire d'habilitation à diriger des recherches, INP Grenoble, 2001.

[DEM 03] **Demazeau Y.**, « Créativité émergente centrée utilisateur », **Briot J.**, **Khaled G.** (dir.), *Déploiement des systèmes multi-agents – Vers un passage à l'échelle (Journées francophones sur les systèmes multi-agents, JFSMA'03)*, Hermès - Lavoisier (Revue RSTI Hors-série), p. 31-36, 2003.

[ELA 06] **Elamy A.-H.**, **Far B.**, « A Statistical Approach for Evaluating and Assembling Agent-Oriented Software Engineering Methodologies », *Proceedings of AOIS'06*, p. 17-34, 2006.

[ELF 03] **El Fallah Seghrouchni A.**, **Suna A.**, « CLAIM : Un langage de programmation pour des agents autonomes, intelligents et mobiles », *Technique et Science Informatiques*, vol. 22, n° 4, p. 91-105, 2003.

[FER 98] **Ferber J.**, **Gutknecht O.**, « Aalaadin : a meta-model for the analysis and design of organizations in multi-agent systems », **Demazeau, Y.** Ed., *3rd International Conference on Multi-Agent Systems*, Paris, IEEE, p. 128-135, 1998.

[GAR 03] **Garcia A.**, **Pereira de Lucena C.**, **Zambonelli F.**, **Omicini A.**, **Castro J.** (dir.), *Software Engineering for Large-Scale Multi-Agent Systems*, vol. 2603 de *LNCS*, Springer, 2003.

[GAR 04] **Garro A.**, **Fortino G.**, **Russo W.**, « Using Method Engineering for the Construction of Agent-Oriented Methodologies », *Proceedings of WOA 04 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti razionali*, p. 51-54, 2004.

[GEO 03a] **Georgé J.-P.**, **Gleizes M.-P.**, **Glize P.**, « Conception de systèmes adaptatifs à fonctionnalité émergente : la théorie AMAS », *Revue des sciences et technologie de l'information*, vol. 17, n° 4, p. 591-626, Hermès, 2003.

[GEO 03b] **Georgé J.-P.**, **Picard G.**, **Gleizes M.-P.**, **Glize P.**, « Living Design for Open Computational Systems », **Fredriksson M.**, **Ricci A.**, **Gustavsson R.**, **Omicini A.** (dir.), *International Workshop on Theory And Practice of Open Computational Systems (TAPOCS) at 12th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03)*, IEEE Computer Society, p. 389-394, 2003.

[GEO 05] **Georgé J.-P.**, **Gleizes M.-P.**, « Experiments in Emergent Programming Using Self-organizing Multi-Agent Systems », **Pechoucek M.**, **Petta P.**, **Varga L. Z.** (dir.), *4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'05)*, Budapest, Hongrie, Springer, p. 450-459, 2005.

[GER 02] **Gervais M.-P.**, « Towards an MDA-Oriented Methodology », *26th International Computer Software and Applications Conference (COMPSAC 2002), Prolonging Software Life: Development and Redevelopment*, p. 265-270, 2002.

[GIO 05] **Giorgini P.**, **Kolp M.**, **Mylopoulos J.**, **Castro J.**, « Tropos: A Requirements-Driven Methodology for Agent-Oriented Software », **Henderson-Sellers B.**, **Giorgini P.** (dir.), *Agent Oriented Methodologies*, Idea Group, p. 20-45, 2005.

[GOM 02] **Gomez Sanz J.**, **Fuentes R.**, « Agent Oriented System Engineering with INGENIAS », *Fourth Iberoamerican Workshop on Multi-Agent Systems, Iberagents'02*, 2002.

[GRO 00] **Group O. M.**, Mobile Agent Facility Specification v1.0, Rapport formal/2000-01-02, OMG Document, 2000.

[HEN 05] **Henderson-Sellers B.**, **Giorgini P.** (dir.), *Agent Oriented Methodologies*, Idea Group, 2005.

[IGL 99] **Iglesias C.**, **Garijo M.**, **Gonzalez J.**, « A Survey of Agent-Oriented Methodologies », *ntelligent Agents V, ATAL'98*, vol. 1555 de *LNAI*, Springer, 1999.

[JAC 99] **Jacobson I.**, **Booch G.**, **Rumbaugh J.**, *The Unified Software Development Process*, Addison-Wesley, 1999.

[JEN 99] **Jennings N. R.**, « Agent-Oriented Software Engineering », **Garijo F. J.**, **Boman M.** (dir.), *Multi-Agent System Engineering, 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'99*, Valence, Espagne, juin-juillet 1999, vol. 1647 de *LNAI*, Springer, 1999.

[KAS 05] **Kasinger H.**, **Bauer B.**, « Towards a Model-Driven Software Engineering Methodology for Organic Computing Systems », **Press I.** Ed., *Proceedings of the 4th IASTED International Conference on Computational Intelligence (CI 2005)*, p. 141-146, 2005.

[KIN 96] **Kinny D.**, **Georgeff M.**, **Rao A.**, « A Methodology and Modelling Technique for Systems of BDI agents », **Van de Velde W.**, **Perram J. W.** (dir.), *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a MultiAgent World*, vol. 1038 de *LNAI*, Springer, p. 51-71, 1996.

[LEM 07] **Lemouzy S.**, **Bernon C.**, **Gleizes M.-P.**, « Living Design: Simulation for Self-Designing Agents », *European Workshop on Multi-Agent Systems (EUMAS)*, Hammamet, Tunisie, 13/12/2007-14/12/2007, 2007.

[LUC 05] **Luck M.**, **Mc Burney P.**, **Shehory O.**, **Willmott S.**, *Agent Technology : Computing as Interaction, A Roadmap for Agent Based Computing*, AgentLink, 2005.

[MOR 02] **Moraitis P.**, **Petraki E.**, **Spanoudakis N.**, « Engineering JADE Agents with Gaia Methodology », **Kowalczyk R.**, **Muller J.**, **Tianfield H.**, **Unland R.** (dir.), *Agent Technologies, Infrastructures, Tools and Applications for E-Services – Best (revised) papers of NODe 2002 Agent-Related Workshops*, vol. 2592 de *LNAI*, Springer, p. 77-92, 2002.

[NWA 99] **Nwana H.**, **Ndumu D.**, **Lee L.**, **Collis J.**, « ZEUS: A Tool-Kit for Building Distributed Multi-Agent Systems », *Applied Artifical Intelligence Journal*, vol. 13, n° 1, p. 129-186, 1999.

[OCC 01] **Occello M.**, **Koning J.-L.**, **Baeijs C.**, « Conception de SMA. Quelques éléments de réflexion méthodologique », *Revue technique et science informatiques*, vol. 20, n° 2, Hermès, 2001.

[ODE 04a] **Odell J.**, **Giorgini P.**, **Müller J. P.** (dir.), *Agent-Oriented Software Engineering V, 5th International Workshop, AOSE 2004*, New York, 19 juillet 2004, Revised Selected Papers, vol. 3382 de *LNCS*, Springer, 2004.

[ODE 04b] **Odell J.**, **Nodine M. H.**, **Levy R.**, « A Metamodel for Agents, Roles, and Groups », *Agent Oriented Software Engineering*, p. 78-92, 2004.

[OTT 06] **Ottens K.**, **Picard G.**, **Camps V.**, « Transformation de modèles d'agents dans la méthode ADELFE : Des stéréotypes de conception à l'implémentation », *Revue technique et science informatique – L'objet*, vol. 12, n° 4, p. 43-72, Hermès, 2006.

[PAD 03] **Padgham L.**, **Winikoff M.**, « Prometheus: A Methodology for Developing Intelligent Agents », **Giunchiglia F.**, **Odell J.**, **Weiß G.** (dir.), *Agent-Oriented Software Engineering III, Third International Workshop, AOSE 2002*, Bologne, Italie, 15 juillet 2002, Revised Papers and Invited Contributions, vol. 2585 de *LNCS*, Springer, p. 174-185, 2003.

[PER 03] **Perini A.**, **Pistore M.**, **Roveri M.**, **Susi A.**, « Agent-oriented modeling by interleaving formal and informal specification », **Giorgini P.**, **Müller J. P.**, **Odell J.** (dir.), *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003*, Melbourne, Australie, 15 juillet 2003, Revised Papers, vol. 2935 de *LNCS*, Springer, p. 36-52, 2003.

[PIC 04] **Picard G.**, **Gleizes M.-P.**, « The ADELFE Methodology – Designing Adaptive Cooperative Multi-Agent Systems », **Bergenti F.**, **Gleizes M.-P.**, **Zambonelli F.** (dir.), *Methodologies and Software Engineering for Agent Systems*, Kluwer Publishing, p. 157-176, 2004.

[RAO 95] **Rao A. S.**, **Georgeff M. P.**, BDI-Agents: from Theory to Practice, Rapport n° 56, Australian Artificial Intelligence Institute, Melbourne, Australie, 1995.

[RIC 01] **Ricordel P.-M.**, Programmation orientée multi-agent : développement et déploiement de systèmes multi-agents Voyelles, Thèse de doctorat, Institut national polytechnique de Grenoble, 2001.

[ROU 07] **Rougemaille S.**, **Migeon F.**, **Maurel C.**, **Gleizes M.-P.**, « Model Driven Engineering for Designing Adaptive Multi-Agent Systems », *International Workshop on Engineering Societies in the Agents World (ESAW'07)*, Athènes, Grèce, 22/10/07-24/10/07, Springer, 2007.

[SAN 05] **Sansores C.**, **Pavón J.**, « Agent-Based Simulation Replication: A Model Driven Architecture Approach », *MICAI*, Springer, p. 244-253, 2005.

[SHE 01] **Shehory O.**, **Sturm A.**, « Evaluation of Modeling Techniques for Agent-Based Systems », *Proceedings of the Fifth International Conference on Autonomous Agents*, ACM Press, p. 624-631, 2001.

[STU 03] **Sturm A.**, **Shehory O.**, « A Framework for Evaluating Agent-Oriented Methodologies », *Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2003), Melbourne, Australia, at AAMAS'03*, vol. 3030 de *LNCS*, Springer, p. 94-109, 2003.

[SUD 04] **Sudeikat J.**, **Braubach L.**, **Pokahr A.**, **Lamersdorf W.**, « Evaluation of Agent-Oriented Software Methodologies – Examination of the Gap Between Modeling and Platform », *Agent Oriented Software Engineering*, p. 126-141, 2004.

[THI 03] **Thiefaine A.**, **Guessoum Z.**, **Perrot J.-F.**, **Blain G.**, « Génération de systèmes multi-agents à partir de modèles », *JFSMA'03*, Hermès, p. 107-111, 2003.

[TRE 98] **Treur J.**, Report of the First SIG Meeting, Contribution to Methodologies/software engineering SIG, Rapport, First SIG Meeting – AgentLink, 1998.

[WOO 00] **Wooldridge M.**, **Jennings N. R.**, **Kinny D.**, « The Gaia Methodology for Agent-Oriented Analysis and Design », *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 3, n° 3, p. 285-312, Kluwer Academic Publishers, 2000.