Table des matières

Liste des Figures

Chapitre 01 : La Qualité Logicielle

Chapitre 02 : Génie Logiciel Orienté-Agent

Chapitre 03 : La Qualité des Systèmes Multi-Agents

Chapitre 04 : Modèles de Qualité pour les Systèmes Multi-Agents

Chapitre 05 : Applications du Modèle QM4MAS

Chapitre 06 : Evaluation de la Complexité des Systèmes Multi-Agents

Liste des Tableaux

Chapitre 01 : La Qualité Logicielle

Chapitre 03 : La Qualité des Systèmes Multi-Agents

Chapitre 04 : Modèles de Qualité pour les Systèmes Multi-Agents

Chapitre 05 : Applications du Modèle QM4MAS

Chapitre 06 : Evaluation de la Complexité des Systèmes Multi-Agents

Introduction Générale

1. Contexte général de la problématique

Aujourd'hui, le monde se caractérise par l'expansion de la concurrence dans tous les domaines. En conséquence, les utilisateurs cherchent de plus en plus la qualité. Malgré que l'origine de la qualité remonte à l'origine de l'homme avec la fabrication de ses premiers outils, l'étude du contrôle de la qualité est apparue après la seconde guerre mondiale aux Etats-Unis. Depuis ce temps là, l'étude de ce concept a évolué en touchant presque tous les produits.

Les logiciels représentent maintenant un produit omniprésent. En fait, ce produit est intégré dans notre vie à travers son application dans tous les produits à usage quotidien. En plus, les produits logiciels représentent des composants essentiels dans les produits complexes et critiques. Le développement de logiciels de mauvaise qualité n'est pas seulement déconseillé mais devient parfois intolérant. En conséquence, la gestion de la qualité représente une activité importante dans le processus de développement des logiciels. Contrairement aux présomptions qui considèrent la gestion de la qualité une activité post-implémentation, cette activité est considérée comme une activité *umbrella* parce qu'elle couvre tout le processus de développement [Pre01].

Il semble évident que la gestion de la qualité nécessite au premier lieu la spécification du concept « *qualité* ». Cet avis est soutenu par l'ambigüité, la complexité et les multiples de facettes qui caractérisent la qualité du logiciel. Pour faire face à ces lacunes, il semble inévitable de procéder à la modélisation de ce concept pour offrir une meilleure gestion de la qualité logicielle. Particulièrement, les modèles de qualité peuvent servir de support pour la spécification, l'évaluation et/ou la prédiction de la qualité [Wag13].

L'importance de la modélisation de la qualité se manifeste dans la consécration de premières études de la qualité des logiciels à la proposition de modèles de qualité. En conséquence, plusieurs modèles ont été proposés comme le modèle de McCall et *al.* [McC77] et le modèle de Boehem [Boe78]. Ces modèles représentent les différentes caractéristiques et attributs qui influencent sur la qualité des logiciels. Malgré leurs anciennetés, ces modèles représentent encore des bases valides pour l'étude de la qualité.

La diversité de modèles de qualité est une conséquence naturelle à la diversité de points de vue concernant ce concept. Cependant, cette diversité représente un handicap réel durant l'échange de résultats ou la comparaison de différents produits. Evidemment, l'utilisation de divers modèles pour la spécification ou l'évaluation de la qualité d'un produit logiciel va conduire aux divers résultats. En conséquence, il est important de proposer un modèle de qualité standard qui unifie les différents points de vue. L'*organisation internationale de normalisation* (*ISO*) a proposé un modèle standard de la qualité des produits logiciel appelé ISO/IEC 9126 [ISO01]. Ensuite, ce modèle a été révisé en vue de la proposition d'un nouveau modèle standard appelé ISO/IEC 25010 [ISO11].

La proposition de premiers modèles de qualité et la standardisation de ceux-ci ne représente qu'un pas vers l'évolution de ce domaine. En fait, la nature de produits logiciels a été complètement changée depuis l'apparition de premiers modèles à la fin des années soixante-dix. L'évolution du domaine de génie logiciel a fait apparaître des nouveaux paradigmes qui introduisent des nouveaux principes, des nouveaux concepts et de nouvelles techniques. Sans doute, les modèles de qualité doivent prendre en compte ces nouveautés introduites avec les nouveaux paradigmes. D'où, plusieurs modèles ont été proposés pour le paradigme objet [Alo98, Ban02], les applications B2B [Beh09], la programmation orientée aspect [Kum09, Kum12], …etc.

Le paradigme agent est un paradigme de programmation relativement nouveau. Constitué l'intersection de plusieurs domaines (comme génie logiciel, les systèmes distribués et l'intelligence artificielle), ce paradigme semble le plus adéquat pour le développement des systèmes complexes qui se caractérisent par la distribution d'exécution et l'imprévisibilité de comportements. Les particularités du paradigme agent ont conduit à la proposition de méthodes spécifiques pour le développement des logiciels basés sur les systèmes multi-agents (*SMA*). En fait, la problématique de développement des SMA fait l'objet essentiel de génie logiciel orienté-agents (*AOSE*). L'ingénierie des SMA est maintenant en pleine évolution grâce à la proposition continue de méthodes, de méthodologies et des techniques qui simplifient le développement des SMA.

Malgré que la qualité des logiciels représente une piste intéressante dans le domaine de génie logiciel, très peu de travaux ont abordé la question de la qualité des SMA [Alo08]. En addition, la plupart de travaux proposés visent la mesure des attributs de tels systèmes en ignorant l'introduction de ces mesures dans le cadre de la qualité. Les rares tentatives de la modélisation de la qualité des SMA sont limitées parce qu'elles omettent les caractéristiques fondamentales de la qualité des logiciels (comme : *la fiabilité, l'efficacité, la réutilisabilité*, …etc.) [Alo08].

Nous pensons que le domaine des SMA a atteint un niveau de maturité important qui nécessite l'étude de la qualité des logiciels basés SMA. Cependant, nous sommes persuadés que la qualité des SMA n'a pas encore été abordée de manière profonde. Un modèle de qualité des SMA doit présenter dans une *vue unifiée* les caractéristiques de qualité des SMA en tant que produits logiciels avec les spécificités inhérentes à ce paradigme. Dans ce contexte, la problématique de notre thèse émerge de l'absence d'un modèle des SMA avec cette vue unifiée. Un tel modèle représente, comme sus mentionné, la brique de base pour la gestion de la qualité des SMA.

2. Contributions

Dans le cadre de l'étude de la qualité des SMA, nous avons classifié les différents travaux effectués dans ce domaine selon deux axes : les travaux proposants des modèles de qualité spécifiques aux SMA [[Alo08, Alo09, Alo10a, Bit12] et les travaux proposant l'évaluation de certains attributs de la qualité [Dum10, Siv12, Mah14]. En plus, l'étude de ces différents travaux nous permet d'identifier les insuffisances majeures qui affectent ce domaine. Ces insuffisances peuvent être résumées dans l'absence d'un modèle de qualité global qui, d'une part, met en relation les attributs affectant la qualité des SMA et, d'autre part, relie ces derniers avec les caractéristiques génériques de la qualité des logiciels. Nous pensons que l'existence de modèles de qualité des logiciels indépendants de paradigmes de programmation représente un point d'amorçage pour le développement des modèles de qualité spécifiques aux SMA. Tout d'abord, une vue synthétique de notre problématique ainsi que nos contributions et perspectives ont été présentées dans le consortium doctoral de la conférence internationale MATES 2014 [Mar14c]. Nos contributions proposées dans le cadre de cette thèse peuvent être organisées autour de trois axes :

- **La modélisation de la qualité des SMA :** afin de développer un modèle de qualité spécifique aux SMA, nous nous sommes basés sur les modèles standards existants. Notre première contribution consiste au développement d'un modèle de qualité pour les SMA appelé *QM4MAS* (Quality Model for Multi-Agent Systems) basé sur le modèle ISO/IEC 9126 [Mar15]. Le modèle proposé offre l'avantage de combiner les caractéristiques de la qualité des logiciels définies dans le modèle standard avec les caractéristiques intrinsèques des SMA. Ensuite, nous avons proposé notre deuxième contribution qui consiste à une extension du modèle ISO/IEC 25010 pour supporter les spécificités des SMA [Mar14a].
- **L'évaluation de la qualité des SMA :** dans le cadre de l'application de notre modèle proposé (*QM4MAS*) sur des plateformes multi-agents spécifiques, nous avons proposées des métriques spécifiques pour deux plateformes connues. Notre première contribution dans ce cadre consiste à l'application de notre modèle sur la plateforme *JADE* en tant que plateforme indépendante de modèles d'agents [Mar15]. Dans notre deuxième contribution, nous avons choisi une plateforme fondée sur un modèle spécifique d'agents. Il s'agit de la plateforme *DIMA*.
- **L'étude des attributs particuliers :** dans le cadre de l'étude des attributs particuliers des SMA, nous avons remarqué que la complexité des SMA n'est pas étudiée profondément malgré son impact sur la qualité de tels systèmes. En effet, notre contribution dans ce cadre consiste à la modélisation et la mesure de cet attribut en tenant compte les spécificités des SMA [Mar14b].

3. Plan de la thèse

La présente thèse est organisée en six chapitres. Les trois premiers chapitres représentent l'état de l'art de domaines attachés à notre problématique. Ensuite, nous présentons nos contributions dans les trois chapitres suivants. Ainsi, ce manuscrit est composé des chapitres suivants :

- **La qualité logicielle :** le premier chapitre est consacré à la présentation du domaine de la qualité des logiciels en montrant les pistes pertinentes à notre thème : la modélisation de la qualité, la standardisation des modèles, les spécificités des modèles envers les paradigmes logiciels.

- **Génie logiciel orienté-agent :** dans le deuxième chapitre nous avons présenté l'ingénierie des SMA en mettant l'accent sur la modélisation des SMA et l'implémentation de ceux-ci. Bien entendu, nous avons introduit au début de ce chapitre les concepts de base du paradigme agent.
- **La qualité des systèmes multi-agents :** ce chapitre présente l'intersection de deux domaines précédents en détaillant les différents travaux ciblant la qualité des SMA. Une étude comparative permettant de tirer les insuffisances majeures de travaux présentés est également présentée dans ce chapitre.
- **Modèles de qualité pour les systèmes multi-agents :** dans le quatrième chapitre, nous avons présenté nos deux contributions essentielles. La première contribution consiste au développement d'un modèle de qualité basé sur le standard ISO/IEC 9126 pour les SMA, appelé *QM4MAS*. Notre deuxième contribution représente un premier pas vers l'extension du nouveau standard ISO/IEC 25010 pour supporter les particularités des SMA.
- **Applications du modèle QM4MAS :** dans ce chapitre nous avons appliqué le modèle proposé sur deux plateformes multi-agents à travers la définition d'un ensemble de mesures.
- **L'évaluation de la complexité des systèmes multi-agents :** le dernier chapitre consiste en une contribution pour l'évaluation d'un seul attribut de la qualité des SMA. Nous avons choisi l'évaluation de la complexité des SMA à cause de son impact sur la qualité de tels systèmes.

Chapitre 01

La Qualité Logicielle

« The Quality: I know it when I see it! »

1. Introduction

Le génie logiciel est apparu à la fin des années soixante suite à la crise du logiciel. Cette crise était due, à la fois, à l'augmentation de la complexité des logiciels d'une part et à la baisse significative de la qualité de ces derniers. En conséquence, le développement des *logiciels de qualité* est considéré parmi les objectifs majeurs de ce domaine. Malgré que la *qualité* ait été étudiée depuis longtemps dans le domaine industriel, les spécificités du produit logiciel prohibent l'application directe de techniques connues dans les autres domaines. En fait, le produit logiciel se caractérise par sa nature abstraite qui complique le processus de mesure de ses différents attributs. En plus, la référence de la qualité dans le domaine industriel est la *conformité aux besoins*. Cependant, on fait généralement recours aux spécifications incomplètes durant les premières phases de développement à cause de la difficulté des cette activité en génie logiciel. En outre, le génie logiciel est un domaine en pleine évolution qui a connu, depuis son apparition, la proposition de plusieurs paradigmes. Bien que ces paradigmes aient apportés de réelles avancées dans le développement de logiciel, ils ont leur influence sur la qualité du logiciel.

Nous présentons dans la première partie de ce chapitre un aperçu général sur la qualité logicielle. Ensuite, nous détaillons les trois pistes relatives à notre problématique, à savoir : la modélisation de la qualité logicielle, la standardisation des modèles et la mesure de la qualité.

2. Aperçu général

La qualité est un concept complexe et vague. En effet, pour certains chercheurs la question fondamentale n'est pas ***qu'est ce qu'on ignore de la qualité logicielle*** ?, mais ***qu'est ce qu'on croit en savoir*** ? [Cro79]. Ainsi, plusieurs définitions ont été proposées dans la littérature spécialisée. A titre illustratif, nous citons quelques définitions parmi les plus répandues :

- La qualité est la conformité aux besoins [Cro79] ;
- La qualité d'un logiciel peut être évaluée en mesurant ses fonctionnalités, sa maintenabilité, …etc. [Oul91] ;

- la qualité c'est le degré de conformité d'un système, d'une composante ou d'un processus à sa spécification, ou le degré de satisfaction de son utilisateur [IEEE90].

Comme nous pouvons le remarquer dans la définition de Crosby [Cro79] et la définition standard proposée par l'organisation IEEE [IEEE90], les besoins de l'utilisateur représentent la base sur laquelle la définition de la qualité a été fondée. En conséquence, il est impératif de spécifier la qualité (comme elle est vue par l'utilisateur) durant la spécification des besoins. Cette combinaison de la spécification des besoins avec la spécification de la qualité durant la phase de spécification engendre plusieurs inconvénients [Dro92]. Tout d'abord, la naïveté de l'utilisateur fait que l'implication de ce dernier pour l'établissement d'une spécification de la qualité est une tâche difficile. En plus, cette combinaison peut compliquer la spécification, ce qui influe sur les autres phases de développement. Il est important de souligner que la phase de spécification représente la phase la plus difficile dans le développement des logiciels. On fait souvent recours à des spécifications incomplètes à cause des problèmes d'élicitation des besoins. Il semble claire que la spécification de la qualité durant cette phase sera plus difficile que la spécification des besoins fonctionnelles des utilisateurs. Finalement, la conformité aux besoins de l'utilisateur ne représente pas la seule caractéristique exigée pour satisfaire la qualité. Il est fortement possible de concevoir ou implémenter un logiciel qui répond aux besoins de l'utilisateur mais avec une qualité médiocre. La qualité est plutôt un ensemble de caractéristiques. Ould [Oul91] propose l'évaluation de la qualité par la mesure des caractéristiques comme les fonctionnalités et la maintenabilité. Cependant, cette définition de la qualité ne clarifie pas la relation éventuellement contradictoire entre ces caractéristiques.

Il est important de noter que la complexité du concept qualité n'est pas exclusive au produit logiciel [Wag13]. En fait, la nature abstraite du logiciel augmente la complexité de ce concept. Plusieurs raisons contribuent à la complexité de ce concept [Kan02, Nai08] : la confusion entre le point de vue vulgaire et professionnel dans l'utilisation de ce concept, sa nature multi-niveaux d'abstraction et sa nature multidimensionnelles. Par la première raison on entend la différence entre un spécialiste professionnel (comme un concepteur, un gestionnaire de projet ou un gestionnaire de qualité) et un utilisateur naïf concernant la compréhension de ce concept. Malgré que la satisfaction de l'utilisateur soit un repère important de niveau de la qualité d'un produit, l'avis de ce dernier est généralement superficiel et n'est pas objectif. En plus, la qualité possède plusieurs niveaux d'abstraction En effet, la qualité de la spécification d'un produit est différente de la qualité de sa conception qui est à son tour différente de la qualité de son implémentation. Finalement, Garvin [Gar84] décrit la qualité selon plusieurs dimensions :

- **La vue transcendantale :** cette vue représente la qualité comme un attribut qu'on peut reconnaître mais qu'on ne peut ni définir ni mesurer ;
- **La vue de l'utilisateur :** selon cette vue la qualité est basée sur la satisfaction de l'utilisateur ;
- **La vue de fabrication :** un produit de qualité est un produit conforme à sa spécification ;
- **La vue du produit :** la qualité est représentée par les caractéristiques intrinsèques du produit ;
- **La vue basée sur la valeur :** la qualité dépend du coût du produit.

Wagner [Wag13] remarque que cette vue multidimensionnelles du concept qualité proposée par Garvin [Gar84] est une mise en application de la vue multi-niveaux d'abstraction. Ainsi, on exploite la vue de l'utilisateur et la vue basée sur la valeur pour définir les buts de la qualité. Ensuite, on raffine ces buts durant la phase de spécification pour spécifier les caractéristiques du produit (la vue du produit). Durant la phase de fabrication, on focalise sur les approches qui garantissent la conformité du produit à sa spécification (la vue de fabrication).

La vue multidimensionnelles de la qualité montre l'importance de la gestion de qualité en génie logiciel. Elle contredit les croyances de beaucoup de développeurs qui commencent la gestion de la qualité après la génération de code [Pre01]. La gestion de la qualité peut être qualifiée comme une activité *umbrella*, c'est-à-dire une activité qui se déroule durant toutes les phases d'un projet logiciel [Pre01]. Cette activité peut être divisée en trois tâches principales [Som07]: l'assurance de qualité, le contrôle de qualité et la planification de qualité. Par l'assurance de qualité on désigne l'établissement de framework de procédures organisationnelles et standards pour l'obtention d'un logiciel de qualité. Une fois le framework établi, la planification de qualité consiste à choisir parmi ces procédures et standards les plus appropriées pour un projet spécifique. En fin, le contrôle de qualité consiste à définir le processus qui assure le suivi des standards par le groupe de développement.

Sans doute, la qualité d'un produit est liée au processus de sa fabrication. Malgré que l'origine de cette proposition soit le monde industriel d'une part et la complexité de relation entre le produit logiciel et le processus de son développement d'autre part ; l'expérience montre, en effet, la relation entre la qualité de ce produit et le processus de son développement [Per01, Kan06, Som07, Wag13].

La qualité du processus est liée à la manière du développement du produit [Kan06]. L'importance de ce type de qualité est due au caractère mesurable du processus logiciel ce qui améliore la qualité du produit développé [Pre01]. En fait, pour pouvoir améliorer la qualité du processus logiciel (et par conséquence améliorer la qualité du produit logiciel), l'organisation du développement doit évaluer ses compétences et ses limites. Cela représente un élément essentiel et décisif pour la confiance des clients [Nai08].

Comme susmentionnée, la relation entre le produit logiciel et le processus suivi pour son développement est complexe. Le produit logiciel est un produit développé et non fabriqué. En effet, plusieurs facteurs externes peuvent affecter le processus de développement logiciel ce qui influence sur la qualité du produit logiciel [Nai08]. En plus, l'un des problèmes majeurs rencontrés lors de l'application d'une gestion de qualité basée processus est l'ignorance du type de logiciel à développer [Som07]. Plusieurs alternatives insistent sur la gestion de la qualité d'un produit logiciel au lieu du processus de développement. Dans cette thèse, nous traitons la qualité des SMA au point de vue produit logiciel. L'étude de la qualité du produit logiciel revient aux années soixante-dix avec la proposition des premiers modèles de qualité logicielle. Nous présentons dans la section suivante la modélisation de la qualité logicielle. Il est important de noter que la vieillesse des modèles présentés à l'aube de ce domaine et le changement quasi radical des approches de développement logiciel n'affectent pas profondément son applicabilité [Pre01].

3. Modélisation de la qualité logicielle

La qualité est un concept complexe avec plusieurs caractéristiques éventuellement contradictoires. En conséquence, la modélisation représente un moyen qui facilite la compréhension de ce concept. Un modèle de qualité combine les différents attributs contribuant à la qualité dans un seul framework. Evidement la relation entre ces attributs est primordialement dépendante de l'approche utilisée pour la construction du modèle. Un modèle de qualité représente un moyen bien accepté au soutien de contrôle de la qualité du logiciel [Wag13]. Généralement, un modèle de qualité permet de [Sur14] :

- Aider à comprendre comment les différentes facettes de la qualité contribuent à la qualité globale ;
- Souligner clairement que la qualité n'est pas seulement les défauts et les erreurs ;
- Aider à identifier et définir les besoins de la qualité ;
- Aider à naviguer à travers les différents niveaux de définition de la qualité ;
- Aider à définir un profile d'évaluation (précisément ce qui doit être évalué).

Malgré que l'idée de base de la modélisation de la qualité soit la même (*la combinaison des attributs de la qualité dans un seul framework*), les modèles résultants peuvent être utilisés pour des objectifs divers. En effet, un modèle de qualité peut être utilisé pour la définition, l'évaluation et/ou la prédiction de la qualité [Wag13]. La figure 1.1 représente la relation entre ces objectifs.

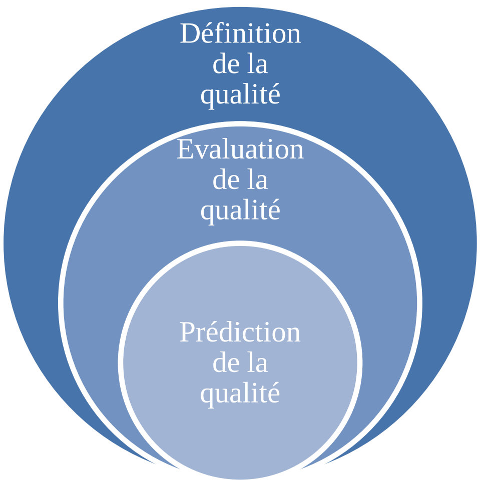


Figure 1.1 : La relation entre les objectifs des modèles de la qualité logicielle [Wag13].

La relation entre les objectifs des modèles de qualité permet de les classifier afin d'éviter des comparaisons infructueuses. Ainsi, il existe des modèles pour la définition de la qualité, des modèles pour l'évaluation de la qualité et des modèles pour la prédiction de la qualité. Comme il est montré dans la figure 1.1, ces objectifs ne sont pas totalement indépendants. En fait, l'évaluation de la qualité passe généralement par la définition de celle-ci. De manière similaire, il est difficile de prédire la qualité sans pouvoir l'évaluer. Cependant, cette relation n'est valide que dans le cas idéal [Wag13]. Le modèle MI (*Maintainability Index)* [Col95] est un modèle d'évaluation qui ne spécifie pas une définition claire de la qualité. En plus, le modèle RGM (*Reliability Growth Model*) [Luy96] est un modèle prédictif basé sur les données. Il n'est pas explicitement relié à un modèle global de la définition de la qualité.

Les premiers modèles de qualité ont été proposés à la fin des années soixante-dix. En fait, les modèles de McCall et *al.* [McC77] et Boehm et *al.* [Boe78] représentent les premières initiatives pour la modélisation de la qualité. Ces deux modèles partagent la même idée de base : *la décomposition de la qualité en caractéristiques et sous-caractéristiques*. On parle alors de modèles de qualité hiérarchiques. A titre d'exemple nous présentons dans le reste de cette section le modèle de McCall et *al.* [McC77].

Considérés parmi les premiers qui se sont intéressés à la qualité logicielle, McCall et ses collègues orientent ses recherches vers la définition de qualité en termes de facteurs et de critères [McC77]. Le modèle intitulé modèle de McCall (ou bien facteurs de McCall) représente le résultat de ces travaux. Ce modèle est constitué de onze facteurs de qualité. Par facteur de qualité, McCall et *al.* désignent les caractéristiques comportementales du système. Une brève définition est donnée pour chaque facteur dans le tableau 1.1.

Ces facteurs sont organisés autour de trois façades : le fonctionnement du produit, sa révision et sa transition. Sous la catégorie fonctionnement du produit s'organisent les facteurs de qualité suivants : Correctitude, Fiabilité, Efficacité, Intégrité et Utilisabilité. La Maintenabilité, la Testabilité et la Flexibilité représentent des facteurs de révision du produit. Tandis que, la Portabilité, la Réutilisabilité et l'Interopérabilité sont des facteurs de la transition du produit. Bien entendu, l'importance de ces facteurs varie d'un acteur (développeurs, clients, ingénieurs de qualité) à un autre [Nai08].

Tableau 1.1 : Définition de facteurs de McCall et *al.*

Facteurs	Définitions
Correctitude	Mesure la capacité du programme de satisfaire ses spécifications et réalise les objectifs de l'utilisateur.
Fiabilité	Mesure la capacité du programme pour exécuter ses fonctions avec la précision exigée.
Efficacité	La quantité des ressources et du code demandés par le programme pour exécuter ses tâches
Intégrité	La capacité de contrôler l'accès aux données ou au programme par une personne non autorisée.
Utilisabilité	L'effort demandé pour apprendre, faire fonctionner le système, préparer les entrées et interpréter les sorties.
Maintenabilité	L'effort demandé pour localiser et corriger un bug dans un système opérationnel.
Testabilité	L'effort demandé pour tester un programme pour s'assurer qu'il exécute les fonctions demandées.
Flexibilité	L'effort demandé pour modifier un programme opérationnel.
Portabilité	L'effort demandé pour transformer un programme d'une plateforme à une autre.
Réutilisabilité	La capacité de réutiliser une partie d'un système dans un autre.
Interopérabilité	L'effort demandé pour coupler un système avec un autre.

La mesure de la qualité à partir de ces facteurs est une tâche difficile et peut être vue comme subjective. McCall et *al.*, utilisent des critères de qualité simples à mesurer pour évaluer le produit logiciel. Par critère de qualité on désigne un attribut de facteur de qualité lié au développement du logiciel [Nai08]. Par exemple, le facteur de qualité *réutilisabilité* est difficile à mesurer de façon objective. Cependant, tous les développeurs de logiciels sont d'accord que la *modularité* est un facteur influant sur la *réutilisabilité*. Il est donc simple d'évaluer la réutilisabilité à partir de la modularité. Cette dernière représente un critère de qualité. McCall et *al*. définissent vingt-trois critères de qualité [Nai08]. Le modèle de qualité montre la relation entre les facteurs et les critères de qualité comme il est illustré par la figure 1.2.

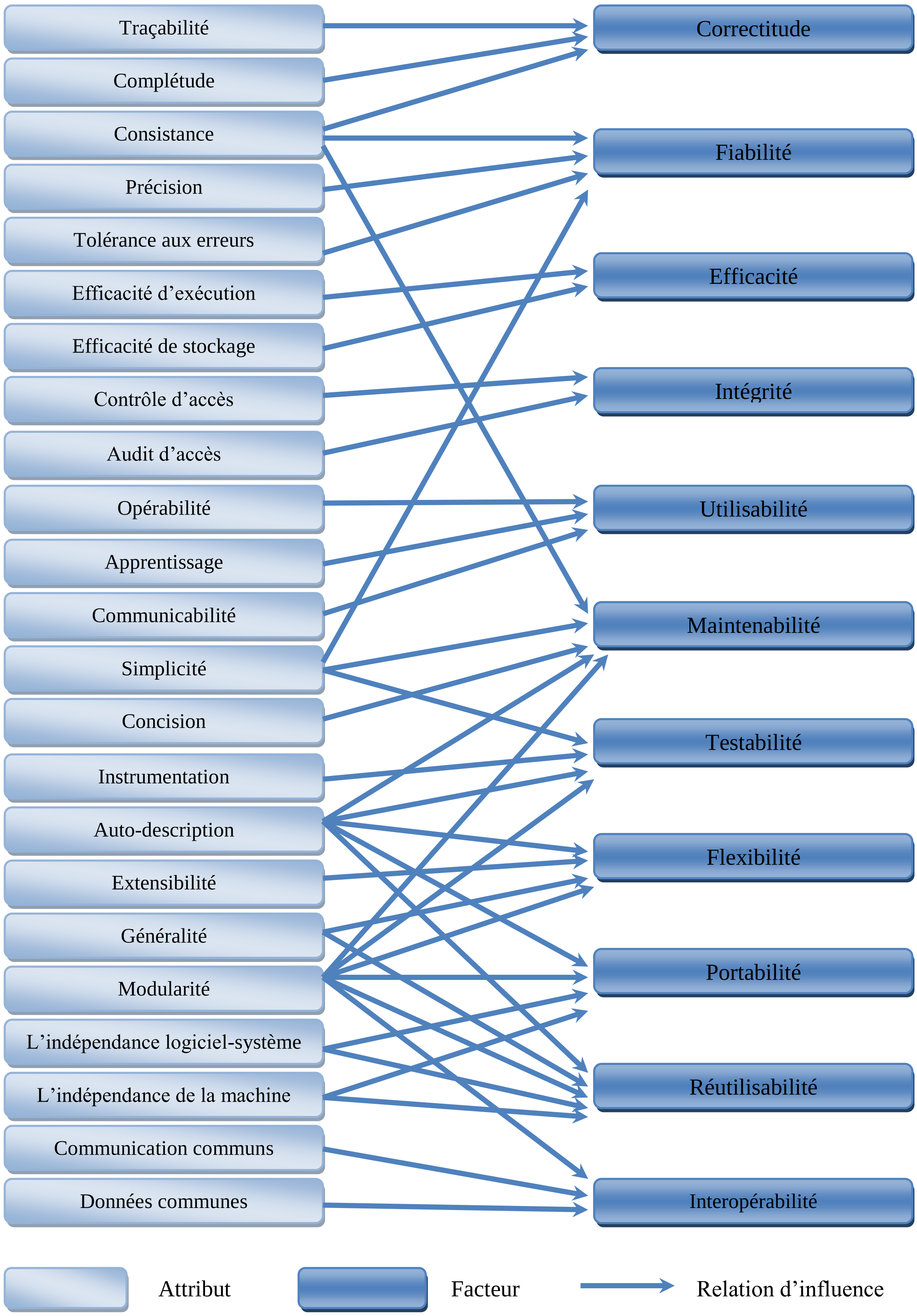


Figure 1.2 : Le modèle de la qualité de McCall et *al.*

Les différents facteurs de qualité du modèle de McCall et *al.* ne sont pas totalement indépendants. En fait, certains facteurs influent sur d'autres de manière négative ou positive. Par exemple, l'augmentation de testabilité d'un système peut dégrader l'efficacité de ce dernier [Nai08]. Comme influence positif, on peut citer les deux facteurs *fiabilité* et *correctitude*. Il est évident que l'amélioration de la correctitude améliore à son tour la fiabilité du système.

Vu ces relations entre les facteurs de qualité, il est impossible d'améliorer tous ces facteurs même s'il est désirable [Nai08]. C'est à l'organisation du développement de choisir parmi ces facteurs les plus importants pour le projet à développer.

Les travaux des groupes de recherche indépendants avec un point de vue spécifique concernant le concept qualité logicielle pour chaque groupe ont engendré plusieurs modèles de qualité. Cette diversité de modèles complique la comparaison de différents produits. Ainsi, la nécessité de développer un seul modèle de qualité qui regroupe les différentes vues de la qualité du logiciel a été manifestée pour la communauté de la qualité logicielle. Les travaux de la standardisation des modèles de la qualité logicielle, présentés dans la section suivante, entrent dans ce cadre.

4. La standardisation des modèles de qualité logicielle

La standardisation dans le domaine du développement des logiciels au niveau international fait l'objet de recherche continue [Nai08]. Plusieurs organisations sont impliquées dans ce sujet et *l'organisation internationale de normalisation* (ISO) ne fait pas l'exception. En fait, l'ISO s'intéresse depuis les années 1940 au développement des standards dans le domaine de l'assurance et la gestion de qualité dans le cadre de la série ISO 9000. Ces standards sont applicables sur tous nos produits quel que soient leurs niveaux de complexité.

Dans le cadre de la qualité logicielle, un groupe d'experts définit, dans l'ISO, un modèle de qualité appelé ISO/IEC 9126 [ISO01]. Un modèle de qualité est défini par ce standard comme *un ensemble de caractéristiques reliées entre elles afin de spécifier ou évaluer la qualité* [ISO01]. Il décrit la qualité selon trois aspects :

- **La qualité interne :** l'ensemble d'attributs du produit qui déterminent sa capacité de satisfaire les besoins implicites et explicites quand on l'utilise sous des conditions spécifiques ;
- **La qualité externe :** c'est l'ensemble des caractéristiques du produit selon un point de vue externe. Cet aspect de qualité se manifeste durant l'exécution du produit ;
- **La qualité en utilisation :** cet aspect représente le point de vue de l'utilisateur concernant la qualité. Il représente la qualité du produit quand on l'utilise dans un contexte spécifique.

Les aspects interne et externe de la qualité ont été modélisés par un modèle hiérarchique organisé autour de six caractéristiques de qualité : la *fonctionnalité, la fiabilité, l'utilisabilité, l'efficacité, la maintenabilité et la portabilité* (tableau 1.2). Chaque caractéristique est divisée en sous caractéristiques. En plus, chaque sous-caractéristique peut être évaluée par un ensemble de mesures. Ce standard propose une liste de plus de deux-cent métriques pour la mesure de la qualité interne et externe. Néanmoins, il est possible d'étendre cette liste par des mesures supplémentaires à cause du caractère non-exhaustif de mesures proposées. La figure 1.3 présente la relation entre les caractéristiques et les sous-caractéristiques du modèle ISO/IEC 9126.

Tableau 1.2 : Définition de caractéristiques du modèle ISO/IEC 9126.

Caractéristique	Définition
Fonctionnalité	Mesure la capacité de programme de satisfaire ses spécifications et réalise les objectifs de l'utilisateur
Fiabilité	Mesure la capacité de programme pour exécuter ses fonctions avec la précision exigée
Efficacité	La quantité des ressources et de code demandés par le programme pour exécuter ses tâches
Portabilité	La capacité de contrôler l'accès aux données ou au programme par une personne non autorisée
Utilisabilité	L'effort demandé pour apprendre, fonctionner le système, préparer les entrées et interpréter les sorties
Maintenabilité	L'effort demandé pour localiser et corriger un bug dans un système opérationnel

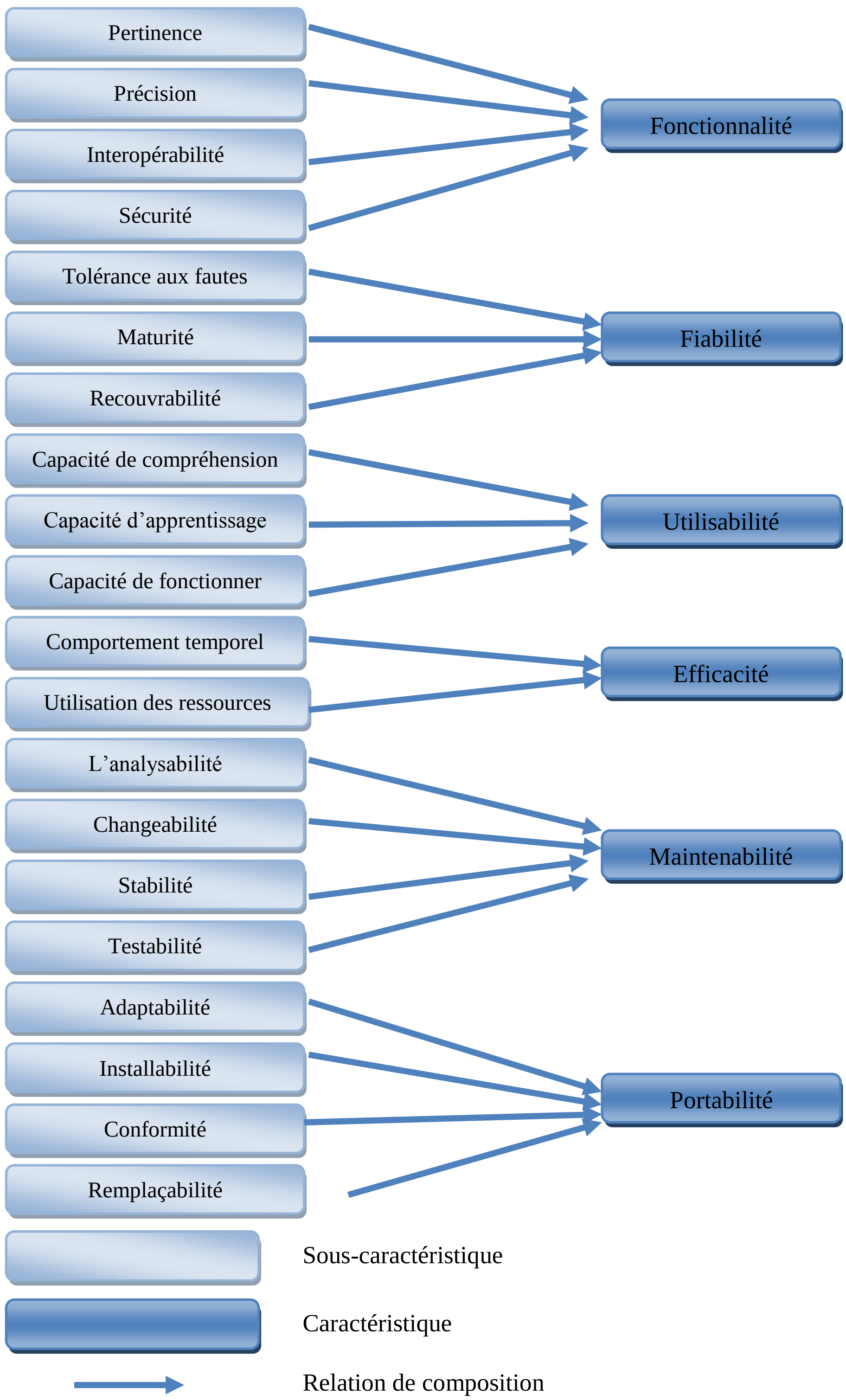


Figure 1.3 : Le modèle de qualité ISO/IEC 9126.

La qualité en utilisation est constituée de quatre caractéristiques : l'effectivité, la productivité, la sécurité et la satisfaction. Les différentes caractéristiques peuvent être directement mesurées par un ensemble de mesures proposées.

Il est important de noter que le choix de caractéristiques formant le modèle de qualité ISO/IEC 9126 est guidé par plusieurs buts dont les plus importants sont [ISO01] :

1) Couvrir ensemble la définition de la qualité proposée par l'ISO ;
2) Décrire la qualité du produit avec le minimum de chevauchement ;
3) Etre entre six et huit caractéristiques pour des raisons de clarté et de manipulation.

Le modèle standard ISO/IEC 9126 est compatible avec les différentes facettes de la qualité proposées par Garvin [Gar84] que nous avons citées auparavant [Sur14]. En fait, le modèle de la qualité interne et externe représente la vue de fabrication et la vue du produit. Par contre, le modèle de la qualité en utilisation représente la vue basée sur la valeur et la vue de l'utilisateur. Naturellement, la vue transcendantale ne peut être explicitement représentée à cause de sa nature. En plus, ce modèle se caractérise par un degré accepté d'unanimité grâce à la nature internationale de l'organisation qui le développe [Sur14].

Depuis sa dernière version apparue en 2001, le modèle ISO/IEC 9126 n'a pas connu de grandes modifications. En fait, une enquête à travers le Web a été initiée par un group de travail de l'ISO afin de collecter les feedbacks de la communauté du génie logiciel. Cette initiative avait pour but de préparer la deuxième génération des modèles de qualité du produit logiciel. Ainsi, une série de modèles appelée SQuaRE (pour *Systems and software Quality Requirements and Evaluation*) a été proposée à cet effet [Sur14]. Cette série est constituée de quatorze modèles organisés en cinq sections suivantes :

- Gestion de qualité (ISO/IEC 2500n) ;
- Modèles de qualité (ISO/IEC 2501n) ;
- Mesure de qualité (ISO/IEC 2502n) ;
- Besoins de qualité (ISO/IEC 2503n) ;

- Evaluation de qualité (ISO/IEC 2504n).

Le modèle de qualité proposée au sein de cette série (appelé ISO/IEC 25010) (Figure 1.4) trouve sa continuité dans la même approche proposée par son prédécesseur (le modèle ISO/IEC 9126). Ainsi, il propose deux vues distinctes de la qualité : le modèle de qualité en utilisation et le modèle de qualité du produit logiciel/système.

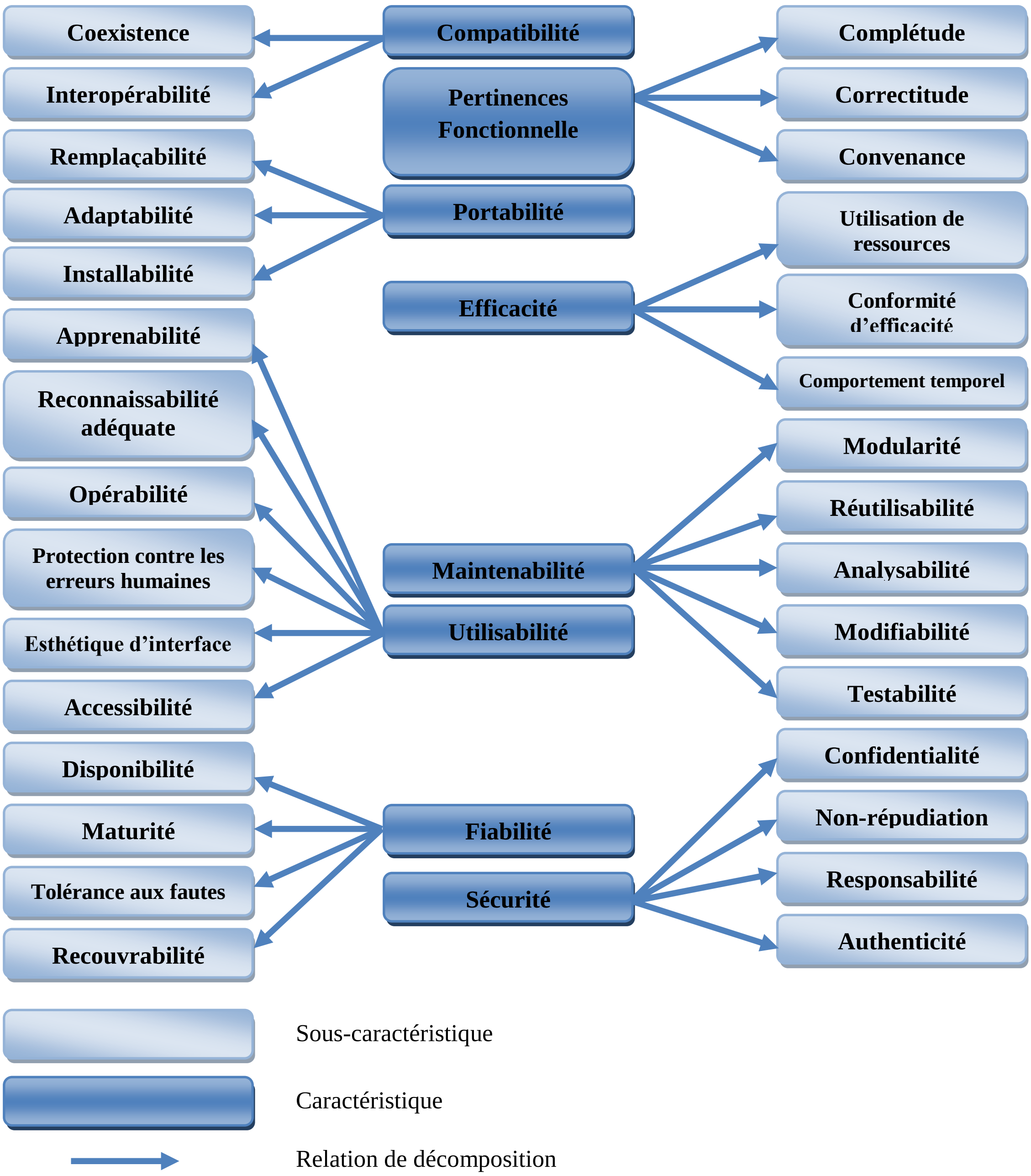


Figure 1.4 : Le modèle de qualité ISO/IEC 25010.

Par rapport au modèle ISO/IEC 9126, les révisions proposées dans le modèle ISO/IEC 25010 touchent essentiellement les aspects suivants [ISO11] :

- Le modèle a été étendu pour supporter les systèmes informatiques ;
- La sécurité a été ajoutée comme une caractéristique au lieu d'être une sous-caractéristique dans le modèle ISO/IEC 9126 ;
- La compatibilité a été ajoutée comme une caractéristique ;
- La liste des sous-caractéristiques a été modifiée en supprimant des sous-caractéristiques ou en ajoutant des nouvelles ;
- Le modèle ne considère qu'une seule vue de la qualité du produit en ignorant les deux façades : la qualité interne et la qualité externe.

5. La qualité et les produits logiciels spécifiques

Depuis l'apparition des premiers modèles de qualité à la fin des années soixante-dix, le domaine du génie logiciel a connu un changement quasi-radical. En fait, ce changement n'affecte pas seulement les méthodes et les techniques du développement mais il touche aussi, en profondeur, les paradigmes du développement logiciel. Plusieurs nouveaux paradigmes ont vu le jour durant ces dernières décennies. Naturellement, la modélisation de la qualité doit prendre en compte les caractéristiques spécifiques de différents paradigmes ou de certains produits logiciels. La spécification de modèles de qualités relatifs à des paradigmes particuliers peut être appliquée suivant deux approches essentielles [Rad11] :

- **L'approche ascendante :** dans cette approche on commence par des mesures pour définir les différentes sous-caractéristiques qui sont à leur tour combinées pour former des caractéristiques ;
- **L'approche descendante :** contrairement à l'approche précédente on commence par les caractéristiques pour définir les sous-caractéristiques. Ces dernières sont ensuite reliées à des mesures.

Dans le domaine de la qualité des produits basés sur des paradigmes spécifiques, on distingue deux différentes classes de travaux : le développement de nouveaux modèles de qualité pour spécifier les nouveautés du produit ciblé ou l'adaptation des modèles existants pour exprimer les particularités du produit étudié.

5.1. Les modèles conçus pour des produits spécifiques

Les travaux inclus dans cette catégorie sont basés sur une idée simple : on ne peut pas utiliser les modèles de qualité génériques pour capter les particularités de divers produits logiciels spécifiques. Ainsi, la proposition des modèles spécifiques conçus dès le départ pour des produits spécifiques représente la tendance alternative. Plusieurs travaux ont été proposés dans ce cadre. Nous expliquons cette catégorie à travers deux cas illustratifs.

Bansiya et Davis [Ban02] remarquent que l'introduction de nouveaux concepts dans le paradigme orienté-objet par rapport à l'approche procédurale (comme l'encapsulation, l'héritage et le polymorphisme) a changé la manière d'évaluer la qualité. Malgré la variété des métriques proposées dans le cadre du paradigme orienté-objet, ils [Ban02] remarquent l'existence de plusieurs problèmes qui empêchent l'utilisation de ces métriques :

- La validation de ces métriques est faite par un petit ensemble de données ;
- Ces métriques ne sont pas reliées aux facteurs externes de qualité ;
- Ces métriques sont applicables seulement après le développement du logiciel parce que les informations sont collectées à partir de l'implémentation de ce logiciel.

Ainsi, les auteurs [Ban02] ont proposé un modèle de qualité pour les conceptions orientées-objets appelé QMOOD (pour *Quality Model for Object Oriented Design*). Ce modèle est un modèle hiérarchique composé de trois niveaux : les attributs, les propriétés et les métriques. Les attributs définis dans le modèle sont : fonctionnalité, effectivité, compréhensibilité, extensibilité, réutilisabilité et flexibilité. Cette liste d'attributs est choisie après revue et critique des attributs du modèle ISO/IEC 9126. Cependant, la liste des attributs, d'après les auteurs [Ban02], n'est pas exhaustive. Les propriétés de conception représentent des concepts tangibles qu'on peut estimer par l'analyse de la structure interne, de la structure externe, des relations, des fonctionnalités des composantes, des attributs, des méthodes et des classes. Ainsi les propriétés définies dans le modèle sont : la taille de conception, la hiérarchie, l'abstraction, l'encapsulation, le couplage, la cohésion, la composition, l'héritage, le polymorphisme, la complexité et le passage des messages (messaging). Au troisième niveau, on trouve des mesures utilisées pour l'évaluation empiriques de différentes propriétés. En fait, ce niveau est conçu en déterminant le nombre minimum de mesures pour évaluer *efficacement* chaque propriété. En conséquence, chaque propriété est assignée à une seule mesure. Le tableau 1.3 présente les relations entre les attributs et les propriétés du modèle QMOOD. Par rapport à la version présentée par Bansiya et Davis [Ban02], nous avons ajouté, dans cette présentation, les forces d'influence négative (représentée par une flèche descendante). On note que ces relations sont modélisées par des poids (négatifs ou positifs) afin d'exprimer le degré d'influence de la propriété sur l'attribut.

Tableau 1.3 : Le modèle QMOOD (inspiré de [Ben02]).

	Fonctionnalité	Effectivité	Compréhensibilité	Extensibilité	Réutilisabilité	Flexibilité
Taille de Conception	↑		↓		↑	
Hiérarchie	↑					
Abstraction		↑	↓	↑		
Encapsulation		↑	↑			↑
Couplage			↓	↓	↓	↓
Cohésion	↑		↑		↑	
Composition		↑				↑
Héritage		↑		↑		
Polymorphisme	↑	↑	↓	↑		↑
Complexité			↓			
Passage des Messages	↑				↑	

Le deuxième cas de modèles conçus dès le début pour des produits logiciels spécifiques est la modélisation de la maintenabilité proposée par [Dei07]. La maintenabilité représente un attribut clé de la qualité du logiciel. Cependant, la modélisation de cet attribut ignore les activités qui peuvent influencer sur son niveau. Cette ignorance prend la forme de confusion entre les propriétés du logiciel influençant sur la maintenabilité et les activités attachées à la maintenance. Par exemple, le modèle de la qualité de Boehm et *al.* [Boe78] combine des activités (comme la testabilité, la compréhensibilité et l'augmentabilité) avec des propriétés du logiciel (comme l'auto-description et être communicatif). Cette confusion engendre des ambigüités durant l'exploitation du modèle. En effet, les auteurs [Dei07] ont proposé la distinction de ces deux aspects différents. En plus, ils ont remarqué que ce n'est pas seulement les propriétés du logiciel qui influent sur la maintenabilité mais il existe plusieurs facteurs qui peuvent affecter cet attribut (comme les compétences des ingénieurs et la disponibilité des outils adéquats). Ces facteurs, appelées des situations, sont ensuite décomposés en des entités atomiques appelées des faits. En conséquence, le modèle généré est un modèle à deux dimensions qui combine les situations et les activités affectant la maintenabilité de logiciels (Figure 1.5). Les trois premières lignes de ce modèle représentent la décomposition des activités. Par contre, les quatre premières colonnes représentent la décomposition de situation. Les intersections entre les lignes et les colonnes représentent l'impact éventuel de la situation sur l'activité.

				Maintenance			
				L'analyse		L'implémentation	
				Location de concepts	Analyse d'impact	Codage	Modification
Situation	Produit	Aspects dynamique	Concurrence		✓		✓
			Récursivité		✓		✓
		Aspects Statiques	Identificateur	✓		✓	
			Clonage		✓		✓
			Format de code			✓	
	Infrastructure	Outils	Débuggeur	✓	✓		
			Refactoring				✓

Figure 1.5 : Modèle basé-activité de la maintenabilité [Dei07].

5.2. Les modèles étendus à partir des modèles existants

Contrairement aux approches citées dans la section précédente, une autre alternative consiste à profiter de la généricité des modèles existants pour spécifier la qualité des produits logiciels particuliers. En fait, l'exploitation des modèles existants limite la diversité des modèles de qualité qui est une source d'ambigüité. En plus, cette tendance permet la comparaison de différents produits en utilisant des frameworks semblables.

En 1998, F. Alonso et *al*. [Alo98] ont remarqué qu'il y avait peu de travaux sur les modèles de qualité des logiciels orientés-objets. En conséquence, ils ont pris pour base le modèle de McCall et *al*. [McC77] pour développer un modèle de qualité pour ces logiciels. Les auteurs jugent que les facteurs proposés par McCall et ses collègues [McC77] sont suffisants pour la spécification de la qualité des logiciels orientés-objets. Ce point de vue est justifié par la nature de facteurs de qualité indépendante de paradigmes du développement. En fait, ces facteurs représentent des caractéristiques externes de logiciels. Par contre, les critères de qualité ont été étendus par la documentation, la stabilité et la structure. Les relations entre les critères et les facteurs sont présentées dans le tableau 1.4.

Tableau 1.4 : Les relations entre les facteurs et les critères du modèle de Alonso et *al.*

Facteurs	Critères
Correctitude	Traçabilité, Complétude, Consistance
Fiabilité	Complétude, Consistance, Précision, Tolérance aux erreurs, Simplicité
Efficacité	Efficacité d'exécution, Efficacité de stockage
Intégrité	Contrôle d'accès, Audit d'accès
Utilisabilité	Opérabilité, Apprentissage, Capacité de Communication, Documentation
Maintenabilité	Concision, Consistance, Modularité, Auto-description, Documentation, Simplicité, Stabilité, Structure, Traçabilité.
Testabilité	
Flexibilité	Extensibilité, Généralité, Modularité, Simplicité, Auto-description, Documentation, Stabilité et structure.
Portabilité	Documentation, Modularité, Structure, Auto-description, L'indépendance logiciel-système, L'indépendance de la machine.
Réutilisabilité	Documentation, Modularité, Structure, Auto-description, L'indépendance logiciel-système, L'indépendance de la machine, Généralité.
Interopérabilité	Communication commune, Données communes, Modularité, Structure, Documentation.

La standardisation a ouvert la porte vers la possibilité de l'utilisation d'une base unifiée pour la spécification et l'évaluation de la qualité du logiciel. Les deux modèles standards ont été adaptés pour des produits ou des paradigmes logiciels spécifiques. La supériorité numérique des propositions basée sur le modèle ISO/IEC 9126 [Azu04, Zei07, Kum09, Beh09, Rad11, Kum12, Mit13] est due, à notre avis, seulement à l'ancienneté de ce modèle par rapport au modèle ISO/IEC 25010 [Lew10, Her10, Bau12]. Nous soulignons certaines propositions dans les paragraphes suivants.

Behkamal et *al.* [Beh09] traitent l'évaluation de la qualité des applications *B2B*. Le modèle ISO/IEC 9126 a été choisi comme une base pour leur proposition. Ensuite, ils ont identifié les caractéristiques de ces applications. En fait, les auteurs ont remarqué que les applications B2B sont toujours des applications exploitées à travers des sites web. Les caractéristiques des applications web ont été donc identifiées dans cette étape. Après la pondération de différentes caractéristiques de ces applications et l'élimination de caractéristiques existantes déjà dans le modèle standard, quatre caractéristiques essentielles pour les application B2B n'apparaissant pas dans le modèle ISO/IEC 9126 ont été déterminées : la traçabilité *(la capacité du logiciel d'explorer l'exactitude de transformation des informations au cours d'un processus)*, la disponibilité *(le degré de possibilité de l'utilisation du logiciel quand il est nécessaire)*, la navigabilité *(le degré d'accès aux informations en terme de rapidité et d'efficacité)* et la personnalisabilité *(la capacité du logiciel d'augmenter la satisfaction de l'utilisateur en accord avec ses besoins).* Naturellement, le modèle ISO/IEC 9126 doit être étendu pour couvrir ces caractéristiques. Les auteurs [Beh09] considèrent que le modèle ISO/IEC 9126 est complet dans son premier niveau. En conséquence, les caractéristiques des applications B2B déterminées ont été ajoutées dans le deuxième niveau (le niveau de sous-caractéristiques). Ainsi, la traçabilité a été considérée comme une sous-caractéristique de fonctionnalité, la disponibilité a été insérée sous la fiabilité et les deux autres caractéristiques (la navigabilité et la personnalisabilité) ont été ajoutées à l'utilisabilité. Des degrés d'importance ont été affectés pour les caractéristiques et les sous-caractéristiques via un sondage dans lequel vingt spécialistes participent.

Kumar et ses collègues ont tenté de développer un modèle de qualité pour les logiciels orientés-aspects dans deux reprises [Kum09, Kum12]. La première fois

[Kum09], le modèle généré (appelé *AOSQUAMO Model*) est basé sur le modèle ISO/IEC 9126. Par rapport au modèle standard, ce modèle à été étendu par quatre sous-caractéristiques : *la réutilisabilité, la complexité, la réductibilité de code et la modularité.* Ces sous-caractéristiques sont ajoutées respectivement aux caractéristiques suivantes : *fonctionnalité, utilisabilité, efficacité et maintenabilité.* Ensuite, Kumar [Kum12] a enrichi ce modèle en intégrant de nouvelles caractéristiques de la programmation orientée-aspects afin de développer le modèle AOSQ (pour *Aspect-Oriented Software Quality*). En fait, les caractéristiques recherchées derrières le paradigme aspect sont : *l'extensibilité, la durabilité, la stabilité de conception et la configurabilité.* Ces caractéristiques de la programmation orientée-aspects ont été ajoutées au deuxième niveau du modèle précédent (le niveau de sous-caractéristiques). En outre, le modèle proposé (AOSQ) a été étendu par rapport au modèle précédent au niveau de caractéristiques par l'*évolvabilité.* Cette caractéristique regroupe les quatre sous-caractéristiques précédentes.

Le tableau 1.5 donne une vue unifiée de trois modèles de qualité : ISO/IEC 9126 [ISO01], de Behkamal et *al.* [Beh09], AOSQUAMO [Kum09] et AOSQ [Kum12].

Tableau 1.5 : Une vue unifiée de modèles de qualité présentés et dérivés du modèle ISO/IEC 9126.

Caractéristiques	Sous-caractéristiques	Le modèle ISO/IEC 9126	Le modèle de Behkamal et *al.*	Le modèle AOSQUAMO	Le modèle AOSQ
Fonctionnalité	Pertinence	✓	✓	✓	✓
	Précision	✓	✓	✓	✓
	Interopérabilité	✓	✓	✓	✓
	Sécurité	✓	✓	✓	✓
	Réutilisabilité			✓	✓
	Traçabilité		✓		

Fiabilité	Tolérance aux fautes	✓	✓	✓	✓
	Maturité	✓	✓	✓	✓
	Recouverabilité	✓	✓	✓	✓
	Disponibilité		✓		
Utilisabilité	Capacité de compréhension	✓	✓	✓	✓
	Capacité d'apprentissage	✓	✓	✓	✓
	Capacité de fonctionner	✓	✓	✓	✓
	Navigabilité		✓		
	Personalisabilité		✓		
	Complexité			✓	✓
Efficacité	Comportement temporel	✓	✓	✓	✓
	Utilisation des ressources	✓	✓	✓	✓
	Réductibilité de code			✓	✓
Maintenabilité	Analysabilité	✓	✓	✓	✓
	Changeabilité	✓	✓	✓	✓
	Stabilité	✓	✓	✓	✓
	Testabilité	✓	✓	✓	✓
	Modularité			✓	✓
	Adaptabilité	✓	✓	✓	✓
	Installabilité	✓	✓	✓	✓

Portabilité	Conformité	✓	✓	✓	✓
	Remplaçabilité	✓	✓	✓	✓
Évolvabilité	Extensibilité				✓
	Durabilité				✓
	Stabilité de conception				✓
	Configurabilité				✓

On remarque clairement que les différents travaux partagent la même démarche : l'identification de différentes caractéristiques du produit ciblé puis l'insertion de caractéristiques identifiées dans le modèle. Cependant, on peut remarquer que certaines caractéristiques ajoutées (comme la complexité, l'extensibilité et la stabilité de conception) ne représentent pas des caractéristiques de qualité exhaustives pour des produits logiciels spécifiques. En plus, la plupart des travaux ne donnent pas une réponse pour la question pertinente suivante : *quel est le critère utilisé pour décider le niveau adéquat pour la caractéristique à ajouter ?* En d'autre terme, pourquoi ajoute-t-on la réutilisabilité, l'extensibilité, la navigabilité, …etc. comme des sous-caractéristiques alors que l'évolvabilité a été ajouté comme une caractéristique ? A notre avis, donner un critère clair et précis simplifie l'extensibilité du modèle proposé.

Comme un exemple de l'utilisation du modèle ISO/IEC 25010, nous citons ici le travail de Herrera et *al.* [Her10]. C'est la qualité en utilisation des portails web (*Web Portal*) qui a été étudiée dans ce travail. Un portail web est une architecture de sites web qui propose un seul point d'accès à une large gamme d'informations, des applications et des services. En effet, la qualité d'un portail web permet l'attraction des nouveaux utilisateurs et le maintien des anciens. Ainsi, il est important de déterminer la qualité des portails web telle qu'elle est vue par les utilisateurs. En effet, une adaptation du modèle de qualité en utilisation a été proposée afin de prendre en compte les caractéristiques d'une telle architecture des sites web. Les caractéristiques des portails web ont été identifiées à partir d'une revue de la littérature spécialisée. Ensuite, la liste des caractéristiques et sous-caractéristiques du modèle de qualité en utilisation ISO/IEC 25010 a été mise à jour à base de caractéristiques de portails web identifiées précédemment. Les mises à jour prennent la suppression de certaines sous-caractéristiques du ISO/IEC 25010 jugées non importantes, l'ajout de nouvelles sous-caractéristiques et l'adaptation de toutes les définitions proposées dans le modèle standard au domaine cible. En fait, le modèle proposé (appelé *QiUWeP*) maintient les trois caractéristiques constituant le modèle standard : *l'utilisabilité, la sécurité et la flexibilité.* Les sous caractéristiques de l'utilisabilité du modèle ISO/IEC 25010 (*l'effectivité, l'efficacité et la satisfaction*) ont été maintenues avec le changement des composantes de la satisfaction. Cette sous-caractéristique est devenue composée de *la simplicité de l'utilisation, l'expérience, la qualité d'interaction, la qualité de transaction et le sens de communauté*. Les sous-caractéristiques de la sécurité connues dans le modèle standard ont été remplacées par *le risque de sécurité personnelle* et *le risque de dommages économiques*. Le modèle *QiUWeP* spécifie deux sous-caractéristiques pour la flexibilité : l'accessibilité (spécifiée déjà dans le modèle de la qualité en utilisation ISO/IEC 25010) et la personnalisation. Le tableau 1.6 présente le modèle *QiUWeP* en comparant l'existence de sous-caractéristiques au sein du modèle standard.

Tableau 1.6 : Le modèle de la qualité en utilisation *QiUWeP*.

Les caractéristiques	Les sous-caractéristiques	Existence dans le modèle standard
L'utilisabilité	L'effectivité	Existe
	L'efficacité	Existe
	La satisfaction (composée de : *la simplicité de l'utilisation, l'expérience, la qualité d'interaction, la qualité de transaction et le sens de communauté*)	La sous-caractéristique existe mais ses composantes ont été changées
La sécurité	Le risque de sécurité personnelle	N'existe pas
	Le risque de dommages économiques	N'existe pas
La flexibilité	La personnalisation	N'existe pas
	L'accessibilité	Existe

Nous croyons que Herrera et *al.* [Her10] n'ont ciblé qu'une vue restrictive de la qualité des portails web qui consiste à celle de l'utilisateur. En effet, les différentes autres vues ont été omises.

Notons que les différentes propositions liées à la qualité des systèmes multi-agents, notre domaine de recherche, ne sont pas présentées dans cette section. Nous avons consacré le chapitre 03 pour cet objectif après la présentation de différents concepts relatifs à ce domaine dans le chapitre 02.

6. Mesure de la qualité logicielle

"*Mesure ce qui est mesurable, et rend mesurable ce qui ne peut être mesuré*" est un principe fondé par Galilée (1564-1642) et qui représente l'une des maximes de la science moderne. En conséquence, l'application de mesures sur la qualité logicielle est fondamentale. Cet avis est soutenu par la nature ambigüe de la qualité logicielle et la subjectivité éventuelle de différents acteurs (les utilisateurs, les concepteurs, les développeurs,…etc.). Donc, il est important, en plus de la spécification de la qualité du logiciel, de la mesurer. Le standard ISO/IEC 9126 [ISO01] propose la terminologie suivante :

- **Mesurer :** est l'affectation d'une valeur (qui peut être numérique ou une catégorie) d'une échelle à un attribut d'une entité ;
- **Mesure :** c'est la valeur affecté à un attribut par une opération de mesurage ;
- **Métrique :** est l'échelle de mesures et la méthode de mesurage utilisées pour affecter une mesure à un attribut d'une entité.

Malgré que le mesurage soit considéré généralement comme une activité non-productive pouvant perturber en plus le rythme du développement, cette opération offre plusieurs avantages [Sur14] :

- Une communication précise et effective ;
- Un bon moyen de contrôle et de supervision des projets ;
- L'identification rapide des problèmes et de leurs solutions ;
- Une base solide pour la prise des décisions rationnelles et justifiées.

Plusieurs métriques ont été proposées dans le domaine du génie logiciel. En fait, la plupart des modèles cités dans cette thèse proposent des métriques pour la mesure de certains attributs du produit logiciel. Par exemple, le standard ISO/IEC 9126 propose plus de deux cents mesures pour la qualité interne et la qualité externe [ISO03a, ISO03b]. En plus, plusieurs mesures ont été proposées sans tenir compte de la proposition d'un modèle de qualité. Des métriques célèbres comme la taille d'un logiciel en nombre de lignes de code (LOC), la complexité cyclomatique [McC76] ou la taille en points de fonctions (FP) entrent dans cette catégorie. En conséquence, le domaine des métriques des logiciels a connu de plus en plus d'expansion. En fait, plusieurs ouvrages ont été dédiés à ce domaine [Kan02, Lai06, Jon08].

Le mesurage est, généralement, encadré par un processus de préparation et d'exécution afin d'obtenir des bonnes mesures. En fait, la recherche d'améliorer les méthodes de mesure et l'établissement de standards dans ce domaine font la naissance de la métrologie (la séance des mesures). Des bonnes mesures se caractérisent par la fiabilité et la validité [Kan02]. La fiabilité désigne la consistance des valeurs obtenues en utilisant la même méthode de mesurage. Par contre, la validité désigne la consistance entre la valeur empirique obtenue et le sens réel de l'attribut considéré. Evidemment, la preuve de la validité des mesures pour un produit abstrait (comme le cas des logiciels) est extrêmement difficile. L'utilisation des techniques expérimentales et statistiques est la solution pour confirmer ou infirmer la validité des mesures. En fait, Kabaili et *al*. [Kab02] ont pu, en utilisant une étude expérimentale empirique, de contredire l'hypothèse de corrélation de la cohésion et le couplage déjà connue et acceptée par la communauté du génie logiciel.

A la fin de cette section, il est indispensable d'évoquer le chevauchement entre le mesurage et l'évaluation de la qualité. En fait, la relation entre ces deux notions est vue différemment par chaque spécialiste. Parfois l'évaluation est considérée avec le contrôle et la prédiction comme objectifs du mesurage [Bou06]. Par contre Suryn [Sur14] considère l'évaluation de la qualité comme une tâche complexe qui englobe plusieurs parties : un modèle de qualité, une méthode d'évaluation, des mesures et des outils de soutien. Nous adoptons le point de vue de standard ISO/IEC 9126 [ISO01] qui considère l'évaluation de la qualité comme une inspection systématique au produit afin de décider son niveau de qualité. On peut conclure que l'évaluation est plus générique que le mesurage. En conséquence, les deux concepts sont utilisés de façon interchangeable dans cette thèse.

7. Conclusion

La qualité est un besoin fondamental du produit logiciel. Ce besoin englobe la diversité des points de vue d'acteurs intervenant durant le développement des logiciels, la diversité des niveaux d'abstraction des produits logiciels et la diversité des propriétés essentielles pour chaque type de logiciels. En conséquence, on fait recours à la modélisation de ce concept afin de le comprendre et l'évaluer. Plusieurs modèles ont été proposés dans ce domaine. En plus, des tendances actuelles visent la standardisation des modèles de qualité afin d'offrir une vue unifiée de ce concept. D'un autre coté, des recherches ciblent la modélisation de la qualité de certains produits logiciels spécifiques. Notre problématique consiste à l'étude de la qualité des systèmes multi-agents. Après avoir présenté le premier volet de notre problématique dans ce chapitre, nous allons présenter les concepts de deuxième volet dans le chapitre suivant. Dans le troisième chapitre, on va aboutir à la présentation de fusionnement de ces deux volets à travers la présentation des travaux spécifiques à la qualité des systèmes multi-agents.

Chapitre 02

Génie Logiciel Orienté-Agent

« Le tout est plus grand que la somme des parties »

1. Introduction

Le paradigme multi-agents est, actuellement, utilisé dans de nombreux domaines de recherches portant sur le développement des systèmes complexes de nature distribuée. Le domaine multi-agents est considéré comme étant l'évolution de plusieurs domaines initialement indépendants (comme génie logiciel, l'intelligence artificielle et les systèmes distribués) vers la même perspective : *le développement des systèmes intelligents et distribués*. En effet, ce paradigme n'a pas connu seulement une diversité dans les domaines de son application mais aussi dans les problématiques qu'il aborde. En fait, les problématiques traitées par ce domaine peuvent être remontées au génie logiciel (comme les problèmes de conception, d'implémentation et de vérification des systèmes multi-agents (SMA) [Cia01]), aux systèmes distribués (comme les problèmes d'interaction et de coordination [Wei99, Yok12]) ou à l'intelligence artificielle (comme les problèmes de raisonnement, d'apprentissage et de planification des agents [Sto00, Moh14]). En plus, le spectre d'application des SMA est relativement large. L'objectif de ce chapitre est la présentation des SMA en tant que paradigme de développement logiciel. Cependant, quelques concepts de base doivent être présentés dans la première partie afin de donner un aperçu général sur les SMA.

2. Concepts de base

Comme nous l'avions signalé auparavant, le domaine des SMA est d'origine pluridisciplinaires et d'applications diverses. En conséquence, faire une synthèse aux concepts liés aux SMA est une tâche difficile à cause de la multitude et la dispersion de ces derniers. Plusieurs propositions essaient de représenter tous ces concepts dans une vue globale [Bey09, Dem01, Tra07]. Dans son approche appelée *VOYELLES*, Demazeau [Dem01] décrit les SMA selon quatre façades : *Agent*, *Environnement*, *Interaction*, *Organisation*. Nous adoptons cette approche pour la présentation des concepts de base des SMA.

2.1. Les agents

Bien que le concept *agent* représente la pierre angulaire dans les SMA, sa définition ne fait pas l'unanimité. Dans l'un des papiers classiques de la littérature spécialisée, Franklin et Graesser [Fra96] présentent plusieurs définitions des agents afin de tirer les caractéristiques de ces derniers. Cependant, les propriétés qui font de l'agent un nouveau paradigme logiciel sont toujours des sources de débats. Nous optons pour la proposition de Wooldridge et Jennings [Woo95]. Ces derniers proposent deux notions d'agent distinguées: la notion d'agent *faible* et la notion d'agent *fort* [Woo95].

La notion d'agent faible est définit comme *une entité physique ou logicielle autonome, réactive, proactive et sociable* [Woo95]. Les auteurs considèrent l'autonomie comme la propriété la plus importante lors de la définition d'un agent. Par l'autonomie, ils désignent la capacité de l'agent de fixer ses objectifs et de contrôler ses comportements.

La réactivité, la pro-activité et la sociabilité déterminent la flexibilité de l'agent [Woo95]. Ces concepts sont définis comme suite [Woo95]:

- **La réactivité :** la capacité de l'agent de percevoir son environnement et répondre aux différents changements apparus dans le temps adéquat.
- **La pro-activité :** la capacité de l'agent de prendre l'initiative en démontrant des comportements orientés objectifs.
- **La sociabilité :** la capacité de l'agent d'interagir avec les autres agents en utilisant un langage de communication.

En fonction des propriétés précédentes, plusieurs exemples conventionnels peuvent être considérés comme des *agents*. Les systèmes de contrôle, les processus exécutant en arrière plan (*software daemon*) et les *softbots* (*software robots*) sont des exemples des agents faibles [Woo95, Woo02a, Woo02b].

Il semble important de noter que Wooldridge [Woo02a, woo02b] a enrichit les caractéristiques précédentes par une autre propriété ; il s'agit du caractère *situé* des agents. En effet, un agent est une entité située dans un environnent. Nous discutons l'environnement des agents ultérieurement dans ce chapitre.

Par contre, aux agents faibles, un agent fort est une entité conçue ou implémentée en utilisant, outre les propriétés citées précédemment, des concepts appliqués généralement sur les êtres humains. Ainsi, ces agents sont spécifiés par des notions mentales comme : les connaissances, les croyances, les intentions et les obligations [Woo95]. Le concept d'agent fort est dérivé des études de l'IA. En fait, McCarthy

[McC87] note que l'utilisation des notions mentales offre une description de haut-niveau d'abstraction aux comportements des agents.

La nature vague d'un agent se reflète sur la spécification de ces propriétés. En effet, les propriétés d'un agent sont aussi discutables que la définition de l'agent lui-même. En conséquence, il n'est pas étrange de trouver dans la littérature spécialisée une contradiction claire lors la définition des propriétés d'un agent. A titre d'exemple, Alonso et *al* [Alo08] présente la *mobilité* comme une caractéristique importante d'un agent. Cependant, Beydoun et *al.* [Bey09] la considère comme une caractéristique marginale.

A notre avis, il y a des propriétés essentielles pour tous les agents. C'est à cause de ces propriétés que l'agent soit connu par rapport aux concepts similaires tels que les objets ou les systèmes experts. En plus, un agent peut avoir d'autres propriétés optionnelles pour atteindre ses objectifs dans certains contextes et applications.

Luck et d'Inverno [Luc95] font la distinction entre les agents et les agents autonomes. Un agent est associé aux états mentaux. Particulièrement, un agent est un objet avec un but. Le but représente l'état de l'environnement où l'agent cherche à satisfaire. Par contre, les agents autonomes se caractérisent par la motivation. Malgré l'intérêt apporté par ce travail en formalisant les concepts *agent* et *autonomie* en utilisant le langage Z, nous pensons que la spécification des propriétés des agents est loin d'être acceptée par la communauté des chercheurs du domaine. En fait, plusieurs travaux considèrent l'autonomie comme la propriété principale des agents [Fer95, Fra96, Woo95, Woo02a, Woo02b]. A notre avis, il est inadéquat de distinguer les agents des agents autonomes parce que toute agent doit être autonome. En conséquence, l'autonomie c'est une caractéristique essentielle des agents.

L'autonomie ne signifie pas seulement la capacité de l'agent de contrôler son comportement mais aussi de contrôler ses ressources [Fer95]. Le contrôle du comportement désigne que l'agent ne soit pas dirigé par des *commandes* venant de l'extérieur mais par des *tendances*. En plus, l'existence de l'agent autonome est propre et indépendant des autres agents [Arl04].

Dans certaines versions de la définition de l'agent proposé par Wooldridge [Woo02a, woo02b], l'environnement prend une place importante dans les caractéristiques des agents. En fait, c'est l'existence de l'agent dans un environnement qui fait la différence entre les agents et les systèmes experts [Woo02b]. Un agent *situé* dans un environnement peut percevoir et agir dans ce dernier. Dans le cas des agents physiques, l'environnement représente l'espace empirique dans lequel l'agent vit [Wey07]. Par contre, l'espace mémoire partagé entre les agents est l'environnement des agents logiciels. Malgré l'importance de l'environnement dans un SMA, les agents appelés des agents purement communiquant peuvent vivre sans environnement [Fer95].

Le domaine des SMA est issu de l'intelligence artificielle distribuée (IAD). En conséquence, les agents sont souvent caractérisés par l'intelligence. En accord avec Wooldridge [Woo02b], la flexibilité représente la caractéristique principale de l'intelligence. Un agent intelligent est un agent doté de capacité de flexibilité. Wooldridge [Woo02b] décrit la flexibilité par la réactivité, la pro-activité et la sociabilité.

Comme nous l'avions noté auparavant, les domaines d'applications des SMA sont très divers. En effet, certaines applications exigent des caractéristiques optionnelles afin d'atteindre leurs objectifs. Par exemple, les agents exploités dans l'*Internet* ont la capacité de se déplacer entre les machines afin d'exécuter leurs requêtes. La mobilité est l'une des caractéristiques de ce type d'agents. La figure 2.1 présente les propriétés des agents selon leurs types.

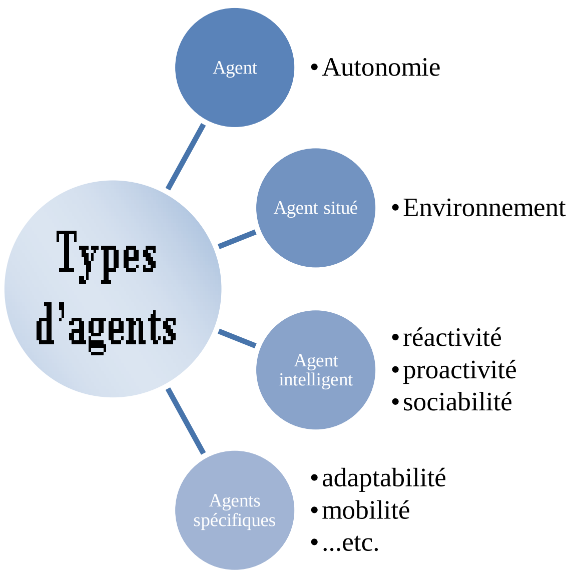


Figure 2.1 : Les propriétés des agents selon leurs types.

Les différentes propriétés des agents sont souvent utilisées comme critères de classification. Ainsi, on trouve dans la littérature des classifications basées sur la mobilité ou l'adaptabilité des agents. En plus, Ferber [Fer95] propose l'environnement comme une base pour classifier les agents. On distingue donc les agents situés et les agents communicants. Cependant, la classification la plus répandue est celle basé sur le degré d'intelligence de l'agent [Sav03] : les agents cognitifs, les agents réactifs et les agents hybrides.

2.2. L'environnement

Par environnement on entend, comme Ferber [Fer 95], les objets passifs composant le monde dans lequel évolue l'agent (exemple : base de données, variables descriptives du système..). Bien que l'environnement de l'agent soit composé d'un ensemble des objets passifs, l'environnement lui-même est considéré comme une entité active et dynamique [Gue12]. En effet, l'environnement a été modélisé par un ou plusieurs agents depuis les premiers travaux dans ce domaine [Bra87, Bou92].

Deux difficultés compliquent l'étude de l'environnement des SMA [Wey04] : la confusion produite à cause de la multitude de définitions de ce concept et le caractère implicite de ses fonctionnalités. La première raison de cette difficulté est engendrée par la confusion entre l'environnement en tant que concept logique et en tant qu'infrastructure d'exécution. En effet, l'environnement peut être vu comme l'infrastructure matérielle (espace mémoire, processeur, réseau, …etc.) ou logicielle (système d'exploitation, middleware, …etc.) nécessaire à l'exécution des agents. En plus, l'environnement représente la modélisation du problème traité dans laquelle les agents existent et évoluent. A titre d'exemple, un environnement de fourmis représente l'espace dans lequel les fourmis vivent avec une modélisation de différents objets de cet espace (obstacle, nourriture, …etc.). Un modèle d'environnement en trois couches (environnement d'application, plateforme d'exécution et infrastructure physique) à été proposé par Weynes et *al.* pour remédier ce problème [Wey04] (Figure 2.2).

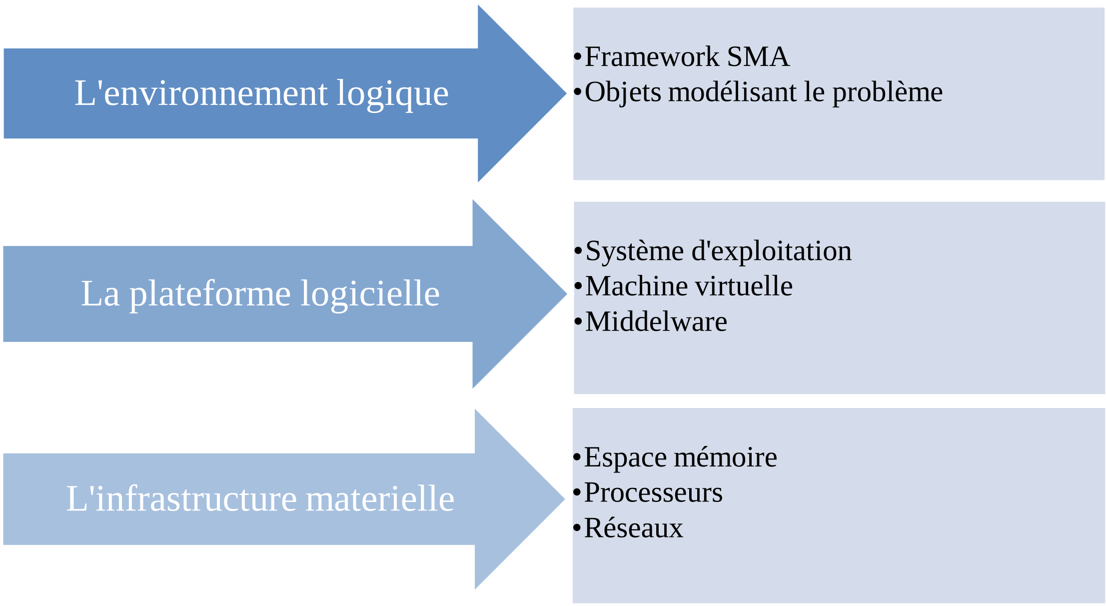


Figure 2.2 : L’architecture en couches d’un environnement (inspirée de [Wey04]).

Ferber et ses collègues [Fer05] proposent un modèle où l’environnement représente l’espace physique et le concept social de SMA (appelé AGRE). Basé sur le célèbre modèle AGR (*Agent-Groupe-Rôle*) [Fer98], le modèle AGRE introduit des nouveaux concepts comme : l’espace, l’univers et la modalité. Un espace peut être une région dans le cas d’un environnement physique ou un groupe dans le cas d’un espace social. Un univers est un ensemble d’espaces de même type. L’existence d’un agent dans un espace se manifeste dans une modalité. Cette dernière peut être représentée par un corps dans l’espace physique ou un rôle dans l’espace social. Ce modèle a été étendu par [Bar07] pour supporter des concepts sociaux plus riches comme le pouvoir, les normes et les relations de dépendance.

La deuxième raison de la difficulté d’étude de l’environnement signifie que l’environnement n’a pas été traité comme une première classe. On entend par une entité de première classe, un module logiciel indépendant qui fournit une abstraction (ou un mécanisme de masquage d’information) permettant le changement de son implémentation sans modifier les autres modules [Wey04]. En effet, l’environnement a été vu dans la plupart des cas comme une partie implicite intégrée aux SMA de manière *ad-hoc* [Gal09]. Malgré que l’importance de la représentation de l’environnement en tant que première classe ait été reconnue pour les systèmes modélisant les composantes spatiales, elle est négligée pour les autres types de SMA

[Tra07]. Cependant, des travaux récents mettent l'accent sur l'importance de cette présentation pour tous les types des SMA [Tra07, Wey07, Bey09].

En plus de la distinction entre l'environnement en tant qu'infrastructure d'exécution et l'environnement en tant que concept logique citée auparavant, on peut distinguer les environnements selon plusieurs autres propriétés. Selon sa distribution, un environnement peut être un système monolithique centralisé comme il peut être un système distribué [Fer99]. Dans ce contexte, Russell et Norvig [Rus03] proposent plusieurs propriétés pour classifier les environnements des SMA:

- **L'accessibilité :** dans un environnement accessible les agents peuvent accéder à son état intégral. Par contre, la portée des actions de l'agent et de sa perception est locale dans un environnement inaccessible.
- **Le déterminisme :** l'état d'un environnement déterministe est lié seulement à son état précédent et à l'action. Dans un environnement indéterministe, les résultats de la même action et dans le même contexte peuvent être différents.
- **Le dynamisme :** un changement de l'état dans un environnement statique se produit exclusivement par les actions des agents. Cependant, l'état d'un environnement dynamique peut être modifié sans l'intervention des agents.
- **La continuité :** si le nombre de perceptions et d'actions possible de l'agent dans un environnement est illimité, on dit que l'environnement est continu. Sinon, on le considère discret.

2.3. L'organisation

Les SMA représentent une société d'agents. Il est donc évident d'étudier cette société en deux dimensions différentes : le niveau micro et le niveau macro. Par le niveau micro on entend l'architecture interne de l'agent avec tous les mécanismes qui lui permettent de raisonner et influer sur son environnement. Par contre, le niveau macro représente la dimension sociale des SMA. En fait, les études ciblant cette dimension sont souvent inspirées de métaphores sociales. Le rôle de l'informatique est de formaliser les études inspirés afin de permettre leurs utilisation sur machine [Tra07].

Afin de définir l'organisation dans le contexte des SMA, Ferber [Fer95] adopte la définition du sociologue Edgar Morin [Mor77]. Ainsi, une organisation est « *un agencement de relations entre composants ou individus qui produit une unité, ou système, dotée de qualités inconnues au niveau des composants ou individus*. » [Mor77]. On distingue deux types d'organisation [Fer95]: l'organisation structurelle et l'organisation concrète. L'organisation structurelle représente les caractéristiques abstraites d'une organisation. Elle est composée d'un ensemble de classes d'agents (dotés de leurs rôles) avec les relations abstraits existants entre ces rôles. Par contre, une organisation concrète représente une instance de l'organisation structurelle.

Partant du fait que l'organisation est une approche de partitionnement d'un système, on doit considérer que chaque partition est un contexte d'interaction des agents qui la composent. Chaque agent connaît les autres agents de sa partition et il peut communiquer librement avec eux. Cependant, l'étude de l'organisation d'un SMA considère les agents comme des boites noires. En fait, les mécanismes de raisonnement des agents et leurs structures internes restent masqués dans le niveau organisationnel. En plus, le niveau organisationnel ne décrit pas le comportement des agents (le *comment*) mais il impose seulement la structure dans les schémas d'activité des agents (le *quoi*) [Fer05]. Nous pensons qu'il est important de distinguer l'aspect comportemental de l'aspect fonctionnel. Bien que le niveau organisationnel ignore l'aspect comportemental, il représente le niveau structurel et le niveau fonctionnel. Le niveau structurel représente les relations entre les individus composant la partition. Par contre, le niveau fonctionnel décrit les activités de ces individu dans cette partition (c'est-à-dire *qui* fait *quoi* mais pas *comment*) [Tra07]. En conséquence, l'étude d'une organisation doit se faire selon trois axes [Fer95] :

- **L'analyse structurelle :** le rôle de cette analyse est d'identifier la forme de l'organisation. En fait, plusieurs formes peuvent caractériser les organisations des SMA. Horling et Lesser [Hor04] proposent un aperçu sur les différents paradigmes organisationnels. Nous citons à titre d'exemple :
    - *La hiérarchie :* dans ce type d'organisations, les agents sont arrangés sous forme d'un arbre (Figure 2.3 – A) où le niveau de hiérarchie de chaque agent représente son niveau de contrôle. Bien

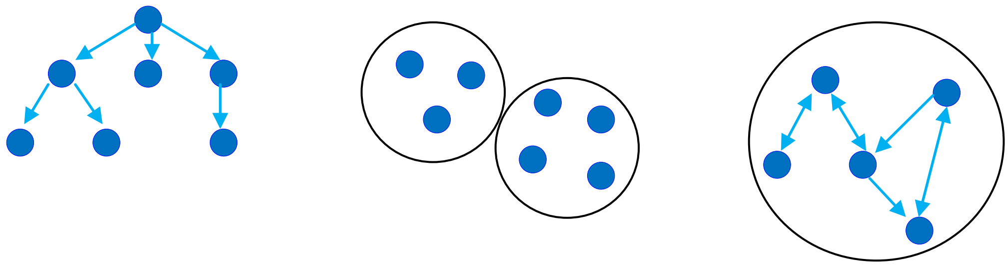
entendu, seuls les agents connectés peuvent communiquer entre eux. Chaque agent communique à son supérieur ses données produites afin d'assurer une vue globale aux niveaux supérieurs. Par contre, un agent contrôle ceux qui lui sont inferieurs à partir d'un flot descendant [Hor04].

- *La coalition :* en observant un système multi-agents, on peut considérer chaque sous-ensemble d'agents comme une coalition. Une coalition se caractérise par l'existence d'un but derrière sa formation et sa durée de vie est courte. En fait, les agents forment la coalition pour satisfaire leurs buts. En conséquence, la coalition se dissout à la fin de l'objectif ou quand les agents qui la forment disparaissent. Comme il est présenté dans la Figure 2.3 – B, la coalition n'a pas une structure hiérarchique. Cependant, ce type d'organisation supporte les relations d'inclusion de chevauchement. Afin d'assurer la communication entre les coalitions, il est possible de trouver un agent représentatif pour chaque coalition [Hor04].
- *Le groupe :* un groupe consiste en un nombre d'agents acceptant de travailler ensemble pour un objectif commun. Malgré qu'un groupe ressemble beaucoup dans sa structure aux coalitions (Figure 2.3 – C), l'existence d'un objectif commun représente l'élément central dans les organisations orientées-groupes. En plus, il apparait que tous les agents coopératifs forment *implicitement* des groupes. Cependant l'existence d'une organisation explicite permet aux agents de raisonner sur des comportements collectifs afin d'augmenter la flexibilité et la fiabilité [Hor04].

- **L'analyse fonctionnelle :** cette analyse détermine les fonctions des membres de l'organisation. Indépendamment de la structure de l'organisation, certains rôles génériques sont communément identifiés [Sec03]:
  - *Le facilitateur (the facilitator) :* le responsable de la communication entre les agents locaux et les agents distants.

- *Le courtier (the broker) :* c'est un agent accessible par tous les agents du système pour assurer les fonctions d'un annuaire. Un agent *courtier* est un agent *facilitateur* avec les fonctions de notification et de supervision.
- *Le médiateur (the mediator) :* un agent *courtier* avec la capacité de coordination entre les agents.

- **Les paramètres de concrétisation :** cette dimension traite la réalisation des organisations SMA en passant de l'organisation structurelle vers l'organisation concrète.



A B C

Figure 2.3 : Exemples de structures organisationnelles [Hor04].

Adopter le concept d'organisation lors de développement des SMA offre plusieurs avantages [Tra07] :

- **Développer des systèmes hétérogènes :** un système multi-agent peut être ouvert et composé d'un ensemble hétérogène d'agents. En effet, la coordination entre les agents pour atteindre les buts communs est devenue difficile. L'organisation offre des moyens explicites pour favoriser la coordination entre les agents. Ce concept peut offrir cette possibilité parce qu'il fait abstraction à la structure et au comportement des agents.
- **Assurer un bon compromis entre l'autonomie et le contrôle :** la caractéristique principale des agents réside dans leurs autonomies. Cependant, cette autonomie peut engendrer des comportements chaotiques si chaque agent essaie de satisfaire seulement ses buts. En revanche, designer un agent superviseur peut limiter gravement l'autonomie des autres agents. Dans ce cas, l'organisation peut être vue

comme un cadre idéal pour assurer un compromis autonomie/contrôle par la spécification des rôles et des responsabilités des agents.

- **Maîtriser la complexité :** en spécifiant l'organisation on désigne principalement « *qui* fait *quoi* ». En conséquence, on peut diminuer les conflits entre les agents et augmenter l'efficacité. En plus, on peut maîtriser la complexité de l'espace de recherche de l'agent en spécifiant les autres éléments qui composent le système et ses relations avec eux.
- **Augmenter la modularité :** du point de vue génie logiciel, l'organisation permet d'accroître la modularité et la réutilisabilité. En fait, une organisation est un schéma générique d'interaction grâce à la représentation abstraite des composants du système (par exemple à l'aide de notion de rôle).

2.4. L'interaction

Les SMA sont, par définition, un ensemble d'agents en interaction. En effet, l'interaction est une façade importante dans la constitution des SMA. On peut même confirmer certains avis qui considèrent l'interaction comme la *raison d'être* des SMA [Sch01]. Cet avis est justifié par la nature limitée des compétences d'un agent isolé. Cependant, l'interaction représente aussi l'origine des problèmes des SMA [Fer95].

L'interaction représente l'influence d'un agent sur le comportement des autres agents à l'aide d'une série d'actions [Fer95]. Cependant, le problème d'interaction ne se résume pas à la simple spécification de mécanismes permettant aux agents d'effectuer cette série d'actions élémentaires (par exemple l'échange des messages). Le problème d'interaction traite l'analyse et la conception des formes d'interaction qui permettent aux agents d'accomplir ses tâches [Fer95]. Les formes proposées dans la littérature peuvent être classifiées selon plusieurs critères [Fer95, Tra07] comme : la compatibilité des buts, la suffisance des ressources et l'existence des compétences [Fer95]. Nous adoptons ici la taxonomie de Ferber [Fer95] qui considère trois formes de l'interaction : la collaboration, la coordination et la négociation. En effet, la coopération est considérée comme la forme la plus générale d'interaction.

La collaboration consiste à la répartition des tâches dans une société d'agents afin d'accomplir un travail commun. En effet, le problème de la collaboration s'intéresse à la question : *« qui doit faire quoi et avec quel moyen, en fonctions des buts et des compétences des agents et des contraintes contextuelles »* [Fer95]. La collaboration nécessite deux mécanismes essentiels : la décomposition des tâches et la répartition de ces dernières sur les différents agents. Le premier mécanisme est effectué souvent manuellement. En revanche, il existe plusieurs techniques pour assurer la répartition des tâches. Ces techniques peuvent être classifiées en deux grandes familles : l'allocation centralisée et l'allocation distribuée. Dans l'allocation centralisée on trouve un agent central qui assure la répartition des tâches sur les autres agents. Cet agent peut être désigné de manière figée (si la structure est hiérarchique) ou être sélectionné par les autres agents (si la structure est égalitaire). Par contre, chaque agent s'occupe individuellement d'obtenir les services dont il a besoin de ses fournisseurs dans l'allocation distribuée. Dans ce cas, l'agent possède une représentation de réseau d'accointances pour pouvoir identifier les capacités des autres agents (l'allocation par réseau d'accointances), ou il fait recours aux appels d'offre.

L'interaction n'est pas conditionnée par l'existence d'un travail commun. En fait, les agents peuvent interagir entre eux et avec leurs propres buts et leurs actions individuelles. On parle dans ce cas de coordination d'actions. Ce type d'interaction est défini par « *l'articulation des actions individuelles accomplies par chacun des agents de manière à ce que l'ensemble aboutisse à un tout cohérent et performant* » [Fer95]. Les agents coordonnent leurs actions pour quatre raisons [Fer95] :

- Le besoin d'informations et de résultats.
- La limite des ressources.
- L'optimisation des coûts.
- La dépendance des buts.

La coordination entre les agents exige l'intervention de différents mécanismes. Bien qu'il existe des mécanismes spécifiques aux SMA, certains mécanismes remontent à des domaines indépendants de SMA. Par exemple, les agents peuvent coordonner entre eux à l'aide de techniques de synchronisation appliquées dans les systèmes distribués. La coordination dans ce cas consiste à décrire précisément l'enchaînement des actions d'agents. Certains travaux utilisent les techniques de planification connues dans le domaine de l'IA pour coordonner les actions des agents.

Dans ce cas, on génère plusieurs plans à partir des actions permettant d'atteindre le but. Puis, un plan va être choisi pour l'exécution. Ce mécanisme souffre de la nécessité de révision dans les environnements dynamiques. En conséquence, la coordination réactive propose d'exploiter la réactivité des agents pour prendre en compte les actions des autres en réponse aux évènements au lieu de raisonner à priori sur des plans. Le dernier mécanisme appliqué pour la coordination des agents (appelé la coordination par réglementation) consiste à utiliser des *lois sociales* pour éliminer les conflits possibles. Le code de la route est un exemple typique de ce mécanisme de coordination [Fer95].

La dernière forme de coopération est la coopération compétitive. En effet, les agents peuvent tomber dans des situations de conflit à cause de la contradiction de leurs buts. Afin de résoudre leurs conflits, les agents peuvent faire recours à l'arbitrage ou à la négociation. L'arbitrage consiste à utiliser des règles de comportement pour limiter les conflits. En effet, la définition des règles sociales est limitée aux agents cognitifs [Fer95]. En plus, nous pensons que ces règles peuvent être considérées comme des contraintes qui limitent l'autonomie des agents. L'application de la négociation est la solution préférée [Fer95]. La négociation consiste à trouver un compromis dans une situation conflictuelle ou changer les croyances de certains agents afin d'imposer l'avis d'un autre agent.

La communication représente la base de tous les types de l'interaction. Par la communication on entend, la transformation d'une information d'un agent à un ou plusieurs autres agents à l'aide d'un langage articulé ou par d'autres codes [Leg03]. En fait, la communication n'est pas spécifique aux SMA parce qu'elle est considérée comme un thème central en informatique depuis longtemps. La synchronisation des processus distribués et l'invocation des méthodes en programmation orientée objet sont des exemples de ce thème [Woo02b]. La spécificité de l'agent consiste à son autonomie qui empêche l'accès à son état interne et au contrôle de son comportement. En conséquence, la communication en SMA consiste à influencer sur le comportement des agents via l'échange des informations et du savoir-faire en préservant leurs autonomies [Fer95, Woo02b]. Deux grandes approches existent pour assurer ce transfert de messages : la communication indirecte et la communication directe. Dans la première approche, un agent « *transfert ses messages* » par la manipulation de son environnement en laissant des signes perçues et interprétées par les autres agents. Par contre, la deuxième approche consiste à la transformation des informations sous forme de messages. Cette approche est basée généralement sur la théorie d'acte du langage développée dans les années soixante du vingtième siècle [Aus62, Sea69]. Selon cette théorie, l'étude de la communication est fondée sur les effets produits sur les destinataires. En autre terme, le langage de communication est formalisé par des pré-conditions et post-conditions attachées aux états mentaux des agents [Tar07]. Le langage FIPA-ACL [FIPA01] est l'exemple célèbre des langages de communication basés sur cette théorie. Les performatifs de ce langage peuvent être regroupés en cinq groupes selon leurs fonctionnalités [Jar06]:

- **Actes pour le passage d'information :** *Inform, Inform-if, Inform-ref, Confirm, Disconfirm.*
- **Actes pour la réquisition d'information :** *Query-if, Query-ref, Subscribe.*
- **Actes pour la négociation :** *Accept-proposal, Cfp, Propose, Reject-proposal.*
- **Actes pour la distribution des tâches :** *Request, Request-when, Request-whenever, Agree, Cancel, Refuse.*
- **Actes pour le traitement des erreurs :** *Failure, Not-understood.*

Bien entendu, l'interaction entre les agents ne se limite pas au simple échange d'un message. L'interaction représente un enchaînement conversationnel combiné de plusieurs messages. La modélisation de cet enchaînement est basée sur l'une de deux approches [Jar06]:

- La construction du plan de la communication se fait par l'agent selon ses connaissances et ses buts. Il n'y a pas une représentation à priori de ce plan, mais l'architecture de l'agent lui permet de mener ses interactions.
- Les plans de communication sont spécifiés à l'avance par des règles appelées un protocole d'interaction. Ces protocoles spécifient pour chaque message reçu un ensemble limité de réponses possibles. Evidement, cette approche diminue la flexibilité de l'agent. Cependant, elle permet aussi de maîtriser la complexité de l'architecture interne des agents.

3. L'ingénierie des SMA

Après avoir présenté dans la partie précédente de ce chapitre les concepts de base sur lesquels les SMA sont fondés, nous présentons dans cette partie l'ingénierie des logiciels basés sur ce paradigme. Il est évident que l'application des techniques et méthodes de génie logiciel est la clé de réussite du développement logiciel. Les techniques et les méthodes appliquées doivent être adéquates aux spécificités du logiciel à développer. En conséquence, les logiciels basés agents nécessitent leurs propres méthodes, méthodologies et techniques de développement. En fait, la particularité des SMA peut être résumée en deux caractéristiques principales. D'une part, un logiciel basé sur le paradigme agent est considéré comme un ensemble *distribué* d'entités *actives* en *interaction* qui fonctionnent éventuellement dans un système *ouvert*. Il est connu que la nature des logiciels dans les paradigmes traditionnels (comme le paradigme objet) considère un logiciel comme un système *monolithique*, *fermé* et composé d'un ensemble d'entités *passives* [Tim06, Luc08]. D'autre part, la pluridisciplinarité des SMA influence sur leurs méthodes de développement. Malgré qu'on considère génie logiciel orienté agent comme une évolution de paradigmes logiciels traditionnels, ce dernier a été influencé par les méthodes existantes dans les domaines d'origine des SMA. En particulier, l'ingénierie des connaissances (le domaine du développement des systèmes à base de connaissances) a un impact clair sur quelques méthodologies de développement multi-agents [Tim06].

Depuis l'apparition du premier aperçu sur les méthodologies de génie logiciel orienté agent à la fin des années quatre-vingt-dix [Mül97], plusieurs ouvrages ont été consacré à la présentation de ce domaine [Ber04, Kir06a, Lin07, Bor09, Ste09, Das10, Cos14]. En effet, la présentation détaillée et exhaustive de ce domaine est hors la portée de notre thèse. Cependant, nous consacrons cette partie à la présentation des grands axes de développement multi-agents. Ainsi, nous commençons par la présentation de quelques modèles des SMA, suivi par un aperçu sur les méthodologies multi-agents. Ensuite, nous focalisons sur les approches d'implémentation des SMA parce que notre objectif est l'étude de la qualité des applications multi-agents.

3.1. La modélisation des SMA

Nous avons abordé dans la première partie de ce chapitre la nature de l'agent et ses propriétés. En fait, c'est la *théorie de l'agent* qui traite ce genre de questions. Par contre, les modèles des agents (ou les *architectures des agents*) ciblent la question de la réalisation des composants logiciels qui peuvent assurer les propriétés des agents [Woo95]. Nous abordons la question de la modélisation des SMA dans cette partie parce que les modèles des SMA représentent la pierre angulaire de beaucoup de travaux en génie logiciel orienté agent.

Le modèle d'agent BDI (pour *Belief-Desire-Intention*) [Rao95] est, sans doute, l'un des modèles les plus référencés dans la littérature spécialisée. Ce modèle a été conçu initialement par Bratman [Bra88] comme une théorie pour la représentation du raisonnement des êtres humains. Ensuite, le modèle a été exploité dans le domaine des agents intelligents compte tenu de sa simplicité. En fait, le modèle, comme son nom l'indique, est basé sur trois concepts : les croyances, les désires et les intentions. Ces derniers représentent des états possibles du monde. Les croyances représentent le monde comme il est vu par l'agent. Bien entendu, l'agent actualise ces croyances en fonction de ses perceptions. Les désires représentent les buts de l'agent. Par contre, les intentions sont un sous-ensemble de désirs et représentent les buts en cours d'achèvement.

Malgré son succès, le modèle BDI n'est qu'un exemple de modèles d'agents. En effet, plusieurs autres modèles ont été proposés pour des objectifs génériques ou pour des applications restreintes. Cependant, Beydoun et *al*. [Bey09] proposent une autre alternative qui consiste à la proposition d'un meta-modèle pour les agents. La méta-modélisation consiste à identifier les concepts utilisés pour décrire un modèle [Cos14]. En fait, le meta-modèle *FAML* proposé par Beydoun et *al.* [Bey09] n'est pas le premier meta-modèle proposé dans le domaine multi-agents [Bey09, Cos14]. Cependant, il a la caractéristique d'être générique. Les premiers meta-modèles ont été attachés à des méthodologies de développement multi-agents spécifiques. Bien entendu, leurs diversités représentent un obstacle lors du partage des résultats entre les groupes.

Le méta-modèle *FAML* a été développé par la combinaison de deux approches d'analyse : l'approche ascendante et l'approche descendante. Durant la phase descendante, les concepts nécessaires pour la modélisation des agents ont été identifiés de la littérature spécialisée. Par contre, la phase ascendante consiste à réviser le modèle initial par la validation de ce dernier par rapport aux autres modèles et méthodologies de développement multi-agents proposées dans la littérature. Cette phase de validation a pour objectif l'identification des éventuels concepts omis ou pour l'assurance de la cohérence des concepts. Ce méta-modèle est proposé comme support durant le développement des SMA. Ainsi, il distingue deux catégories de concepts : des concepts durant le temps d'exécution et des concepts durant le temps de conception.

Durant la conception, la définition d'un agent consiste en un état initial avec une ou plusieurs spécifications de plan (Figure 2.4). Chaque spécification de plan est composée d'un ensemble de spécifications d'action et elle utilise un ensemble de ressources. Les spécifications d'actions peuvent être des actions reliées à l'environnement (appelées *Facet Action Specifications*) ou des actions de communication (appelées *Message Action Specifications*). Par contre, l'agent est considéré, durant son exécution, comme une collection d'états mentaux. Les états mentaux représentent le concept révisé des concepts reliés au modèle BDI (croyances, désires et intention) qui ont été désigné dans la version initiale de ce méta-modèle. Sachant que les états mentaux peuvent être des croyances ou des buts, le méta-modèle FAML désigne pour chaque but un ou plusieurs plans. Un plan est composé d'un ensemble d'actions qui peuvent être des actions reliées à l'environnement ou des actions de communication. Par rapport à l'agent durant la phase de conception, un agent dans son exécution joue un ou plusieurs rôles. Il peut posséder des obligations et ses communications se déroulent au sein des protocoles d'interaction.

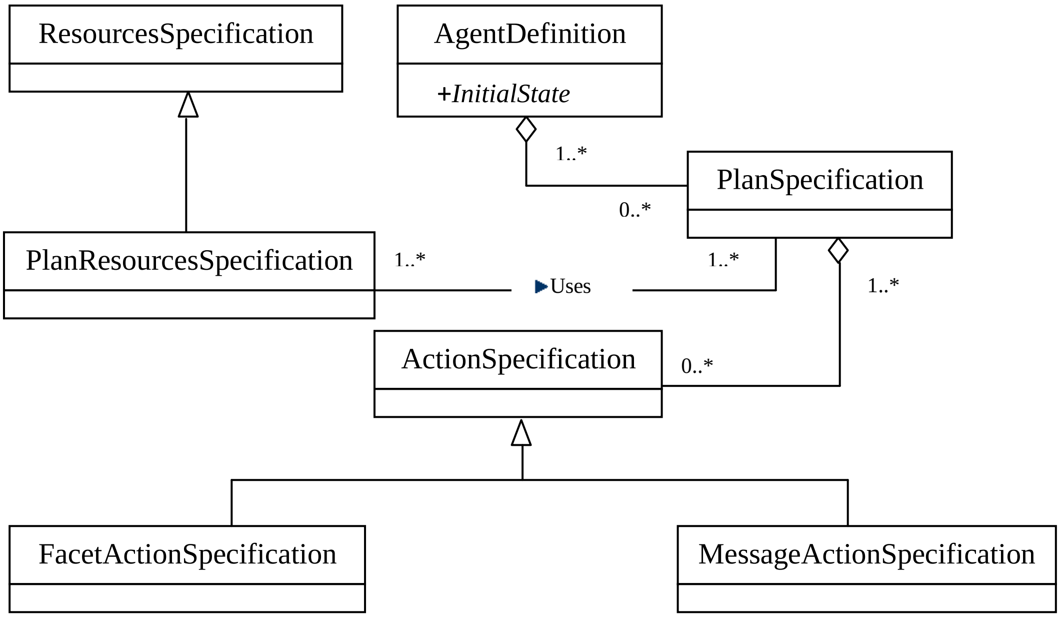


Figure 2.4 : Modèle d'agents durant la phase de conception [Bey09].

Le méta-modèle FAML spécifie également le niveau système selon les deux vues : la conception et le temps d'exécution. Le système représente l'entité centrale d'un SMA. Cette entité est composée d'une organisation et d'un environnement. Une organisation est une collection d'agents pouvant jouer des rôles. Le rôle peut utiliser des ontologies pour offrir des services à l'aide de l'exécution des tâches dont il est responsable. Malgré les points communs entre un SMA en conception et un SMA en exécution selon le méta-modèle FMAL, ce dernier considère l'environnement comme l'entité centrale d'un SMA en exécution.

Contrairement au méta-modèle FAML inspiré de la littérature spécialisé, le méta-modèle MASQ (pour Multi-Agent Systems based on Quadrants) [Tra09] est basé sur de la théorie de la « *vision intégrale* » proposée par Wilber [Wil01] dans les sciences sociales et psychologiques. Illustré par la Figure 2.5, ce méta-modèle est basé sur quatre concepts définis par rapport à deux axes : (Interieur/Exterieur) et (Individuel/Collectif) [Tra09]:

- **L'esprit *(Individuel/Interne)*:** représente l'architecture interne de l'agent. Il englobe les mécanismes de raisonnement de l'agent qui sont protégés contre l'accès d'un autre agent ou de l'environnement.

- **L'objet *(Individuel/Externe)*:** représente une entité de l'environnement. Ce méta-modèle distingue l'esprit de l'agent de son corps. En fait, le corps de l'agent est un objet relié à l'esprit de l'agent. Il lui permet d'interagir avec l'environnement.
- **L'espace brut *(Collectif/Externe)*:** ce concept met en relation les objets de l'environnement.
- **La culture *(Collectif/Interne)*:** modélise la connaissance partagée entre les différents esprits, comme les normes sociales, les ontologies ou les interprétations collectives.

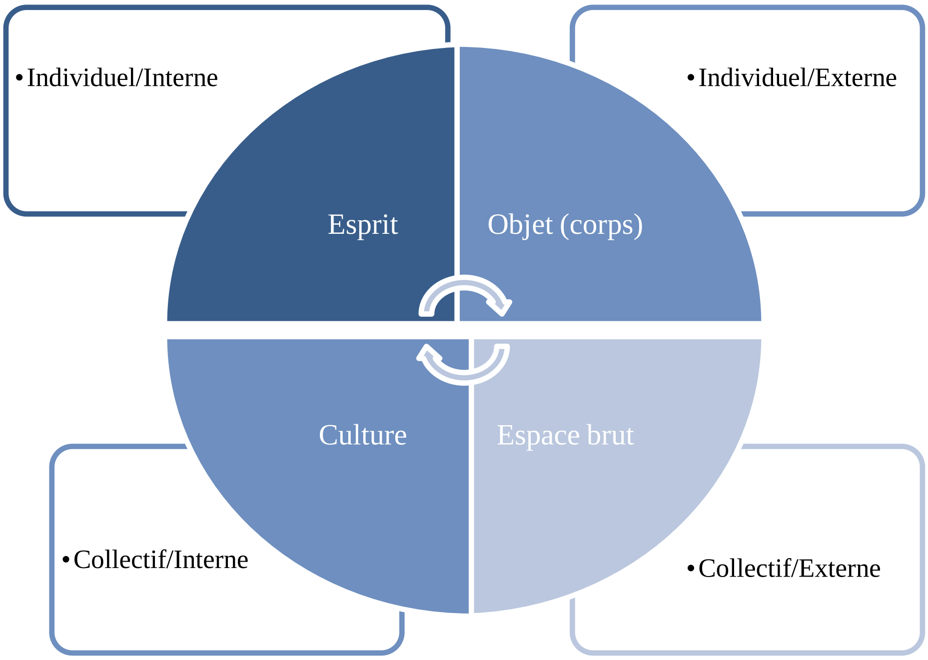


Figure 2.5 : Le méta-modèle MASQ [Tra09].

Bien entendu, ces concepts ne sont pas séparés. Le méta-modèle, en tant que framework générique qui intègre les concepts connus dans le domaine multi-agents (comme l'action, l'environnement, l'organisation,…etc.), met les relations entre ses différents quadrants. Par exemple, un agent raisonne par son esprit et agit dans l'espace brut par son corps [Tra09]. Intuitivement, le principe de ce framework peut être expliqué à travers ce scenario : ***l'esprit*** agit à travers un ***corps*** avec les autres ***objets*** existant dans *l'**espace brut***. La ***culture*** permet l'interprétation collective de l'interaction [Din10].

Le méta-modèle MASQ a été appliqué dans plusieurs applications visant la modélisation et la simulation multi-agents. Ce framework a été implémenté sous forme d'un plugin appelé ABC Lab (pour *Agent-based Business Coordination Lab*) [Din10]. Par ailleurs, Dinu et *al.* [Din10] proposent la formalisation de ce modèle pour limiter le développement ad-hoc des applications basées sur MASQ.

3.2. Les méthodologies de développement des SMA

Une méthodologie est « *un ensemble de méthodes appliquées tout au long du cycle du développement d'un logiciel, ces méthodes étant unifiées par une certaine approche philosophique générale* » [Boo92]. En fait, le terme *méthodologie* est souvent confondu avec le terme *méthode* qui signifie « *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* » [Arl04]. En d'autre terme, une méthode représente une description étape par étape d'une tâche, alors qu'une méthodologie représente un système de méthodes appliquées tout au long de cycle de vie d'un logiciel [Aza07].

Dans la littérature, plusieurs méthodologies le développement des SMA ont été émergées. Malgré que les débuts de ce paradigme aient connu un développement *ad-hoc*, les travaux sur les méthodologies ont atteint maintenant un certain niveau de maturité [Hil08]. On peut trouver dans la littérature spécialisée plusieurs synthèses de méthodologies multi-agents existantes accompagnées souvent par des études comparatives [Ber04, Sel05, Ste09, Cos14]. Bien entendu, il n'existe pas une méthodologie idéale pour le développement des SMA. En plus, la présentation d'une autre étude comparative de ces méthodologies ne semble pas pertinente à notre problématique.

Les méthodologies multi-agents sont souvent classifiées selon leurs domaines d'origines [Gio05, Ber09]. A notre avis, la classification de ces méthodologies ne se limite pas à ce critère. On peut catégoriser les méthodologies orientées-agents selon trois critères :

- **Le domaine d'origine de la méthodologie :** généralement, les méthodologies de développement des SMA sont regroupées en deux grandes familles [Gio05, Ber09] : des méthodologies basées sur le paradigme orienté objet et des méthodologies étendues de l'ingénierie de connaissances. Cette classification reflète la nature des SMA en tant qu'un domaine émergé de la convergence du génie logiciel orienté-objet et de l'intelligence artificielle distribuée. La méthodologie *MaSE (Multiagent Systems Engineering)* [Del99] est un exemple de méthodologies de la première famille. Cette méthodologie considère l'agent comme spécialisation de l'objet. Ainsi, un agent est un objet actif possédant un objectif et interagissant avec les autres agents en utilisant un langage de communication. En conséquence, cette méthodologie utilise des modèles basés sur les modèles standards d'*UML*. Ces modèles sont divisés en sept étapes qui sont organisées à leurs tours en deux phases. La première phase (la phase d'analyse) est constituée de trois étapes : capturer les buts, appliquer les cas d'utilisation et raffiner les rôles. En revanche, la phase de conception est composée de quatre étapes : créer les classes d'agents, construire les conversations, assembler les classes d'agents et concevoir le système.

La deuxième famille des méthodologies multi-agents est celle des extensions des méthodologies utilisées pour le développement des systèmes à base de connaissances. Ces dernières ont l'avantage de fournir des techniques pour la représentation de l'état mental de l'agent. La méthodologie *CoMoMAS* [Gla96] utilise six modèles (le modèle d'agent, le modèle d'expertise, le modèle de tâche, le modèle de coopération, le modèle du système et le modèle de conception) décrits à l'aide du langage *CML* (*Conceptual Modeling Language*).

Malgré que cette classification soit largement acceptée par les chercheurs du domaine, cette dichotomie n'existe pas réellement. Plusieurs méthodologies ont été conçues à l'aide d'une vue hybride qui combine les idées de deux familles. A titre exemple, la méthodologie *MAS-CommonKADS* [Lgl98], considérée comme une extension de méthodologies de développement des systèmes à base de connaissances [Gio05], représente réellement la combinaison de la méthodologie de développement des systèmes à base de connaissances *CommonKADS*

[Sch94] et la méthodologie orientée objet *OOSE* (*Object Oriented Software Engineering*) [Jac92].

Certaines méthodologies multi-agents sont basées sur des principes plus originaux, comme l'application des méthodes formelles [Zhu03, Reg05], l'utilisation de l'ingénierie des besoins [Bre04] ou l'utilisation de l'approche dirigée par les modèles [Pav06, Aza07].

- **Le domaine d'application de la méthodologie :** selon cet axe, les méthodologies multi-agents peuvent être classifiées selon leurs objectifs. On peut distinguer des méthodologies multi-agents génériques comme on peut trouver des méthodologies multi-agents développées pour des domaines d'application spécifiques. Les méthodologies *MaSE* [Del99] et *Gaia* [Woo00] sont des exemples de la première catégorie. Par contre, la méthodologie *ADELFE* [Pic04] est proposée pour le développement des SMA adaptatifs, la méthodologie *RT-MESSAGE* [Jul04] est une extension de la méthodologie *MESSAGE* [Cai01] pour prendre en compte les contraintes temporelles et la méthodologie *MOBMAS* [Tra08] a été conçue pour prendre en compte les spécificités des SMA basés sur les ontologies.
- **L'étendu de la méthodologie :** les méthodologies multi-agents s'articulent généralement sur deux phases : l'analyse et la conception [Cos14]. Cependant des récentes méthodologies ciblent de plus en plus les phases d'implémentation et de déploiement. En conséquence, on peut considérer les phases couvertes par une méthodologie de développement des SMA comme un axe de classification. La méthodologie *Gaia* [Woo00], l'une des premières méthodologies de développement des SMA, se limite aux phases d'analyse et de conception. Comme il a été constaté dans le premier axe de classification (le domaine d'origine de la méthodologie), certaines méthodologies (comme *Tropos* [Bre04]) couvrent même la phase de l'ingénierie des besoins. En plus, la méthodologie *Tropos* [Bre04] assiste les développeurs durant la phase d'implémentations. Les phases post-implémentation, notamment la phase

de test et la phase de déploiement, sont couvertes par quelques méthodologies comme *MASSIVE* [Lin01] et *PASSI* [Cos05]

3.3. L'implémentation des SMA

Afin de garantir le succès de développement des SMA, il est indispensable de *combler le fossé* entre l'analyse et la conception de tels systèmes d'une part et leurs implémentations réelles d'autre part. En fait, le problème de l'implémentation des SMA consiste au développement des langages de programmation et des outils permettant la réalisation des concepts clés connus au domaine multi-agents de manière *efficace* et dans un *framework unifié* [Bor07a]. En fait, la problématique de l'implémentation des SMA est d'un large spectre parce qu'elle couvre en plus de la programmation des SMA, les langages de spécification, les techniques de vérification et les outils de développement de tels systèmes [Bor09]. Dans cette partie on ne s'intéresse qu'à la phase d'implémentation des SMA.

Comme dans le cas des méthodologies de développement des SMA, notre objectif n'est pas l'introduction d'une liste exhaustive des langages de programmation multi-agents. Il existe plusieurs ouvrages spécialisés qui présentent un état de l'art actualisé de cette question [Bor05, Bor07a, Bor09]. Cependant, nous consacrons cette partie à présenter un panorama des approches existantes dans ce domaine. Nous avons donc choisi cinq axes autour desquels nous présentons ces approches. Ces axes représentent seulement des critères de classification. En conséquence, cette classification ne doit pas être considérée comme une étude comparative des approches de développement des SMA. La comparaison des langages de programmation multi-agents a été l'objet de plusieurs études soit en ciblant des plateformes génériques [Ber08, Coa12] ou en spécifiant une catégorie plus restreinte des plateformes [Tob04, Rou14]. Ainsi, les approches de développement des SMA peuvent être positionnées selon les cinq axes suivants :

- **La nature de l'approche :** afin de développer des SMA, plusieurs choix peuvent cohabités : la proposition des langages de programmation, l'extension des langages existants par des bibliothèques ou le développement des plateformes. La première solution consiste à proposer des langages de programmation spécifiques au paradigme agent. Ces langages proposent des primitives permettant la représentation des concepts de haut niveau d'un SMA (comme l'agent, l'interaction,…etc). Les deux langages *AGENT0* et *AgentSpeak(L)* représentent les exemples types de cette approche. *AGENT0* permet la modélisation de l'agent en tant que module qui englobe des états mentaux. Les états mentaux représentent des croyances, des compétences et des capacités de décision formalisées à l'aide de logique modale [Gue01]. Contrairement à cette solution, plusieurs chercheurs proposent l'introduction d'un niveau d'abstraction permettant la construction des SMA par des langages de programmation conventionnels. Ce niveau d'abstraction est implémenté par des bibliothèques de primitives et de structures pour le langage d'origine [Gue01]. La bibliothèque *ALBA* [Dev07] entre dans cette catégorie. *ALBA* est une bibliothèque écrite en *Prolog* pour programmer les agents mobiles. Cette bibliothèque offre les possibilités de la création, de l'exécution, et de la mobilité des agents comme elle permet la communication entre eux. Une bibliothèque représente selon Devèze et *al.* [Dev07] l'infrastructure la plus simple qui permet l'intégration facile des agents dans les systèmes existants. Cependant, plusieurs approches ont été fondées sur le développement des plateformes de développement. Une plateforme multi-agents est une infrastructure agissante en tant que médiateur entre le système d'exploitation et les applications multi-agents. Elle peut être donc vue comme un ensemble de services permettant aux développeurs de créer des agents, de les exécuter et les tester. En plus, une plateforme est vue comme une infrastructure assurant l'interaction entre les agents [Aza07]. La plateforme *JADE (Java Agent DEvelopment framework)* [Bel07] est maintenant la plateforme multi-agents la plus populaire. Ecrite en *Java*, la plateforme *JADE* permet le développement, l'exécution et le débogage des applications multi-agents. Plusieurs raisons ont promu la plateforme *JADE* pour être parmi les plateformes multi-agents les plus utilisées, comme : la compatibilité avec la norme FIPA [FIPA01], la distribution open-source, l'exécution répartie et la possibilité de développer des agents mobiles. Une présentation plus détaillée de cette plateforme sera présentée dans la suite de cette thèse.

- **La dépendance à un modèle :** comme il est cité précédemment, la modélisation représente la pierre angulaire dans plusieurs approches de développement multi-agents. Ainsi, plusieurs approches d'implémentation des systèmes multi-agents sont fondées sur un modèle spécifique des agents. A titre d'exemple, la plateforme *Jadex* [Pok05] est conçue pour le développement des agents cognitifs BDI. En conséquence, un agent implémenté en utilisant la plateforme *Jadex* représente la modélisation des concepts : plans, buts et croyances. La structure de l'agent est représentée à l'aide de langage XML tandis que le corps des plans est décrit en langage *Java*. A l'opposé de la plateforme *Jadex* qui est fondée sur un modèle d'agent conçu indépendamment (le modèle BDI), Guessoum [Gue01] propose un modèle d'agent appelé *DIMA* et une plateforme qui supporte le développement des agents basés sur ce modèle. Un agent *DIMA* est un agent *hybride* implémenté à l'aide de langage *Java*. Une discussion détaillée de cette plateforme est présentée dans les chapitres suivants de cette thèse. En alternatif aux approches basées sur des modèles spécifiques d'agent, il existe des approches indépendantes de modèles d'agents. La plateforme *JADE* [Bel07] est un exemple de cette alternative. En fait, la plateforme *JADE* propose un noyau minimal qu'on peut utiliser pour l'implémentation de plusieurs types d'agents.
- **L'existence d'un concept clé :** la difficulté de la programmation des agents réside dans la multiplicité de notions existantes dans ce paradigme, comme l'organisation, l'environnement, l'interaction, …etc [Gue01]. Face à la difficulté d'unifier toutes ces notions dans une seule infrastructure, les concepteurs choisissent un concept clé pour chaque plateforme. Nous présentons deux exemples de plateformes qui focalisent sur des concepts clés. Le premier exemple est le framework *CArtAgO* [Ric11]. Ce dernier est une infrastructure pour la programmation des mondes computationnels partagés appelés des environnements de travail (*Works Environment*). Cet environnement peut englober des agents implémentés dans des plateformes hétérogènes afin de travailler ensemble dans un SMA unifié. *CArtAgO* est basé sur le modèle conceptuel *A&A (Agents and Artifacts)*. Les environnements de

travail sont programmés comme un ensemble d'*artefact* englobé dans un *workspace*. Un agent considère un artefact une entité de première classe qui représente les ressources et les outils dynamiquement instanciés et partagés pour supporter les activités individuelles et collectives des agents. Contrairement à ce framework qui considère l'environnement comme une entité de première classe, la plateforme *Janus* [Gau08] est un exemple de plateformes de programmation organisationnelles. Basé sur le méta-modèle organisationnel *CRIO*, cette plateforme considère les concepts rôle et organisation comme des entités de première classe. Nous croyons que la maturité de cet axe va participer à la fusion de plusieurs concepts dans une seule plateforme. La plateforme *JaCaMo* [Boi13], à notre avis, entre dans cette perspective. Cette plateforme est basée sur le principe suivant : *les SMA sont basés sur plusieurs concepts tous considérés comme des entités de première classe*. Ainsi, *JaCaMo* est une combinaison de trois plateformes qui traitent des concepts séparés : *Jason* [Bor07b] pour la programmation des agents autonomes, *CArtAgO* [Ric11] pour la programmation des environnements des SMA et *Moise* [Hüb10] pour la programmation des organisations multi-agents.

- **Le paradigme de base :** la programmation orientée-agent est proposée la première fois par Shoham [Sho93]. Naturellement, ce nouveau paradigme de programmation propose une nouvelle vision de la programmation basée sur la modélisation des entités dotées d'états mentaux (des croyances, des buts, des compétences, …etc.). Cependant, ce nouveau paradigme de programmation trouve ses origines principalement dans la programmation logique et la programmation orientée-objet. Cette classification a été adoptée par Bordini et *al.* [Bor05] dans la présentation des plateformes et langages de programmation orientée-agent. Le langage *Jason* [Bor07b] est un exemple de langages de programmation orientée-agent basés sur la programmation logique. Par contre, la plateforme *JADE* [Bel07] est un exemple de la deuxième catégorie de cette classification. Malgré la différence entre ces deux approches concernant le paradigme de base, le principe adopté est le même : *étendre les paradigmes de base par des primitives qui permettent la modélisation des concepts des SMA*. Une approche originale est proposée récemment par Pokahr et *al.* [Pok14]. Elle consiste à la programmation des agents BDI en utilisant seulement le langage *Java*. En conséquence, cette approche permet une implémentation facile des SMA sans avoir besoin d'apprendre les concepts liés au modèle BDI. Ces concepts peuvent être représentés en *Java* grâce aux capacités de méta-données.

- **Le domaine d'application :** les langages de programmation orientée-agent peuvent être des langages génériques ou des langages conçus pour des domaines spécifiques. Bien entendu, le terme « *générique* » ne signifie pas la capacité de développer tous les systèmes. Dans ce contexte, on entend par *généricité* le caractère indépendant de la plateforme ou du langage sur les domaines d'applications. La plupart des langages et plateformes cités auparavant sont conçus pour des domaines d'application génériques. *JADE* [Bel07], *Jadex* [Pok05], *Jason* [Bor07b] sont des exemples de cette catégorie. En utilisant ces plateformes, on peut implémenter des applications diverses malgré les possibles variations de leurs expressivités. Dans le même contexte existent des plateformes conçues pour des applications plus restreintes. Pour implémenter des SMA opérationnels sur des applications mobiles, la plateforme *JaCa-Android* [San11] a été proposée. Cette plateforme représente l'intégration de deux plateformes multi-agents *Jason* [Bor07b] et *CArtAgO* [Ric11] au-dessus de la plateforme *Google Android*.

4. Conclusion

Les systèmes multi-agents représentent un domaine de recherche émergé de la convergence de plusieurs domaines comme : le génie logiciel, les systèmes distribués et l'intelligence artificielle. Adoptant le point de vue du génie logiciel, ce domaine représente un nouveau paradigme de programmation. Ce dernier a atteint un niveau de maturité important notamment sur le plan implémentation (langages et plateformes d'implémentation) [Bor09]. Ainsi, nous choisissons de présenter dans ce chapitre les SMA du point de vue génie logiciel, ou génie logiciel orienté agent (AOSE). Cependant, une présentation claire du génie logiciel orienté agent nécessite l'introduction de concepts sur lesquels les SMA ont été fondés. Ces concepts selon l'approche *Voyelles* [Dem01] sont : l'agent, l'environnement, l'organisation et l'interaction.

Nous croyons qu'il est impossible de donner une présentation exhaustive de travaux menés en génie logiciel orienté agent à cause de gigantesque expansion de ce domaine. En conséquence, nous avons choisi de présenter les travaux pertinents relatifs à notre axe de recherche. Ainsi, la section intitulée « *l'ingénierie des SMA* » a été consacré beaucoup plus à la modélisation des SMA. En fait, les modèles des SMA représentent la fondation du génie logiciel orienté agent. Ensuite, nous avons présenté les méthodologies de développement des SMA en focalisant dans la dernière partie sur la phase de l'implémentation. Vu l'existence de plusieurs études comparatives tant pour les méthodologies de développement que pour les plateformes de développement, nous avons choisi de présenter un panorama de travaux existants selon différents angles. Bien entendu, nous allons détailler les plateformes utilisés dans nos travaux (à savoir la plateforme *JADE* [Bel07] et la plateforme *DIMA* [Gue01]) ultérieurement dans cette thèse.

Nous allons consacrer le chapitre suivant pour cerner en plus le cadre de notre problématique de recherche. En conséquence, les études en relation avec la qualité des SMA vont être présentées.

Chapitre 03

La Qualité des Systèmes Multi-Agents

« Nous sommes comme des nains juchés sur des épaules de géants »

1. Introduction

Le génie logiciel orienté agent a atteint un niveau de maturité considérable. Comme nous l'avons vu au chapitre précédent, ce niveau de maturité peut apparaitre dans la diversité des méthodes, des méthodologies et des plateformes proposées pour faciliter le développement des applications orientées-agent. En conséquence, nous pensons qu'il est important d'étudier la qualité des logiciels implémentés en utilisant ce paradigme. A notre avis, ces études peuvent prendre plusieurs formes comme la spécification de la qualité en prenant en compte les caractéristiques du paradigme agent et la mesure de la qualité des applications basées agents. En fait, plusieurs mesures ont été proposées pour l'évaluation des différents attributs des SMA. On peut trouver des mesures très récentes [Twu14, Sch14, Sha14] comme on peut trouver d'autres qui reviennent aux débuts de l'émergence de ce paradigme [Lee98, Bar99]. Cependant, ces mesures ne sont pas nécessairement proposées dans le cadre de la qualité des SMA. Ce point de vue est justifié par la rareté des études consacrées à la qualité des applications multi-agents. Nous présenterons dans la première partie de ce chapitre les travaux consacrés à la spécification de la qualité des SMA. Ensuite, nous donnerons un aperçu sur les mesures proposées dans ce domaine. Cet état de l'art nous permet de proposer une étude comparative de différentes approches proposées. Nos contributions proposées dans les chapitres suivants sont basées sur le bilan de cette étude comparative.

2. Modèles de la qualité des SMA

Comme nous il a été vu au premier chapitre, le concept « *qualité* » est un concept vague. En conséquence, la spécification des modèles cernant la définition de ce concept est un besoin primordial. En plus, ces modèles doivent être adaptés aux différents paradigmes logiciels. Cependant, le paradigme agent n'a pas connu des travaux profonds dans cet axe. En fait, les modèles de qualité spécifiques aux systèmes multi-agents se caractérisent, en plus de leur rareté, par l'incomplétude ou la spécificité du domaine d'application.

Alonso et *al.* [Alo08] ont remarqué l'absence d'un modèle globale pour l'évaluation de la qualité des logiciels basés sur le paradigme agent. Ainsi, ils ont proposé une série de travaux [Alo08, Alo09, Alo10a] afin de développer un modèle qui satisfait cet objectif. Dans le premier pas de cette série, Alonso et *al.* [Alo08] ont traité la qualité selon la définition de la standard ISO/IEC 9126 [ISO01] (à savoir, caractéristique - sous-caractéristique (attribut) - métrique). En conséquence, les caractéristiques de l'agent ont été spécifiées selon la littérature spécialisée. En effet, un agent se caractérise par [Alo08] : *la sociabilité, l'autonomie, la pro-activité, la réactivité, l'adaptabilité, l'intelligence et la mobilité*. Il est évident que l'existence de ces caractéristiques dans un agent est fortement liée au type de ce dernier. Ensuite, chaque caractéristique est divisée en un ensemble de sous-caractéristiques qui sont à leurs tours divisées en un ensemble de mesures.

La première caractéristique étudiée est la sociabilité [Alo08]. Cette caractéristique est représentée par trois attributs : la communication, la coopération et la négociation. La communication représente la capacité de l'agent d'échanger les messages avec les autres agents. La sociabilité peut prendre la forme de coopération, si un agent offre des services aux autres agents, comme elle peut avoir recours à la négociation en cas de résolution de conflits. Chaque attribut peut être évalué en utilisant plusieurs mesures.

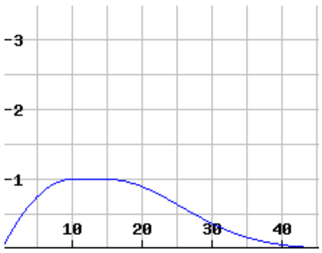
Chaque métrique proposée par Alonso et *al.* [Alo08] est une fonction mathématique avec un ou plusieurs paramètres. Les résultats de ces fonctions ont été normalisés entre 0 (*la mauvaise valeur*) et 1 (*la valeur optimale*). En effet, chaque fonction prend une forme spécifique qui met en évidence le niveau d'optimalité du phénomène étudié. Ces formes sont représentées dans la figure 3.1. Dans la figure (3.1 – A) par exemple, la valeur optimale peut être obtenue quand la valeur de x est inferieur à un paramètre k. Si la valeur de x dépasse le paramètre k, le résultat de la métrique va se dégrader progressivement pour atteindre la mauvaise valeur. Par contre, le deuxième type des formules présenté dans la figure (3.1 – B), montre que la valeur optimale est située entre deux paramètres k_1 et k_2. Le troisième type des formules (Figure 3.1 – C), montre que l'augmentation de la variable x, améliore en conséquence la valeur de la métrique jusqu'à l'obtention de la valeur optimale. Afin d'illustrer l'utilité de ces fonctions typiques, nous présentons la première mesure de l'attribut communication. Cette mesure appelée la réponse aux messages (*Response for message* (RFM)) est spécifié en fonction du nombre moyen des messages envoyés comme réponse à un message reçu (*M*). Les auteurs [Alo08] spécifient la métrique

RFM selon la deuxième fonction typique (Figure 3.1 - B). On peut donc conclure que l'augmentation du nombre des messages envoyés en réponse à un message reçu *(M)* améliore la *communicabilité* de l'agent jusqu'à un paramètre k_1. La *communicabilité* de l'agent atteint son niveau optimal si M est entre k_1 et k_2. Au-delà de k_2, la surcharge devient un obstacle de la communicabilité de l'agent. Bien entendu, plusieurs métriques ont été proposées de la même manière pour chaque attribut.

$$\begin{Bmatrix} 1 & 0 \leq x \leq k \\ e^{-\frac{(x-k)^2}{k^2}} & x > k \end{Bmatrix}$$

(NB. Dans le graphe, la valeur de k=10)

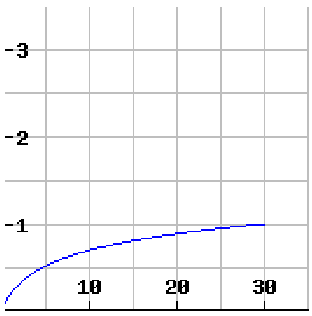
(A)



$$\begin{Bmatrix} \frac{2 * x}{k_1} - (\frac{x}{k_1})^2 & 0 \leq x < k_1 \\ 1 & k_1 \leq x \leq k_2 \\ e^{-\frac{(x-k_2)^2}{k_2^2}} & x > k_2 \end{Bmatrix}$$

(NB. Dans le graphe, la valeur de k_1=10 et la valeur de k_2=15)

(B)



$\log_k(x+1) \quad 0 \leq x \leq k$

(NB. Dans le graphe la valeur de k = 30)

(C)

Figure 3.1 : Les types de formules utilisées pour la mesure des attributs de la sociabilité [Alo08].

Dans le même cadre, Alonso et *al.* [Alo09, Alo10a] ont proposé des attributs et des métriques pour l'évaluation de l'autonomie et de la pro-activité. Le modèle global de la qualité des systèmes multi-agents proposé par Alonso et *al.* [Alo08, Alo09, Alo10a, Alo10b] peut être résumé dans le tableau 3.1. Le niveau des métriques n'est pas présenté dans ce tableau pour des raisons de brièveté. Il est important de noter que ce modèle n'a pas atteint sa version complète à cause de l'absence des études consacrées aux autres caractéristiques définies initialement (la réactivité, l'adaptabilité, l'intelligence et la mobilité). En plus, les auteurs proposent une autre version de mesures de la sociabilité et de l'autonomie sans prendre en compte les fonctions typiques [Alo10b].

Loin de son caractère incomplet, le modèle proposé par Alonso et *al.* pose plusieurs points discutables. Tout d'abord, nous pensons que les caractéristiques de l'agent sur lesquels le modèle a été fondé posent plusieurs inconvénients. En fait, nous croyons que plusieurs caractéristiques ne sont pas des caractéristiques clés pour le paradigme agent. *Est-ce que la mobilité ou l'adaptabilité sont des caractéristiques essentielles pour les agents ?* En plus, ce modèle ignore des caractéristiques plus importantes dans le domaine des SMA comme l'organisation et l'environnement.

Les caractéristiques d'un modèle de qualité doivent être *clairement* spécifiées. On doit éviter tout chevauchement entre ces caractéristiques. Cependant, nous croyons que cette règle n'est pas respectée dans le modèle de Alonso et *al.* En fait, l'intelligence des agents est une caractéristique composée de la sociabilité, la réactivité et la pro-activité de l'agent [Woo02b]. A notre avis, l'adaptabilité représente aussi une caractéristique de l'intelligence de l'agent.

Alonso et *al.* omettent les caractéristiques de haut niveau de la qualité logicielle, comme la fiabilité, l'efficacité, la maintenabilité, …etc. En fait, les caractéristiques de l'agent ne peuvent pas être considérées comme des caractéristiques pour un modèle de qualité si elles ne contribuent pas à la qualité des logiciels basés sur ce paradigme. En conséquence, il est important de spécifier les relations entre les caractéristiques spécifiques des agents et les caractéristiques de haut niveau de la qualité logicielle.

Enfin, on note que ce modèle ne propose pas une méthode pour la mesure des métriques proposées. En accord avec le standard ISO/IEC 9126 [ISO01], la méthode de mesure représente une partie importante de la mesure de la qualité.

Tableau 3.1 : Le modèle de la qualité des SMA de Alonso et *al.* (Synthèse de [Alo08, Alo09, Alo10a]).

Les caractéristiques	Les attributs
La sociabilité	La communication
	La coopération
	La négociation
L'autonomie	Le contrôle de soi (self-control)
	L'indépendance fonctionnelle
	L'évolution des capacités
La pro-activité	L'initiative
	L'interaction
	La réaction
La réactivité	
L'adaptabilité	
L'intelligence	
La mobilité	

Afin d'évaluer les systèmes multi-agents, Bitonto et *al.* [Bit12] ont proposé un modèle basé sur le paradigme *GQM* (pour *Goal-Question-Metric*). Ce paradigme spécifie un modèle selon trois niveaux : les buts, les questions et les métriques. Les buts représentent les attributs spécifiant la qualité d'un logiciel. Dans le deuxième niveau, des questions ont été utilisées pour spécifier chaque but. Le troisième niveau permet l'évaluation quantitative de ces buts.

Le modèle de Bitonto et *al.* [Bit12] considère la qualité des systèmes multi-agents selon cinq dimensions différents appelées des buts: *la complexité de l'environnement, la rationalité, l'autonomie, l'adaptabilité à l'environnement et la réactivité*. Pour chaque but, des questions et des métriques ont été proposées afin de l'évaluer. En fait, chaque but est spécifié à l'aide d'un ensemble de paramètres mesurables par un ensemble de métriques. A titre d'exemple, la complexité de l'environnement est spécifiée par trois paramètres : l'inaccessibilité, l'instabilité et la complexité de l'interaction. L'inaccessibilité de l'environnement d'un agent peut être mesurée en utilisant deux métriques : l'inaccessibilité des composantes de l'environnement et l'inaccessibilité des données de l'environnement. Le moyen des mesures de l'inaccessibilité des agents composant un système multi-agents représente l'inaccessibilité de ce dernier. Le tableau 3.2 présente une vue globale de ce modèle. Comme dans le cas du modèle précédent, on omet les métriques pour des raisons de brièveté.

Ce modèle présente plusieurs avantages. Tout d'abord, il est un modèle générique dans le sens où il est indépendant de l'implémentation des SMA ou le contexte de l'utilisation du système implémenté. En plus, ce modèle propose deux visions complémentaires concernant la qualité des SMA : la qualité des agents en tant qu'entités individuelles et la qualité du système global. Ce modèle peut être utilisé par les concepteurs pour vérifier l'adéquation du système implémenté face aux contraintes du problème comme il peut être utilisé par des évaluateurs pour choisir le meilleur système pour résoudre un problème.

A notre avis, si la généricité des métriques proposées grâce au paradigme *GQM* est considérée comme avantage, elle a aussi une conséquence négative. En fait, les métriques proposées sont des métriques statiques proposées sans spécification de la méthode de mesure. En plus, l'évaluation de ces métriques est subjective parce qu'elle n'est pas basée sur des formules exactes. Le deuxième inconvénient de ce modèle réside dans la définition des différents buts. En effet, le choix de différents buts n'est pas justifié. En conséquence, des caractéristiques optionnelles des agents figurent dans la liste des buts comme la rationalité et l'adaptabilité à l'environnement. Par contre, la pro-activité est considérée comme un paramètre de l'autonomie malgré que les références de domaine multi-agents considèrent ces deux attributs des caractéristiques totalement différentes.

Tableau 3.2 : Le modèle de la qualité des SMA de Bitonto et *al.*

Les buts	Les paramètres
La complexité de l'environnement	L'inaccessibilité
	L'instabilité
	La complexité de l'interaction
L'autonomie	La pro-activité
	L'autonomie dans la structure organisationnelle
La rationalité	Le mode de choix des actions
	La maximisation
La réactivité	L'efficacité de la perception
	La rapidité de la réponse (dans le temps opportun)
L'adaptabilité à l'environnement	La capacité de gérer les différentes situations
	La capacité de répondre aux nouveaux stimuli externes

Contrairement aux travaux présentés au-dessus, Kaddoum et *al.* [Kad09] ont ciblé l'évaluation des systèmes multi-agents adaptatifs. En effet, les auteurs ont proposé une évaluation autour de trois axes : le fonctionnement du système en cours d'exécution, les caractéristiques intrinsèques des systèmes et la méthodologie du développement. Il semble claire que seulement les deux premiers axes sont pertinents à la qualité du produit logiciel. Le troisième axe représente la qualité du processus de développement. Considérant les deux premiers axes qui sont plus proche à notre problématique, les auteurs divisent chaque axe en deux critères. Ainsi, le fonctionnement du système peut être évalué par le critère de performance et le critère d'homéostasie et robustesse. Par contre, les caractéristiques intrinsèques des systèmes sont définies par deux critères : la complexité algorithmique ainsi que la décentralisation et la résolution locale. Chaque critère peut être mesuré à l'aide d'une ou plusieurs métriques.

Malgré que ce travail essaie de donner une vue globale à la qualité en intégrant la qualité du produit et la qualité du processus, nous pensons que ce point de vue est *trop simpliste*. En effet, ce modèle ignore la nature multi-facettes de la qualité en résumant ce concept seulement à quatre critères. En plus, le modèle proposé ne donne aucune indication sur les caractéristiques des SMA.

3. Mesures de la qualité des SMA

La proposition des modèles de qualité pour les SMA ne représente qu'une petite partie du travail effectué dans ce domaine. En effet, la plupart des travaux effectués dans ce domaine consiste à proposer des métriques pour l'évaluation de différents critères de SMA. Les mesures proposées ne sont pas *nécessairement* liées *directement* au domaine de la qualité des SMA. Comme nous l'avons constaté auparavant, plusieurs métriques ont été proposées durant les débuts du domaine multi-agents. En conséquence, les premières mesures proposées ciblent l'évaluation de l'efficacité et les performances de ce nouveau paradigme [Bit12]. Des nouvelles approches s'orientent vers la mesure de caractéristiques intrinsèques des SMA. Malgré le contexte original dans lequel les différentes mesures ont été proposées, tous les attributs mesurés influent sur la qualité. Une synthèse profonde et actualisée de métriques proposées dans ce domaine a été présentée par Dumke et *al.* [Dum10]. En plus, les modèles proposés dans la section précédente présentent quelques métriques pour chaque critère. Cette section est consacrée à la présentation de quelques métriques proposées pour évaluer les spécificités des SMA. L'application des métriques déjà connues en génie logiciel orienté-objet ou génie logiciel traditionnel pour évaluer des propriétés des SMA est aussi possible [Sha14].

Barber et Martin [Bar99] proposent la spécification formelle et la mesure de l'autonomie parce qu'elle représente une caractéristique essentielle des agents. L'autonomie de l'agent est modélisée par trois composantes : les buts (G), les décideurs (D) et les contraintes d'autorité (C). La première composante représente les différents buts prévus et optionnels de différent agents. Tandis que la deuxième composante (D) représente un ensemble d'éléments représentants la capacité d'un agent de prendre les décisions nécessaires pour atteindre les buts de G. Cette capacité de décision est modélisée par un nombre entier strictement positif. Les contraintes d'autorité (C) est une liste d'agents qui doivent exécuter les décisions du décideur. Cette modélisation de l'autonomie permet ensuite de la mesurer. En fait, les auteurs [Bar99] ont choisi de mesurer ce concept en utilisant une seule valeur. Ainsi, l'autonomie d'un agent qui appartient à la liste des décideurs représente sa puissance de décision. Cette puissance peut être obtenue par la division de la capacité de prendre la décision de l'agent sur la somme de capacités de tous les agents de l'ensemble D. Par contre, si l'agent n'appartient pas à l'ensemble des décideurs, son autonomie devient nulle.

A notre avis, ce modèle est basé sur une vue abstraite du concept étudié. En conséquence, il est difficile de passer à une application réelle de cette mesure. Malgré que la capacité de prendre les décisions représente une notion clé dans ce travail, les auteurs [Bar96] ne précisent pas la méthode utilisée pour le calcul de cette capacité. Cette difficulté s'accroît si on considère l'autonomie de l'agent comme une propriété dynamique qui peut être affaiblie ou être renforcée au cours de l'exécution d'un agent. En plus, nous pensons que la proposition d'une mesure en une seule valeur pour les concepts complexes ne représente pas le bon choix. Il semble plus utile de présenter les mesurer de différentes facettes du concept afin d'augmenter l'utilité des ces métriques.

En remarquant qu'il n'existe pas des métriques pour mesurer la réactivité par contre à l'autonomie, la sociabilité et la pro-activité, Sivakumar et Vivekanandan [Siv12] ont proposé un ensemble de métriques pour l'évaluation de cette propriété. Ainsi, trois différents niveaux affectant cette propriété ont été définis : le niveau d'interaction, le niveau de communication et le niveau de perception. Chaque niveau est mesuré par plusieurs métriques. Le tableau 3.3 donne une vue de différents niveaux avec les métriques proposées. A titre d'exemple, le niveau de perception d'un agent peut être mesuré à l'aide de ses connaissances et la mise à jour effectuée sur ces dernières. L'usage de connaissances représente le moyen des attributs internes d'un agent utilisés (par des instructions de lecture) pour la prise de décision. Par contre, la deuxième mesure représente le nombre d'instructions qui mettent à jour les variables d'un agent. Les différentes métriques proposées sont des métriques statiques basées sur l'analyse syntaxique des logiciels basés-agent. Dans ce contexte, un outil a été développé pour l'analyse et la mesure de la réactivité des programmes JADE.

Tableau 3.3 : Les métriques de la réactivité de Sivakumar et Vivekanandan.

Les niveaux	Les métriques
Le niveau d'interaction	Les méthodes par classe
	Nombre de types de messages
Le niveau de perception	L'utilisation de connaissances
	La mise à jour de connaissances
Le niveau de communication	Les réponses aux messages
	Les messages entrants
	Les messages sortants

Nous croyons que ce travail souffre de plusieurs inconvénients majeurs. Tout d'abord, la communication et l'interaction ne représente pas des concepts totalement indépendants dans le domaine multi-agents. Comme nous l'avons présenté dans le chapitre précédent, la communication est considérée comme le support de l'interaction. Le deuxième inconvénient consiste à la présentation vague et superficielle de différentes métriques. En fait, la relation entre les métriques et la réactivité n'est pas toujours évidente. A titre d'exemple, l'influence de méthodes par classe sur l'interaction et la réactivité n'est pas claire. La réactivité désigne la capacité de percevoir l'environnement et répondre aux changements de ce dernier *en temps opportun*. Malgré que les auteurs [Siv12] aient basé leur travail sur cette définition, ils ignorent l'aspect temporel lors la spécification des métriques. A titre d'illustration, la mise à jour de connaissances ne peut être considérée comme une métrique pour la réactivité si cette action n'est pas une conséquence d'un évènement externe. Nous pensons que les métriques statiques ne permettent pas d'évaluer la réactivité des agents parce qu'elles ignorent cet aspect temporel.

L'intelligence peut être considérée comme étant la caractéristique la plus importante après l'autonomie. Il est donc naturel de mesurer cette caractéristique. Mahar et Bhatia [Mah14] ont proposé plusieurs métriques pour la mesure de l'intelligence des agents. Comme il est présenté dans le tableau 3.4, l'intelligence des agents est modélisée par trois attributs : l'apprentissage, l'adaptabilité et l'orientation par des buts.

Tableau 3.4 : Les métriques de l'intelligence de Mahar et Bhatia.

Les attributs	Les métriques
L'adaptabilité	La pondération des méthodes par classe
	Les fonctions de gestion d'exceptions
L'orientation par des buts	Le nombre de rôles
	La réalisation des objectifs d'agents
L'apprentissage	Le facteur de masquage d'attributs
	La densité de variables
	L'utilisation de connaissances
	La mise à jour de connaissances

Les métriques proposées pour l'évaluation de l'intelligence des agents sont [Mah14] :

- La pondération des méthodes par classe (*Weighted Method per Class*) : cette métrique mesure la somme de la complexité *cyclomatique* des méthodes définies au sein d'un agent. Une grande valeur désigne un agent complexe qui est capable de s'adapter aux différentes situations.
- Les fonctions de gestion d'exceptions (*Exception Handling Functionality*) : cette métrique est la somme des types d'exception traités par l'agent. Elle représente l'efficacité de traiter les différents états de l'environnement.
- Le nombre de rôles (*Number of Roles*) : le nombre de rôles exécutés par un agent reflète la complexité des algorithmes de ce dernier.

- La réalisation des objectifs d'agents (*Agent Goal Achievement*) : cette métrique désigne la capacité de l'agent d'atteindre ses objectifs. Selon les auteurs [Mah14], des faibles valeurs de cette métrique désignent une bonne capacité d'adaptation.
- Le facteur de masquage d'attributs (*Attribute Hiding Factor*) : cette métrique représente le ratio des attributs masqués par rapport à l'ensemble des attributs définis dans un agent. Une faible encapsulation donne la possibilité d'affecter le comportement de l'agent.
- La densité de variables (*Variable Density*) : cette métrique est la taille consacrée à la représentation de l'état interne de l'agent.
- L'utilisation de connaissances (*Knowledge Usage*) : cette métrique représente le nombre de variables utilisés dans une instruction de décision.
- La mise à jour de connaissances (*Knowledge Update*) : cette métrique représente le nombre d'instructions qui changent les variables d'un agent.

Loin de la modélisation de l'intelligence qui ignore plusieurs attributs essentiels pour l'agent (comme la réactivité et la sociabilité), plusieurs remarques peuvent être faites sur les métriques proposées. Tout d'abord, la relation de différentes métriques avec les attributs est vague ou injustifiable. Par exemple, les auteurs [Mah14] n'ont pas justifié comment la faible réalisation de buts peut influencer positivement sur la capacité d'adaptation. A notre avis, l'adaptation est une caractéristique qui améliore la capacité d'agent pour atteindre ses buts. De manière similaire, la relation entre la densité de variables et l'apprentissage n'est pas évidente. En plus, le calcul de cette métrique (la densité de variables) n'est pas possible à partir du code source à cause de l'existence possible de variables dynamiques. Enfin, malgré que le masquage d'attributs est une métrique connue dans le génie logiciel orienté objet, il n'est pas convenable d'utiliser cette métrique pour les SMA. En fait, les attributs de l'agent doivent être encapsulés pour assurer l'autonomie qui est une propriété importante de l'agent.

Contrairement à Mahar et Bhatia [Mah14], Cabrera et Orallo [Cab13] ont introduit la notion de l'*intelligence sociale*. Les auteurs [Cab13] ont remarqué que le niveau d'intelligence de l'individu ne lui permet pas toujours d'atteindre ses objectifs s'il n'exploite pas ses capacités sociales. En fait, l'intelligence sociale représente la capacité de l'agent d'interagir correctement dans un environnement peuplé par des agents intelligents. Cette notion est fortement liée aux contextes qui nécessitent des capacités de communication, de coopération et de compétition. Comme nous l'avons vu dans le chapitre précédent, la sociabilité représente une caractéristique importante pour les agents intelligents. En effet, plusieurs travaux visent l'évaluation de cette caractéristique qui a un impact direct sur les performances et la complexité des SMA. Nous présentons ici deux exemples de travaux ciblant l'aspect social des SMA.

Joumaa et *al.* [Jou08] remarquent que l'évaluation des SMA permet de comprendre leurs comportements et la comparaison de différents systèmes. Cependant, l'évaluation de l'interaction se caractérise par deux problèmes spécifiques. Premièrement, l'effet d'une unité d'interaction pour un système peut être équivalent à plusieurs unités pour un autre système. Deuxièmement, l'interaction des agents ne signifie pas l'utilité de messages reçus. En conséquence, l'évaluation de l'interaction proposée dans ce travail est basée sur la quantité de l'information échangée et l'utilité de ces informations. En plus, cette évaluation est basée sur un modèle des SMA défini pour cet objectif. Ce modèle spécifie l'agent comme un système composé des parties suivantes : une *unité de traitement* responsable de la prise de décisions, une *mémoire* représentant l'état interne de l'agent, une *interface* pour la réception de messages et une *interface* pour l'envoie de ces derniers. En fait, l'échange de messages représente la seule possibilité d'interaction entre l'agent et les autres composants du système. Au niveau système, le traitement est divisé en deux étapes : l'interaction et la post-interaction. L'interaction est représentée par l'envoi et la réception des messages. Comme il est présenté dans la formule suivante (3.1), l'interaction est une fonction de l'ensemble des messages qui peuvent être envoyés par l'agent A (M_{Sent}^{A}) vers l'ensemble des messages qui peuvent être reçus par l'agent B. Cet ensemble représente les messages compris par l'agent B ($M_{Received}^{B}$) et les messages non-compris ($m_{\emptyset}^{B}$).

$$Interaction = M_{Sent}^{A} \rightarrow M_{Received}^{B} \cup \{m_{\emptyset}^{B}\} \qquad (3.1)$$

La post-interaction consiste à mémoriser le message puis générer l'action adéquate. La fonction de mémorisation (MEM^{A}) consiste à changer l'état interne de l'agent en fonction du message reçu et l'état interne actuel de l'agent (3.2). En outre, la fonction de décision (DEC^{A}) consiste à utiliser le message reçu et l'état interne de l'agent pour générer une action (3.3). Dans certains cas, cette décision peut être une action nulle représentée par ($ac_{ø}{}^{A}$).

$$MEM^{A} = M^{A}_{Received} \times S^{A} \rightarrow S^{A} \quad (3.2)$$

$$DEC^{A} = M^{A}_{Received} \times S^{A} \rightarrow AC^{A} \cup \left\{ac^{A}_{\emptyset}\right\} \quad (3.3)$$

Comme nous l'avons indiqué avant, ce travail focalise sur deux aspects : la quantité d'information échangée et l'utilité de cette dernière. En effet les auteurs proposent le concept d'*interaction pertinente*. Un message pertinent est un message qui change l'état interne du récepteur ou invoque une action. En conséquence, on focalise sur la quantité des informations pertinentes au lieu de la quantité totale des informations échangées. Ensuite, la notion de *poids de message pertinent* (Φ) a été introduite pour permettre la comparaison de deux situations d'interaction dans deux systèmes différents. Cette fonction quantifie l'effet de différents messages. La quantification de l'effet d'un message peut être basée sur le type de message ou sur son effet sur l'état interne et les actions de l'agent.

Malgré l'importance du concept introduit (l'*interaction pertinente*), nous croyons que les messages non-pertinents et même les messages incompréhensibles ont un effet sur les SMA. En fait, on ne peut pas ignorer le processus d'échange de ces messages à cause de son effet sur les performances des systèmes. A partir de la charge de l'activité de communication, Gutiérrez et Magariño [Gut09] proposent cinq catégories d'agents :

- **L'agent chargeur :** un agent charge la communication par l'envoie excessif de messages ;
- **L'agent chargé :** un agent chargé par la réception excessif de messages ;
- **L'agent chargeur-chargé :** un agent chargeur et chargé en même temps ;
- **L'agent isolé :** un agent qui n'a pas une activité de communication ;
- **L'agent régulé :** un agent avec un comportement idéal en ce qui concerne l'activité de communication.

Ensuite quelques métriques on été proposées pour l'évaluation de la catégorie des agents d'un SMA par rapport à deux portées différentes : au niveau du système global et au niveau des agents jouant le même rôle. Deux métriques ont été proposées pour évaluer la catégorie de l'agent au niveau du système : le nombre de messages envoyés par rapport au nombre total des messages envoyés par les autres agents et le nombre de message reçus par rapport au nombre total des messages reçus par les autres agents composants le système. De la même manière, au niveau des agents jouant le même rôle deux métriques ont été proposées : le nombre de messages envoyés par rapports au nombre de messages envoyés par les agents jouant le même rôle et le nombre de messages reçus par rapports au nombre de messages reçus par les agents jouant le même rôle.

Contrairement aux travaux ciblant les propriétés des agents, il existe plusieurs travaux qui essaient d'évaluer des aspects génériques (c'est-à-dire qui existent dans tous les logiciels) en prenant en compte les spécificités des SMA. Par exemple, Magariño et *al.* [Mag10] ont modélisé et mesuré la qualité de l'architecture des SMA, Russel et Norvig [Rus03] ont introduit l'évaluation des performances des SMA et Caballero et *al.* [Cab03] ont proposé la mesure de l'utilisabilité des SMA dans le domaine de E-learning. Dans la suite de cette partie, nous nous concentrerons sur la complexité des SMA parce qu'une partie de nos contributions présentées dans cette thèse consiste à l'évaluation de cette caractéristique.

La complexité des SMA a été étudiée dans plusieurs niveaux et avec des points de vue différents. D'une part, la complexité peut être évaluée durant la conception [Mag10] comme elle peut être une caractéristique d'un code [Dos03a, Klü08]. D'autre part, la notion de complexité peut faire référence à la complexité computationnelle et algorithmique [Dek02, Dzi07] comme elle peut designer la complexité de comprendre et maintenir des produits logiciels [Dos03a, Klü08]. Dans cette partie nous entendons par la complexité le niveau de difficulté associé à la compréhension et la maintenance du produit logiciel final.

Dospisil [Dos03a] a étudié la complexité de codes des agents mobiles implémentés en utilisant la programmation orientée-aspects. Ce paradigme de programmation permet de séparer les préoccupations transversales du cœur du logiciel. En conséquence, les SMA étudiés ici sont implémentés en séparant les interactions de fonctionnement des agents. Afin de mesurer la complexité de tels systèmes, Dospisil [Dos03a] a utilisé la notion d'*entropie*. Cette notion, proposée initialement dans le domaine de thermodynamique, exprime le niveau de désordre d'un système. Ainsi, l'entropie a été utilisée pour plusieurs auteurs pour mesurer la complexité des logiciels orientés-objets [Dos03b]. Cette approche passe par la présentation du logiciel en graphe où les nœuds représentent les symboles (des variables, des méthodes, des classes ou des aspects) et les arcs représentent les relations de dépendance entre les symboles. L'entropie d'un programme est calculée par la formule (3.4) telle que *H(S)* est l'entropie de tous les arcs, *H(E)* est l'entropie des arcs fournissant des informations sources d'un nœud de symbole, V_s est le nombre total des symboles et V_e est le nombre total des arcs.

$$H(P) = H(S) + \frac{V_s}{V_e} + H(E) \qquad (3.4)$$

L'inconvénient majeur de ce travail est son contexte d'application très limité (les agents mobiles implémentés en utilisant *AspectJ*).

Klügl [Klü08] a proposé l'évaluation de la complexité des logiciels de simulation basés sur les systèmes multi-agents. Comme il est présenté dans le tableau 3.5, la complexité de tels logiciels est classifiée selon trois niveaux : le niveau d'agent, le niveau du système global et le niveau de l'agent-système (c'est-à-dire le niveau d'interaction entre les agents). Chaque niveau est mesuré par un ensemble de métriques. Il est évident que notre niveau de compréhensibilité d'un système est fortement influencé par l'hétérogénéité et la taille de ce dernier. En effet, on considère l'existence de plusieurs métriques inévitables, comme le nombre de types d'agents (l'hétérogénéité des agents), le nombre de types de ressources (l'hétérogénéité de l'environnement) et le niveau de complexité de l'architecture interne de l'agent. Cependant, certaines métriques sont subjectives, inadéquates ou très spécifiques. Le niveau de complexité architecturale d'un agent est défini en référence à trois architectures d'agents (l'architecture décrivant des comportements, celle configurant des comportements et celle génératrice de comportements [Klü08]). Nous croyons que cette métrique est subjective parce qu'on peut trouver des agents de l'architecture décrivant le comportement mais plus complexe que des agents de l'architecture génératrice de comportement. Il semble plus objectif si ce niveau est calculé à l'aide des attributs internes de l'agent (comme le nombre des états possibles ou le nombre des comportements).

Tableau 3.5 : Les métriques de la complexité de Klügl.

Les niveaux	Les métriques
Le niveau de système global	Le nombre de types d'agents
	Le nombre de types de ressources
	Le nombre maximal des agents
	Le nombre minimal des ressources
	La variabilité maximale de nombre des agents
	La variabilité maximale de nombre des ressources
	La relation agent-ressource
	Le nombre des formes d'un agent
	Le nombre des formes de ressources
	La taille maximale des états de ressources
	Le nombre de paramètres affectant les états de ressources
Le niveau d'agent	Le niveau de complexité architecturale
	La plasticité des actions (l'adaptabilité)
	La taille de connaissances procédurales
	Le nombre de règles cognitives
Le niveau de l'agent-système	La somme des Items publiques
	Le nombre des accès aux données externes
	Le nombre de références d'agents dans un agent
	Le nombre de références de ressources dans un agent
	Le nombre d'action de mobilité

La métrique *nombre maximal des agents* est l'exemple des métriques inadéquates. En fait, le nombre des agents dans plusieurs systèmes (comme les systèmes multi-agents ouverts) est dynamique et non borné. Dans ce cas, on parle plutôt de la variabilité du nombre d'agents (une métrique proposée dans ce travail) ou du nombre d'agents à chaque instant. Par la spécificité des métriques, on entend leur contexte très limité. Malgré que le travail soit proposé pour un domaine spécifique (la simulation basée sur les SMA), certaines métriques sont applicables seulement dans des cas limités. Par exemple, la métrique représentant la complexité spatiale (le nombre des formes d'un agent) est limitée à la simulation basée sur des représentations spatiales. De la même façon, le nombre d'actions de mobilité est spécifique à la simulation qui exploite les agents mobiles. En plus de ces remarques, nous pensons que certains aspects plus influant sur la complexité ont été ignorés. Par exemple, l'interaction basée sur l'échange de messages n'est pas mesurée malgré son impact sur la compréhension du système global. Concernant le modèle proposé (les trois niveaux), nous croyons que l'interaction représente une partie intégrée dans la complexité du système.

4. Etude comparative

Nous avons présenté ci-dessus quelques travaux qui traitent la qualité des SMA. Cette problématique a été traitée selon deux angles différents : la modélisation de la qualité des SMA ou la mesure de différents aspects liés à la qualité de tels systèmes. Bien entendu, les contributions apportées dans les solutions présentées n'empêchent pas l'existence de plusieurs insuffisances. En fait, chaque contribution présentée est jugée afin de définir la situation actuelle de ce domaine. Cette évaluation nous permet de tirer plusieurs critères qu'on peut exploiter comme un framework de comparaison. A notre avis ces travaux peuvent être comparés selon les axes suivants :

- **La spécification de la qualité** : la qualité est un concept complexe. Pour la maîtriser et éviter toute éventuelle ambigüité, il serait très intéressant, dans un premier temps, de spécifier ce concept. Selon cet axe, on peut spécifier trois niveaux. Le premier niveau consiste au traitement d'un attribut spécifique de l'agent en ignorant le rapport de ce dernier avec la qualité. Dans un niveau intermédiaire, on peut spécifier la qualité des SMA en fonction des attributs spécifiques des agents (comme l'autonomie, la sociabilité et la réactivité). Nous pensons que ces attributs ne sont pas des objectifs s'ils ne contribuent pas à l'amélioration de la qualité en termes de quelques caractéristiques de haut niveau (comme l'efficacité, la fiabilité et l'utilisabilité). En effet, le dernier niveau représente une modélisation de la qualité en utilisant des caractéristiques de haut niveau et les attributs spécifiques au paradigme agent.

- **La base standardisée :** comme nous l'avons vu au premier chapitre, il existe plusieurs modèles de qualité standards. Il semble évident, que la standardisation des modèles présente une vue homogène et unifiée du concept vague comme la qualité. En conséquence, ces modèles standards facilitent la comparaison et l'échange des résultats. On note que ce critère ne s'applique que dans le cas des travaux modélisant la qualité.
- **La généricité du modèle d'agents:** l'absence de consensus concernant la définition de l'agent, comme nous l'avons présentée au chapitre précédent, influe directement sur la spécification des attributs de ce dernier. En conséquence, la modélisation de la qualité qui est basée sur ces attributs devient discutable. L'utilisation d'un modèle spécifique de l'agent peut être considérée comme un remède à ce problème. Cependant, cette solution est payée en contrepartie par l'applicabilité limitée des propositions. La spécification des attributs les plus acceptés dans ce domaine représente peut être un compromis prometteur.
- **La portée d'application :** la modélisation de la qualité des SMA ou les métriques utilisées pour l'évaluation de différents attributs de ce concept peuvent être génériques (c'est-à-dire applicables sur n'importe quel système) ou spécifique pour des domaines limités. Malgré que la limitation du domaine cerne les différents attributs intrinsèques pour le concept étudié, cette limitation a un impact négatif sur l'applicabilité des propositions. Nous croyons que la généricité offre une base extensible qu'on peut utiliser ensuite pour des domaines plus spécifiques.
- **La méthode de mesure :** la plupart des travaux proposent des métriques sous forme de formules mathématiques sans aucune indication de la méthode de mesure. Selon le standard ISO/IEC 9126 [ISO01], la méthode de mesure est partie non négligeable de métriques. Nous croyons que le problème s'accentue plus pour le cas des SMA. En fait, la flexibilité des agents, qui est une caractéristique essentielle [Woo02b], rend l'application des métriques dynamiques recommandées. Ce type de métriques est généralement plus difficile par rapport aux métriques statiques.

Le tableau 3.6 est une présentation récapitulative de résultats de comparaison.

Tableau 3.6 : Résultats de l'étude comparative.

	Niveau de modélisation	**La base standardisée**	**La généricité du modèle d'agents**	**La portée d'application**	**La méthode de mesure**
Alonso et al. [Alo08, Alo09, Alo10a, Alo10b]	Modélisation de la qualité seulement par les attributs des SMA	Non	Attributs spécifiés à partir de la littérature	Générique	Ignorée
Bitonto et *al.* [Bit12]	Modélisation de la qualité seulement par les attributs des SMA	Non	Attributs spécifiques aux agents	Générique	Ignorée
Kaddoum et *al.* [Kad09]	Ignore les spécificités des SMA	Non	Ignorée	Spécifique aux SMA adaptatifs	Ignorée
Barber et Martin [Bar99]	Un attribut spécifique aux agents (l'autonomie)	/	Spécification formelle de l'autonomie	Générique	Subjective
Sivakumar et Vivekanandan [Siv12]	Un attribut spécifique aux agents (la réactivité)	/	Modélisation de la réactivité en fonction des attributs de l'agent	Générique	Statique
Mahar et Bhatia [Mah14]	Un attribut spécifique aux agents (l'intelligence)	/	Modélisation de l'intelligence en fonction des attributs de l'agent	Générique	Ignorée

Joumaa et *al.* [Jou08]	Un attribut spécifique aux agents (l'interaction)	/	Aucun modèle n'est spécifié	Générique	Ignorée
Gutiérrez et Magariño [Gut09]	Un attribut spécifique aux agents (la communication)	/	Aucun modèle n'est spécifié	Générique	Ignorée
Dospisil [Dos03a]	Un attribut générique (la complexité)	/	Aucun modèle n'est spécifié	Agents mobiles implémentés par *AspectJ*	Statique
Klügl [Klü08]	Un attribut générique (la complexité)	/	Attributs spécifiques à la simulation basée-agent	Simulation basée-agent	Ignorée

Cette étude comparative nous permet de spécifier les insuffisances majeures des études effectuées dans ce domaine. Malgré que les travaux existants aient étudié la qualité des SMA à plusieurs niveaux (la modélisation de la qualité, la mesurer de certains attributs spécifiques aux SMA ou l'étude de spécificités des attributs génériques des logiciels par rapport aux SMA), ce domaine souffre des limites suivantes :

- L'absence d'un modèle de la qualité qui combine les caractéristiques de haut niveau de la qualité (l'efficacité, la fiabilité, la maintenabilité …etc.) avec les attributs des agents (l'autonomie, la sociabilité, la réactivité,…etc.). Nous sommes convaincus que les attributs des agents ne représentent pas un gain pour plusieurs acteurs de développement des logiciels s'ils n'affectent pas les caractéristiques de haut niveau de la qualité. En conséquence, la relation entre ces caractéristiques de la qualité et les attributs de l'agent doit être étudiée.
- La modélisation de la qualité des SMA doit prendre en considération les différents standards de qualité existants. Dans plusieurs domaines du génie logiciel, les spécialistes remarquent la capacité de modèles standards de la qualité pour la modélisation de paradigmes spécifiques éventuellement avec

quelques extensions [Zei07, Beh09, Her10, Bau12]. En conséquence, nous pensons que le développement d'un modèle de la qualité basé sur les standards existants offre plusieurs avantages comme la possibilité d'échange de résultats et la simplicité de comparaison de différents programmes implémentés.

- La méthode de mesure : les méthodes de mesure sont souvent omises dans les travaux cités malgré qu'elles représentent un attribut important dans les métriques logicielles selon le standard ISO/IEC 9126 [ISO01]. Nous croyons que la nature flexible des SMA nécessite l'application des métriques dynamiques. L'application de ces dernières n'est pas évidente. Donc, il est nécessaire de spécifier explicitement la méthode de mesure appliquée.
- La généricité du modèle et d'applications: la spécification de la qualité des SMA doit prendre en considération les caractéristiques intrinsèques de tels systèmes. Ces caractéristiques sont souvent représentées à l'aide de modèles. De la même manière, ces caractéristiques sont en relation avec le domaine d'application. En effet, un modèle de qualité générique peut être étendu pour supporter les spécificités d'un modèle d'agent ou d'un domaine d'application.

5. Conclusion

Le paradigme agent est un paradigme logiciel prometteur. Nous pensons que ce paradigme a atteint un niveau de maturité intéressant, ce qui permet d'aborder la question de la qualité des SMA. En fait, plusieurs études ont été proposées pour la modélisation de la qualité des SMA ou la mesure de différents attributs de tels systèmes. Dans ce chapitre, nous avons présenté un état de l'art de ce domaine suivi d'une étude comparative. Cette étude comparative nous permet d'identifier les insuffisances majeures des travaux présentés dans ce domaine. A notre avis ces insuffisances se résument dans l'absence d'un modèle de qualité global pour les SMA qui combine les caractéristiques standards de la qualité avec les attributs spécifiques aux agents. En conséquence, nous avons exploité ces remarques afin de proposer un modèle de qualité spécifique aux SMA. Le modèle proposé est présenté dans le chapitre suivant.

Chapitre 04

Modèles de Qualité pour les Systèmes Multi-Agents

« La seule utilité du modèle est de montrer
comment les autres définissent
leur propre réalité »

1. Introduction

Dans les chapitres précédents, nous avons dessiné les grands traits qui caractérisent les domaines reliés à notre axe de recherche. L'étude comparative présentée à la fin du chapitre précédent nous permet de mettre en évidence les insuffisances majeures des approches proposées dans le domaine de la qualité des SMA. Ces insuffisances peuvent être résumées ***en l'absence d'un modèle de qualité qui englobe, à la fois, les caractéristiques de haut niveau des produits logiciels et les spécificités intrinsèques des SMA***. Nous allons présenter, dans ce chapitre, nos contributions pour combler ces lacunes. Nos contributions dans la piste de la modélisation de la qualité des SMA consistent à la proposition d'un modèle de qualité pour les SMA (appelé *QM4MAS* pour *Quality Model for Multi-Agent Systems*) [Mar15] et l'étude de l'applicabilité du modèle ISO/IEC 25010 sur les SMA [Mar14a] comme un premier pas vers l'utilisation de ce standard. Ainsi, ce chapitre est articulé autour de deux parties : la présentation du modèle proposé et l'étude de l'applicabilité du standard ISO/IEC 25010.

2. QM4MAS : Un Modèle de qualité pour les SMA

Notre objectif est le développement d'un modèle global pour les systèmes multi-agents. Par un modèle global, nous entendons un modèle qui englobe les caractéristiques de la qualité des SMA contrairement aux modèles qui présentent une caractéristique spécifique de la qualité. En plus, le modèle proposé combine les caractéristiques génériques des logiciels avec les caractéristiques spécifiques des SMA. Notre modèle, appelé *QM4MAS* (pour *Quality Model for Multi-Agent Systems*), est une extension du modèle standard ISO/IEC 9126 [Mar15]. En fait, les modèles hiérarchiques, comme le modèle standard ISO/IEC 9126, représente un moyen bien accepté pour comprendre et définir la qualité.

Sachant qu'un modèle de qualité est défini par l'ISO comme *un ensemble de caractéristiques et sous-caractéristiques reliées entre eux afin de spécifier et évaluer la qualité du logiciel* [ISO01], nous avons suivi les étapes suivantes pour étendre le modèle ISO/IEC 9126 afin de supporter les particularités des SMA [Mar15]:

- Proposer un méta-modèle pour le modèle QM4MAS ;

- Etendre les caractéristiques du modèle ISO/IEC 9126 pour spécifier les caractéristiques des SMA ;
- Etendre les sous-caractéristiques du modèle ISO/IEC 9126 afin d'exprimer les particularités des SMA ;
- Proposer des métriques pour l'évaluation des sous-caractéristiques ajoutées.

2.1. Le méta-modèle de QM4MAS

Avant de développer notre modèle de qualité spécifique aux SMA, nous avons jugé l'importance de la proposition d'un méta-modèle qui spécifie la structure de notre futur modèle. Un méta-modèle sert à la définition précise des éléments composant le modèle et leurs relations [Dei09]. En effet, il empêche les ambigüités lors de l'utilisation du modèle comme il permet le futur raffinement de ce modèle. Parce que nous avons choisi de définir notre modèle comme une extension du modèle ISO/IEC 9126, il est naturel d'inspirer notre méta-modèle de la structure du modèle standard. Il est à noter l'absence d'un méta-modèle *explicite* pour le modèle standard ISO/IEC 9126.

La figure 4.1 présente le méta-modèle proposé. Ainsi, le modèle de qualité est composé de plusieurs caractéristiques. Chaque caractéristique est définie par un nom et une définition précise. En plus, une caractéristique est affectée par une ou plusieurs sous-caractéristiques telle que chacune est spécifiée par un nom et une définition. Une sous-caractéristique est mesurée par des métriques. En accord avec les définitions de l'ISO/IEC 9126, une métrique est définie par un nom, une définition, une méthode de mesure et une échelle. Cependant, nous n'avons pas suivi le standard ISO/IEC 9126 dans sa définition des cardinalités de la relation entre les caractéristiques et les sous-caractéristiques. Le modèle ISO/IEC 9126 limite l'affectation d'une sous-caractéristique à une seule caractéristique. Cependant, dans notre modèle une sous-caractéristique peut affecter plusieurs caractéristiques. Par exemple, la modularité comme une sous-caractéristique affecte les deux caractéristiques : la maintenabilité et la réutilisabilité [Mar15].

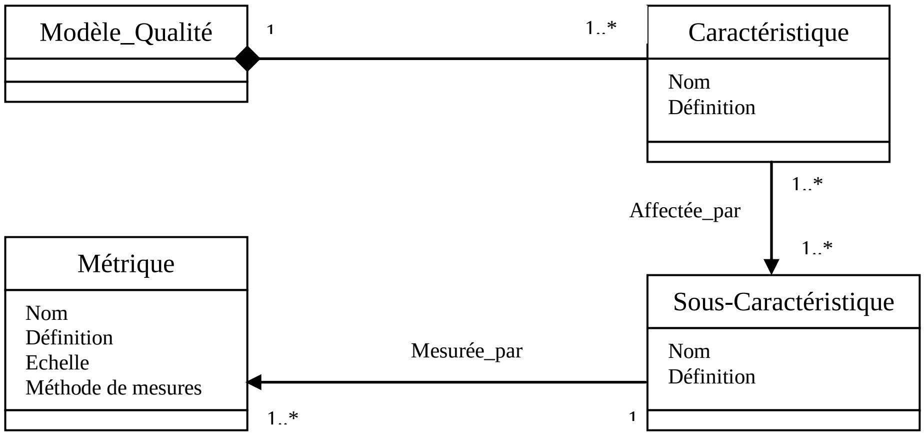


Figure 4.1: Le méta-modèle du QM4MAS [Mar15].

Formellement, la qualité des logiciels basés sur les SMA (Q) est définie en fonction de caractéristiques pondérées (4.1) :

$$Q = f((C_1, P_1), (C_2, P_2), \dots, (C_n, P_n)) \qquad (4.1)$$

Sachant que C_i et P_i représentent respectivement la mesure de caractéristique de qualité et son poids, la fonction f est une fonction de moyenne arithmétique pondérée. Ainsi, la qualité Q est spécifiée par la formule (4.2) :

$$Q = \frac{\sum_{i=1}^{n} C_i * P_i}{\sum_{i=1}^{n} P_i} \qquad (4.2)$$

De la même manière, chaque caractéristique $C_{i(i=1\ \dots n)}$ est spécifiée comme une moyenne arithmétique pondérée de mesures des sous-caractéristiques qui l'affectent. En plus, une sous-caractéristique est calculée par la moyenne arithmétique pondérée de mesures.

Les modèles hiérarchiques, et notamment le modèle ISO/IEC 9126, souffrent de l'ambigüité à cause de l'absence d'un critère précis utilisé pour distinguer les caractéristiques de sous-caractéristiques. Afin d'éviter ce problème, nous proposons d'adopter la terminologie de la méthodologie standard de métriques logicielles proposée par l'IEEE [IEEE98]. Ainsi, une caractéristique (appelée facteur) est *un attribut orienté-gestion qui contribue à la qualité logicielle* [IEEE98]. Par contre, une sous-caractéristique (appelée sous-facteur) est *une décomposition de la caractéristique en ses composantes techniques*. En autre terme, les caractéristiques représentent des propriétés de haut niveau ciblées par la communauté du développement logiciel afin d'améliorer la qualité du produit. Par contre, les sous-caractéristiques représentent les moyens techniques utilisées pour atteindre un niveau donné de la caractéristique ciblée. En plus, une métrique est *une fonction qui prend en entrée des données du logiciel et qui donne comme sortie une seule valeur désignant le degré d'un attribut de qualité existant dans le produit logiciel* [IEEE98].

2.2. Les caractéristiques de QM4MAS

Le méta-modèle proposé spécifie trois composants pour constituer notre modèle. Le premier composant est constitué d'un ensemble de caractéristiques. En fait, le standard ISO/IEC 9126 propose six caractéristiques pour la qualité du logiciel [ISO01] : *la fonctionnalité, la maintenabilité, l'efficacité, la fiabilité, la portabilité et l'utilisabilité.* Ce modèle est applicable sur les logiciels sans tenir compte de leurs types [ISO01]. En conséquence, nous pensons que toutes ces caractéristiques sont des caractéristiques fondamentales pour la qualité des SMA [Mar15]. Généralement, le premier niveau d'un modèle de qualité est indépendant du type de logiciels. Notre avis est en harmonie avec la décision de Alonso et *al.* [Alo98] de réutiliser les mêmes facteurs de McCall et *al.* [McC77] pour spécifier la qualité des logiciels-orientés objets.

Néanmoins, Dromey [Dro95] remarque depuis longtemps que la *réutilisabilité* est une caractéristique importante négligée dans le modèle standard. En conséquence, nous avons ajouté cette caractéristique à notre modèle de qualité pour les SMA [Mar15]. La réutilisabilité est définie dans notre modèle, selon le standard IEEE de terminologies du génie logiciel [IEEE90], comme *le degré de la possibilité d'utilisation d'un module logiciel dans plusieurs programmes*. A notre avis, l'importance de cette caractéristique est indépendante du paradigme agent.

Parmi les raisons encourageant l'utilisation des agents intelligents pour le développement des logiciels modernes on trouve la capacité de produire des comportements flexibles [Woo02b]. La flexibilité désigne *la capacité de l'agent de changer son comportement selon la situation afin d'atteindre ses objectifs* [Kir06]. En analysant le modèle standard, on ne trouve que l'adaptabilité comme une sous-caractéristique. Malgré que le standard IEEE de terminologies du génie logiciel [IEEE90] considère la flexibilité et l'adaptabilité comme des synonymes, nous avons choisi de considérer la flexibilité comme une caractéristique de qualité et laisser l'adaptabilité comme une sous-caractéristique. Notre choix est basé sur le principe de différence entre les caractéristiques et les sous caractéristiques définies dans notre méta-modèle. En fait, la flexibilité (comme un changement de comportement) représente un attribut contribuant à la qualité du logiciel. Cet attribut peut être implémenté par plusieurs techniques comme l'adaptabilité, la réactivité et l'apprentissage [Mar15].

Pour conclure, le modèle QM4MAS est constitué, dans son premier niveau, de huit caractéristiques. Il est important de noter que les extensions appliquées sur le niveau de caractéristiques du modèle ISO/IEC 9126 n'affectent pas l'esprit de ce dernier [Mar15]. Notamment, le nombre des caractéristiques reste dans l'intervalle défini par le standard (entre six et huit) et les caractéristiques sont définies avec le minimum de chevauchement grâce à l'adoption de terminologies standardisées ou bien-acceptées dans la littérature. Le tableau 4.1 présente les différentes caractéristiques du modèle proposé avec leurs définitions.

Tableau 4.1 : Les caractéristiques du modèle QM4MAS.

Caractéristiques	Définition
Fonctionnalité	La capacité du produit logiciel d'offrir des fonctions qui répondent aux besoins implicites et explicites des utilisateurs quand on l'utilise dans des conditions spécifiques [ISO01]
Fiabilité	La capacité du produit logiciel de maintenir un niveau spécifie de performance quand on l'utilise dans des conditions spécifiques [ISO01]
Utilisabilité	La capacité du produit logiciel d'être compréhensible, appris, utilisé et attirant pour l'utilisateur quand il est utilisé sous des conditions spécifiques [ISO01]
Efficacité	La capacité du produit logiciel d'offrir les performances adéquates par rapport à la quantité des ressources utilisées sous des conditions citées [ISO01]
Maintenabilité	La capacité du produit logiciel d'être modifiable. Les modifications peuvent être des corrections, amélioration ou adaptation du logiciel aux changements de l'environnement, des besoins et des spécifications [ISO01].
Portabilité	La capacité de transformer un logiciel d'un environnement à un autre environnement [ISO01]
Réutilisabilité	La capacité d'utiliser un module dans plusieurs produits logiciels [IEEE90]
Flexibilité	La capacité du produit logiciel de changer ses comportements en accord aves la situation courante afin de satisfaire ses objectifs [Kir06b]

2.3. Les sous-caractéristiques de QM4MAS

Selon le méta-modèle défini précédemment, une caractéristique est décomposée en plusieurs sous caractéristiques qui l'affectent. Cependant, la liste de sous-caractéristiques spécifie par le standard ISO/IEC 9126 n'est pas exhaustive. En effet, les sous-caractéristiques peuvent être adaptées aux particularités de certains produits logiciels. Cette adaptation prend la forme de l'ajout, la redéfinition de certaines sous-caractéristiques ou le rétablissement de relations entre les caractéristiques et les sous-caractéristiques [Mar15]. Dans notre modèle, nous avons adapté les sous-caractéristiques au paradigme agent.

Malgré l'absence d'un consensus concernant le concept agent, l'étude de ce domaine présentée dans le chapitre 02 nous a permis de tirer les concepts essentiels de ce nouveau paradigme. Ainsi, les concepts sur lesquels les SMA ont été fondés sont [Mar15] : l'autonomie, la réactivité, la pro-activité, l'interaction (la sociabilité), l'environnement, l'adaptabilité, la rationalité, le rôle (la spécialisation), la granularité et l'organisation. Ces concepts doivent être ajoutés aux sous-caractéristiques du modèle ISO/IEC 9126. Cependant, deux changements mineurs doivent être effectués sur les sous-caractéristiques du modèle standard [Mar15] :

- L'interopérabilité est supprimée du modèle afin d'être remplacée par l'interaction. En fait, l'interopérabilité est définie comme la capacité du logiciel d'interagir avec un ou plusieurs systèmes spécifiés [ISO01]. Cette définition est très semblable à la notion d'interaction des agents qui représente la capacité de s'influencer les uns les autres [Fer95]. Les deux notions exigent au moins la coexistence de plusieurs entités. Il est important d'éviter le chevauchement des concepts en éliminant ceux qui sont semblables. Nous avons choisi d'éliminer l'interopérabilité à cause de la richesse des formes d'interactions dans le domaine multi-agents par rapport à l'interopérabilité. En plus, l'interaction est plus adéquate à notre domaine d'étude.
- L'adaptabilité doit être redéfinie conformément au domaine multi-agents. Comme nous l'avons indiqué auparavant, l'adaptabilité représente une technique parmi les techniques de flexibilité de l'agent [Mar15]. A titre d'exemple, la réactivité (la capacité de répondre aux changements de

l'environnement [Dum10]) ou la sociabilité (la capacité de l'agent d'être affecté par les autres agents [Dum10]) sont des moyens permettant à l'agent de changer son comportement. L'adaptabilité représente aussi une technique qui permet aux agents d'être flexibles. Cependant, cette technique avancée, par rapport aux exemples sus mentionnés, fait référence à la capacité d'agent de changer *sa structure* ou *ses buts* selon sa situation [Rej05]. On note que l'apprentissage est considéré comme un type spécifique de l'adaptabilité (l'adaptabilité dynamique) [Rej05].

En plus des changements précédents, nous croyons que la modularité est une sous-caractéristique essentielle pour tous les types des logiciels [Mar15]. Malgré que cette sous-caractéristique soit négligée dans le modèle ISO/IEC 9126, elle affecte plusieurs caractéristique de la qualité comme la maintenabilité et la réutilisabilité. Dans le contexte des SMA, on peut considérer la notion d'agent comme une nouvelle forme de modularité. Cependant, l'existence de cette sous-caractéristique dans notre modèle est importante pour qualifier la structure interne de l'agent. En autre terme, la modularité dans le domaine des SMA est limitée à la structure modulaire de l'agent [Mar15]. Le tableau 4.2 présente les sous-caractéristiques reliées au domaine multi-agents.

Tableau 4.2 : Les définitions de sous-caractéristiques du modèle QM4MAS.

Sous-caractéristiques	Définitions
Autonomie	La capacité de l'agent de fonctionner sans l'intervention des autres (être humains ou autres agents) [Dum10]
Réactivité	La capacité de l'agent de percevoir son environnement et de répondre en temps opportun aux changements possibles [Dum10]
Pro-activité	La capacité de l'agent d'exécuter des comportements dirigés par les buts [Dum10]
Interaction (sociabilité)	La capacité de l'agent d'affecter les autres agents afin de satisfaire leurs buts [Dum10]
Adaptabilité	La capacité de l'agent de changer sa structure ou ses buts en fonction de sa situation actuelle [Rej05]
Rationalité	La capacité de l'agent de contrôler ses décisions pour générer des comportements optimaux [Car12]
Spécialisation (Rôle)	Une tâche affectée à un individu dans un ensemble de responsabilités données à un groupe [Cam11]
Granularité	Le degré de complexité de l'agent [Dum10]
Organisation	Une collection de rôles qui participant à des modèles

	institutionnelles systématiques d'interaction entre eux [Woo02b]
Environnement	L'espace dans lequel les agents interagissent avec les ressources et les autres agents [Wey05]
Modularité	Le degré de séparation des composantes d'un produit logiciel de façon à ce que le changement possible de l'une de ces composantes n'affecte pas les autres [IEEE90]

2.4. Les relations entre les caractéristiques et les sous-caractéristiques

Comme nous l'avons indiqué précédemment, un modèle de qualité est un ensemble de caractéristiques et de sous-caractéristiques reliées entre elles. Naturellement, nous devons relier les caractéristiques et les sous-caractéristiques spécifiées précédemment. La relation entre les deux niveaux de notre modèle est une relation d'affectation comme il est présenté dans le méta-modèle associé à notre proposition. Une sous-caractéristique influe sur les caractéristiques en relation avec elle. Notre travail dans cette étape est limité à l'étude des notions ajoutées par rapport au modèle standard. Bien entendu, les autres relations définies dans le modèle standard sont maintenues parce qu'un SMA est un logiciel avant toute considération [Mar15].

En accord avec Dromey [Dro95], l'établissement de relations entre les composants d'un modèle de qualité n'est pas une simple tâche. En fait, chaque sous-caractéristique peut affecter la plupart des caractéristiques. Par exemple, tous les attributs techniques d'un logiciel (les sous-caractéristiques) influent sur la maintenabilité de ce dernier. La présentation de toutes les relations possibles engendre un modèle complexe difficile à l'utilisation. En effet, nous avons choisi de présenter seulement les relations importantes afin de développer un modèle de qualité compréhensible [Mar15].

L'établissement de relations se fait à base de l'analyse de définitions citées auparavant. Nous présentons dans la suite de cette section, notre analyse des différentes caractéristiques afin de designer les sous-caractéristiques qui l'affectent.

- **La fonctionnalité :** cette caractéristique désigne la capacité du produit logiciel de satisfaire ses objectifs. Dans les SMA, les objectifs peuvent être divisés en deux classes : les objectifs individuels des agents et les objectifs du système. Un agent cherche à satisfaire ses *objectifs de manière optimale sans l'intervention des autres agents*. Les ressources consacrées à atteindre l'objectif représentent la base d'évaluation de la *rationalité* de l'agent. En plus, l'agent cherche à atteindre ses objectifs individuels sans l'intervention des autres agents, ce qui fait référence à *l'autonomie*. Par contre, l'agent doit *interagir* avec les autres agents afin de satisfaire les objectifs collectifs du SMA. En autre terme, c'est la sociabilité qui influe sur les fonctionnalités globales des SMA [Mar15].

- **La fiabilité :** un logiciel fiable est un logiciel opérationnel malgré l'apparition d'éventuelles erreurs. Le principe de cette caractéristique est de limiter la propagation des erreurs apparues dans une partie du logiciel sur les autres parties. Dans les SMA, on peut exploiter plusieurs techniques afin de limiter cette propagation. Tout d'abord, limiter *les interactions* entre les agents peut influer positivement sur la fiabilité des agents. En fait, en limitant les interactions, les erreurs apparues dans un agent n'affectent pas les autres agents. Cette limitation doit être complétée par la capacité de l'agent de se comporter tout seul sans l'intervention des autres. C'est *l'autonomie* de l'agent qui assure ce type de comportement. Sachant qu'un agent peut être aussi conçu comme une composition de plusieurs parties, il est important d'assurer la fiabilité inter-agents. Un bon niveau de *modularité* peut empêcher la propagation des erreurs entres les modules composant un logiciel. En plus, un agent peut exploiter *l'adaptabilité* pour changer son comportement ou ses objectifs, si des erreurs l'empêchent d'atteindre ses objectifs initiaux ou d'exécuter son comportement prévu [Mar15].
- **L'efficacité :** l'efficacité des logiciels indique l'utilisation de minimum de ressources. Dans les SMA, les ressources nécessaires sont généralement : le temps d'exécution, l'espace mémoire et la bande passante pour assurer la communication entre les agents. La communication entre les agents représente la base de *l'aspect social*. En plus, l'*organisation* représente un moyen pour minimiser l'interaction entre les agents. Par exemple, en utilisant la notion

d'organisation on peut connaitre à chaque instant *qui fait quoi*. Donc, il n'est pas nécessaire dans ce cas de lancer des communications pour rechercher un service donné. Par contre, l'absence de cet aspect exige de l'agent d'entrer dans un processus de communication lourd pour connaitre les agents pouvant lui rendre un service [Mar15].

La *granularité* de l'agent qui représente la complexité de ce dernier se manifeste sur deux façades : la complexité comportementales et la complexité de la structure de l'agent. La complexité comportementale peut influer sur le temps d'exécution des comportements des agents. Par contre, l'effet de la complexité de la structure, dont la complexité de connaissances de l'agent est incluse, est sur l'espace mémoire. Ce dernier est un niveau parmi les niveaux de l'*environnement* cités dans les chapitres précédents. En plus, l'*environnement* peut être constitué d'objets qui occupent à leur tour l'espace mémoire [Mar15].

De manière générale, c'est *la rationalité* de l'agent qui reflète son efficacité parce qu'elle représente le compromis entre les buts visés et les prix à payer [Mar15].

- **L'utilisabilité :** grâce à *l'autonomie* des agents, un logiciel basé agent peut atteindre ses objectifs sans l'intervention des autres. En conséquence, les SMA peuvent augmenter l'utilisabilité parce qu'un agent peut atteindre ses objectifs sans la nécessité de l'aide des utilisateurs. En plus, *l'adaptabilité* (dont l'apprentissage est une partie), est une technique qui augmente l'utilisabilité grâce à la possibilité de décrire des profils d'utilisateurs. L'utilisabilité peut aussi bénéficier de *l'interaction* parce que ce dernier peut exploiter les technologies sémantiques comme les ontologies [Mar15].
- **La maintenabilité :** la maintenabilité est directement attachée à la complexité du produit logiciel. Dans le contexte des SMA, la complexité peut être vue selon deux angles différents : la complexité des agents et la complexité des SMA. Au niveau des agents, la complexité est exprimée par *la granularité*. Cependant, un développement modulaire de l'agent contribue à la maîtrise de sa

complexité. Au niveau système, deux paramètres peuvent influer sur la complexité : le nombre des agents et les interactions entre eux. En autre terme, c'est *l'aspect social* qui influe sur la complexité d'un SMA. Cependant, *l'organisation* offre un cadre qui améliore la compréhensibilité du système. En effet, cet aspect diminue la complexité des interactions dans un SMA [Mar15].

- **La portabilité :** l'environnement est un composant essentiel de SMA. Le changement de l'environnement d'un SMA dépend de trois facteurs : l'agent, l'environnement et l'interaction agent-environnement. Naturellement, la faible *interaction* entre les agents et l'environnement est un indicateur de portabilité de SMA. En plus, si la conception du SMA prend en charge la possibilité de changer *l'environnement*, la portabilité des agents devient plus simple. En ce qui concerne l'agent, celui-ci peut simplifier la portabilité s'il est doté de capacités spécifiques comme *l'adaptabilité*. Cette capacité lui permet de changer sa structure ou ses objectifs en fonction du nouvel environnement [Mar15].
- **La réutilisabilité :** la réutilisabilité simplifie le développement des logiciels et augmente la productivité. Dans le domaine des SMA, la réutilisabilité prend deux formes : la réutilisation des composants d'un agent pour développer des nouveaux agents ou la réutilisation d'un agent dans un nouveau système. Naturellement, le développement d'un agent comme une entité monolithique complique la réutilisation des composants de ce dernier. Par contre, un développement *modulaire* permet la réutilisation de différents modules pour développer des nouveaux agents. Au niveau système, *l'interaction* faible entre les agents composant un système permet l'enlèvement d'un agent de son système initial pour l'intégrer dans un nouveau système sans le besoin de modifications profondes. A notre avis, l'interaction est homologue du concept de couplage étudié dans les autres paradigmes de programmation. En plus, un agent a plus de chance pour être réutilisé s'il est spécialisé dans une tâche spécifique. Il est connu en génie logiciel, que les entités spécialisées ont la chance pour être réutilisées par rapport aux composants

polyvalents. Cette spécificité des tâches est exprimée par la notion de *spécialisation* de l'agent. Pour intégrer un agent multi-rôles dans un nouveau système, nous confrontons l'une des deux situations difficiles : soit le changement du code de l'agent (un gaspillage d'effort) ou l'intégration de l'agent avec tous ses rôles dont on n'a pas besoin (gaspillage de ressources) [Mar15].

- **La flexibilité :** la flexibilité d'un agent désigne : la réactivité, la sociabilité et la pro-activité [Woo02b]. En fait, pour changer son comportement, un agent doit être capable de percevoir l'environnement et répondre aux éventuelles modifications de ce dernier (*la réactivité*). En plus, l'agent doit *communiquer* avec les autres agents afin de changer ses comportements en fonction de l'état des autres agents. Cependant, la prise en considération de l'état du système, ne doit pas empêcher l'agent de prendre l'initiative afin de satisfaire ses buts (*la pro-activité*). Au-delà de ses caractéristiques fondamentales, les agents peuvent exploiter *l'adaptabilité* pour produire des comportements plus flexibles en fonction de nouvelles situations [Mar15].

Le résultat de la précédente analyse est résumé dans le tableau 4.3. Il est facile de remarquer que la sociabilité est une sous-caractéristique pour toutes les caractéristiques proposées. A notre avis, la qualité des SMA peut être étudiée selon deux niveaux différents : le niveau micro (le niveau d'agent) et le niveau macro (le niveau système) [Mar15].

Tableau 4.3 : Les relations entre les caractéristiques et les sous-caractéristiques du modèle QM4MAS [Mar15].

Caractéristiques	Sous-caractéristiques
Fonctionnalité	Autonomie, Rationalité et Sociabilité
Fiabilité	Autonomie, Sociabilité, Modularité et Adaptabilité
Utilisabilité	Sociabilité, Granularité, Organisation, Environnement, Rationalité
Efficacité	Autonomie, Adaptabilité et Sociabilité
Maintenabilité	Modularité, Granularité, Organisation et Sociabilité
Portabilité	Adaptabilité, Environnement et Sociabilité
Réutilisabilité	Modularité, Spécialisation et Sociabilité
Flexibilité	Réactivité, Pro-activité, Adaptabilité et Sociabilité

2.5. Les métriques de QM4MAS

Le troisième niveau de notre modèle est constitué d'un ensemble des métriques. Le niveau des métriques est considéré comme le point faible du modèle ISO/IEC 9126. Cependant, cet avis ne prend pas en considération la classification des modèles de qualité selon leurs objectifs. En fait, le modèle standard sur lequel nous avons basé notre étude n'est pas un modèle basé-métrique qui propose l'évaluation de la qualité du logiciel. Suivant cette tendance, nous considérons l'évaluation de la qualité comme un objectif secondaire par rapport à la spécification de la qualité qui est notre objectif essentiel. Nous croyons que la proposition de métriques génériques applicables *directement* sur tous les SMA est un objectif intangible dans la phase actuelle de ce domaine. En fait, les métriques des SMA sont fortement attachées à plusieurs facteurs comme : le paradigme utilisé pour l'implémentation du système, le modèle de base de l'agent et le langage de développement du système [Mar15]. A titre explicatif, malgré que la cohésion et le couplage soient des métriques largement acceptées pour la modularité des logiciels, ces métriques évoluent selon le paradigme de programmation [Kra04, Hus09]. Dans le contexte de notre domaine, Alonso et *al.* [Alo09] indiquent, par exemple, que la mesure de la complexité comportementale proposée est fortement dépendante du paradigme de développement de SMA (le paradigme orienté-objet, les systèmes à base de connaissance,…etc.). Afin de résoudre ce problème, nous avons décidé de faire remonter le niveau d'abstraction de métriques proposées afin d'être génériques et indépendantes de spécificités de l'implémentation. Contrairement aux métriques qui sont des fonctions spécifiques, nous avons proposé dans ce niveau des métriques génériques appelées des *directives abstraites* [Mar15]. Ces directives décrivent de façon abstraite (indépendamment de l'implémentation) les possibles propriétés utilisées pour l'évaluation d'une sous-caractéristique. Les utilisateurs de notre modèle peuvent utiliser ces directives afin de produire des métriques réelles selon les particularités de la plateforme, de paradigme ou de modèle d'agent utilisé pour l'implémentation de SMA [Mar15]. Nous présenterons deux exemples de telle application dans le chapitre suivant. Dans la suite de cette section, nous présentons les différentes directives abstraites pour la mesure de chaque sous-caractéristique.

- **L'autonomie :** l'autonomie de l'agent peut être mesurée par rapport à deux propriétés:

1. **La demande de service :** cette métrique mesure le nombre de demande de services par rapport au nombre de comportements exécutés. Bien entendu, un agent autonome est un agent qui ne fait pas recours à la demande des services des autres agents [Mar15].
2. **La disponibilité de ressources :** cette métrique est basée sur la disponibilité de ressources pour atteindre un objectif donné. On peut donc estimer l'autonomie de l'agent par l'estimation de ressources nécessaire pour atteindre un but et les ressources disponible pour l'agent [Mar15].

- **La réactivité :** la réactivité d'un agent se manifeste sous forme de changement de l'état d'un agent à cause de la suspension d'un comportement afin d'exécuter un autre. En conséquence, on peut mesurer la réactivité d'un agent par :

1. **Le nombre de changement d'états :** cette métrique permet d'estimer la réactivité en calculant le taux de changement d'états. Le changement d'états est peut être détecté au cours de la suspension d'un comportement pour exécuter un autre [Mar15].
2. **Le temps de réponse :** un agent réactif est un agent capable de produire une réponse aux changements de l'environnement dans les meilleurs délais. Ainsi, cette métrique permet d'évaluer la réactivité de l'agent en calculant son temps de réponse aux changements détectés [Mar15].

- **La pro-activité :** la pro-activité désigne la capacité de l'agent de produire des comportements dirigés par des buts. Cette sous-caractéristique peut être mesurée par :

1. **Le taux de réalisation des objectifs :** cette métrique présente le taux de comportements qui atteignent leurs objectifs par rapport au nombre total des comportements exécutés [Mar15].
2. **Le temps moyen pour atteindre ses objectifs :** cette métrique présente le temps moyen pour atteindre les objectifs

par rapports au temps total consacré pour l'exécution de comportements [Mar15].

- **La sociabilité :** l'aspect social des SMA est basé sur l'échange de messages. Ainsi, on utilise cette propriété d'échange de messages pour mesurer les différentes métriques :

  1. **Le taux d'intention d'interagir :** un agent qui reçoit une demande de services peut, selon sa situation, accepter ou refuser la coopération. Cependant, un agent qui accepte de coopérer avec les autres agents ne peut pas toujours atteindre ses buts. En fait, plusieurs handicaps peuvent l'empêcher de coopérer effectivement, comme des contraintes temporelles (expiration du temps nécessaire pour donner la réponse) ou l'échec durant le processus de négociation. Calculant le taux de situations où l'agent ayant l'intention de coopérer par rapport au nombre total de situations de coopération est un indicateur pour connaitre les anomalies du fonctionnement du système global. En fait, si ce taux est bas, on peut conclure que l'empêchement d'atteindre les objectifs du système global est dû au caractère *égoïste* des agents qui ne cherchent pas à coopérer. Dans le cas contraire, on peut conclure que l'échec de coopération est une conséquence des facteurs externes de l'agent comme les protocoles de coopération ou les contraintes imposées [Mar15].
  2. **Le taux d'utilité de l'interaction :** nous croyons que l'interaction n'est pas un objectif en soi mais c'est un moyen pour assurer les objectifs du système global. En conséquence, l'interaction des agents importe peu par rapport à son utilité. Donc, dans cette métrique on propose le calcul du taux de situations d'interaction permettant aux agents d'atteindre leurs objectifs par rapport au nombre total de situations d'interaction [Mar15].
  3. **Le taux de compréhensibilité :** il existe plusieurs langages de communication dans le domaine des SMA. La diversité des langages de communication est, parfois, devenue un

handicap contre l'assurance de bonne interaction entre les agents. Naturellement, l'incompréhensibilité des messages influe négativement sur l'interaction des agents. Cette situation est plus probable dans les systèmes multi-agents ouverts. Nous proposons de calculer le taux de messages compris par les agents récepteurs par rapport au nombre total de messages échangés [Mar15].

- **La rationalité :** un agent rationnel est un agent qui cherche la satisfaction de ses buts de manière optimale. En effet, cet agent fait le compromis entre les buts recherchés et les ressources consommées. Si le prix d'un but est jugé plus important que les gains obtenus après la réalisation de ce dernier, l'agent rationnel l'abandonne. En conséquence, nous proposons de mesurer la rationalité en calculant la progression aux buts par rapport aux ressources consommées. Un cas particulier de ces ressources consiste au temps d'exécution [Mar15].
- **La spécialisation :** la spécialisation introduit la notion de rôle. Ainsi, on calcule dans cette métrique le nombre moyen de rôle pour chaque agent [Mar15].
- **La granularité :** la granularité représente la complexité de l'agent. Cette complexité peut prendre plusieurs formes, comme la complexité de comportements, la complexité de connaissances ou la complexité de la structure. La complexité de la structure est fortement attachée à la modularité de l'agent. Les deux autres formes de la complexité peuvent être calculées selon le paradigme d'implémentation de l'agent [Mar15].
- **L'organisation :** comme nous l'avons vu au chapitre 02, il n'existe pas une forme unique d'organisations. Cependant, les organisations se caractérisent par un ensemble de propriétés communes. Par exemple, une organisation est généralement composée de plusieurs sous-organisations. En effet, nous proposons des métriques applicables à la plupart des organisations :
  1. **Le nombre de sous-organisations :** une organisation est généralement composée d'agrégation de plusieurs sous-organisations [Fer04]. Ces sous-organisations peuvent être

des groupes, des communautés ou des niveaux dans une structure hiérarchique. Cette décomposition permet la maîtrise de la complexité de grandes organisations. En conséquence, nous utilisons cette métrique pour mesurer la complexité de l'organisation [Mar15].

2. **Le nombre moyen d'agents par sous-organisation :** une sous-organisation est composée de plusieurs agents. Il est clair que la compréhensibilité de la sous-organisation est en corrélation positive avec le nombre d'agents composant cette dernière. En conséquence, le nombre moyen des agents composant les sous-organisations est un indicateur de la complexité de ces dernières [Mar15].
3. **La richesse de relations dans l'organisation :** une organisation (ou une sous-organisation) n'est pas composée seulement d'un ensemble d'agents mais les relations entre les entités composant l'organisation est un élément important durant le développement des SMA. Malgré que la diversité de ces relations soit l'une des caractéristiques de la richesse d'expression des SMA, elle complique, en contre partie, la compréhensibilité du système. Nous proposons cette métrique (le nombre de types de relation entre-agents) pour la mesure de la complexité de l'organisation [Mar15].

- **L'environnement :** en accord avec Wooldridge [Woo02b], la complexité de l'environnement est liée aux attributs suivant : l'accessibilité, la déterminité, l'épisodique, la dynamicité et la continuité. Nous pouvons utiliser donc ces attributs pour la mesure de l'environnement [Mar15].
- **La modularité :** le couplage et la cohésion sont deux métriques bien-acceptées pour la mesure de la modularité d'un composant logiciel. Ces métriques sont, comme il a été indiqué ci-dessus, fortement liées au paradigme de développement logiciel. Nous pouvons donc adopter les métriques existantes dans la littérature pour la mesure de couplage et de cohésion de l'agent selon le paradigme utilisé pour l'implémenter [Kra04, Hus09].

A la fin de cette partie, nous notons que les différentes caractéristiques n'ont pas le même degré d'importance durant la spécification ou l'évaluation de la qualité des SMA [Mar15]. Les agents d'interface, par exemple, mettent l'accent sur l'utilisabilité et l'adaptabilité pour aider les utilisateurs dans leurs tâches. Par contre, les agents fonctionnant dans des systèmes critiques exigent la fiabilité comme une caractéristique fondamentale. De la même manière, les sous-caractéristiques affectent les caractéristiques de degrés différents. En conséquence, comme nous l'avons indiqué au début dans la spécification de notre méta-modèle, on utilise des poids qui expriment l'importance de chaque caractéristique, sous-caractéristique ou métrique. La spécification de valeurs de ces poids est laissée aux utilisateurs de notre modèle en fonctions du domaine d'application du système multi-agents, du modèle d'agent ou d'application elle-même [Mar15].

3. Vers l'application du modèle ISO/IEC 25010 pour les SMA

Le modèle standard ISO/IEC 25010 est un nouveau modèle de qualité présenté dans une série considérée comme la deuxième génération de modèles de qualité [Sur03]. Ce modèle représente une révision du modèle ISO/IEC 9126. En accord avec Behkamal et *al.*[Beh09], la diversité de modèles de qualité est une source de confusion. En plus, nous pensons qu'il est important d'exploiter les nouveaux modèles standards qui donnent une vue actualisée de la qualité logicielle. En fait, la révision du modèle ISO/IEC 9126, à travers la proposition d'un nouveau standard, a permis de traiter les insuffisances du premier modèle en exploitant le feedback de la communauté du génie logiciel. En conséquence, nous sommes persuadés de la nécessité de modéliser la qualité des SMA en prenant en compte les nouveautés apportées par le nouveau modèle standard. Dans cette partie, nous présenterons un premier pas vers le développement d'une extension du modèle ISO/IEC 25010 pour les SMA. Ce premier pas consiste à l'étude de l'applicabilité du nouveau modèle sur le paradigme agent. Cette étude s'articule autour de trois phases [Mar14a] : l'identification des caractéristiques essentielles des SMA, l'étude de la conformité des caractéristiques des agents avec les sous-caractéristiques du modèle standard et la proposition des éventuelles extensions nécessaires pour supporter les propriétés des SMA.

3.1. L'identification des caractéristiques des SMA

Afin d'identifier les caractéristiques des SMA, nous avons appliqué une autre approche différente de celle appliquée dans la première partie durant l'extension du modèle ISO/IEC 9126. Rappelant que nous avons identifié la plupart des caractéristiques essentielles citées dans la littérature dans notre premier modèle, nous avons choisi dans cette étude de limiter les caractéristiques étudiées. Ce choix est justifié, d'une part, par la nature de cette étude qui ne représente qu'un pas vers le développement d'un modèle complet pour les SMA et, d'autre part, pour caractériser notre proposition par l'extensibilité. En effet, l'utilisation d'une liste minimale de caractéristiques essentielles des SMA permet d'étendre le modèle généré pour supporter des caractéristiques plus spécifiques ou plus optionnelles.

Nous avons basé notre étude sur la notion d'agent faible abordée dans le chapitre 02. En fait, malgré l'évolution de définitions, le noyau de cette notion reste le même : un agent est une entité physique ou logicielle ***située*** dans un environnement et il se caractérise par ***l'autonomie***, la ***réactivité***, la ***pro-activité*** et la ***sociabilité*** [Woo95, Woo02a, Woo02b]. La figure 4.2 présente un méta-modèle de cet agent. Ainsi, les caractéristiques essentielles des SMA sont [Woo02a, Mar14a] :

- **L'autonomie :** la capacité de l'agent de fonctionner sans l'intervention des autres. L'agent possède son contrôle sur son état et son comportement ;
- **La situation** : la capacité de l'agent de détecter et d'agir sur son environnement;
- **La réactivité :** La capacité de l'agent de percevoir et de donner une réponse adéquate dans le temps exigé ;
- **La pro-activité :** La capacité de l'agent d'exécuter des comportements dirigés par les buts ;
- **La sociabilité :** La capacité de l'agent d'interagir avec les autres agents afin de réaliser ses buts.

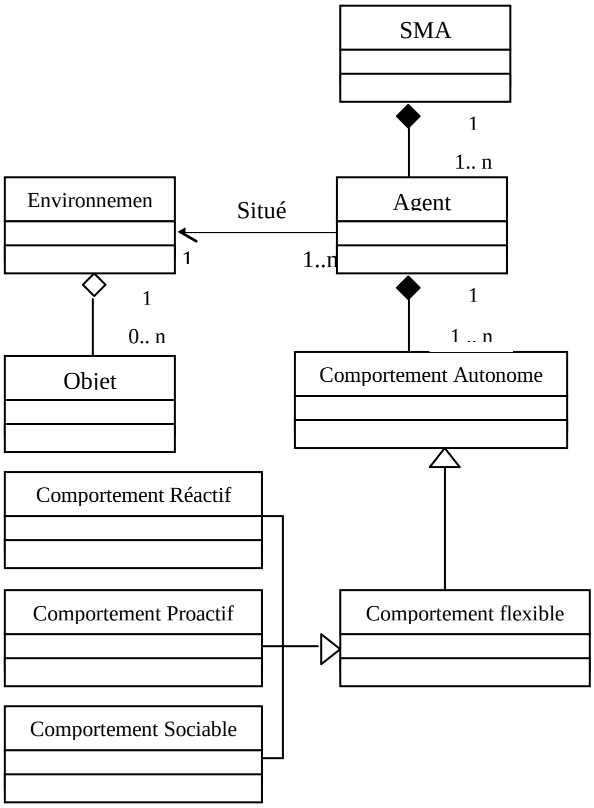


Figure 4.2 : Un méta-modèle de la notion de l'agent faible [Mar14a].

3.2. La conformité entre les caractéristiques des SMA et les sous-caractéristiques du modèle ISO/IEC 25010

Dans la deuxième phase de cette approche, nous allons étudier la conformité entre les caractéristiques des SMA citées ci-dessus et les sous-caractéristiques du modèle ISO/IEC 25010. Par la relation de *conformité,* on entend ici la capacité d'une sous-caractéristique du modèle standard de couvrir la définition de la caractéristique des SMA. Il semble clair que nous avons étudié directement les sous-caractéristiques du modèle standard sans étudier les caractéristiques de ce dernier à cause du caractère indépendant du paradigme de caractéristiques qui est évoqué précédemment. Afin de simplifier l'étude de la conformité, nous avons proposé, tout d'abord, la reformulation de définitions de différentes caractéristiques des SMA selon l'expression du modèle standards (*une sous-caractéristique est le degré de capacité de…*). En conséquence, les différentes caractéristiques deviennent [Mar14a] :

- **L'autonomie :** *le degré de capacité qu'a un agent pour contrôler son état et son comportement ;*

- **La situation :** *le degré de capacité qu'a un agent pour percevoir et agir sur son environnement ;*
- **La réactivité :** *le degré de capacité qu'a un agent pour percevoir les changements apparues et donner des réponses adéquates dans le temps exigé afin d'atteindre ses objectifs ;*
- **La pro-activité :** *le degré de capacité qu'a un agent pour prendre des initiatives afin de satisfaire ses besoins ;*
- **La sociabilité :** *le degré de capacité qu'a un agent pour interagir avec les autres afin de satisfaire ses objectifs.*

Après avoir identifié et reformulé les différentes caractéristiques des SMA, nous étudions la conformité entre ces définitions et les sous-caractéristiques du modèle ISO/IEC 25010. Cette étude est basée sur l'analyse de la définition de chaque caractéristique des SMA par rapport à toutes les sous-caractéristiques du modèle ISO/IEC 25010. Cependant, il est difficile de présenter l'analyse pour toutes les trente sous-caractéristiques. Donc, pour chaque caractéristique des SMA nous ne présenterons que les sous-caractéristiques pertinentes.

- **L'autonomie :** le standard ISO/IEC 25010 définit la confidentialité par *le degré de protection qu'a un produit software contre l'accès, accidentel ou intentionnel, non autorisé aux données* [ISO11]. Sachant que l'état d'un agent est représenté par des connaissances ou des informations, on peut conclure que cette sous-caractéristique couvre partiellement la notion d'autonomie. En fait, la protection d'accès non autorisé aux informations internes d'un agent lui donne un contrôle sur son état [Mar14a]. En plus, la responsabilité (*le degré de capacité d'attribution des actions uniquement à une entité* [ISO11)] est fortement attaché à la capacité d'un agent de contrôler son comportement [Mar14a]. En conséquence, les deux propriétés de l'autonomie de l'agent (le contrôle de l'état et le contrôle des comportements) sont couvertes respectivement par la confidentialité et la responsabilité.
- **La sociabilité :** la sociabilité représente la capacité de l'agent d'interagir avec les autres. L'interaction est la capacité de l'agent de coexister et de coopérer avec les autres afin d'atteindre ses buts. La coopération fait

référence à la notion d'interopérabilité définie dans le standard ISO/IEC 25010 par *le degré de capacité d'un produit logiciel de fonctionner en coopération avec les autres composants et produits logiciels* [ISO11]. En plus, la sociabilité des agents implique l'existence de ces derniers dans un environnement commun dans lequel ils partagent les ressources. Cette notion représente la coexistence dans le modèle standard [Mar14a].

- **La situation :** cette caractéristique désigne la capacité de l'agent d'interagir avec son environnement. L'environnement peut être modélisé, abstraitement, par l'ensemble des objets qui le composent comme il est présenté dans le méta-modèle de l'agent (Figure 4.2). En conséquence, l'interopérabilité semble une sous-caractéristique candidate pour couvrir cette notion. Cependant, une analyse profonde nous montre les limites de ce premier avis. Tout d'abord, l'interopérabilité est basée sur la capacité d'un produit logiciel d'interagir avec les autres produits logiciels. Par contre, l'environnement d'un SMA peut être composé d'objets logiciels ou physiques. En plus, l'environnement de l'agent ne limite pas le fonctionnement de ce dernier sur ses objets à la coopération seulement, mais l'agent peut exécuter des actions sur ces objets. En conséquence, nous pensons que cette caractéristique n'est pas couverte par le modèle standard [ISO11].
- **La réactivité :** la réactivité représente la capacité de l'agent de percevoir son environnement et de générer des réponses dans le temps adéquat. Cette caractéristique décrit deux propriétés essentielles : la perception de l'environnement et les contraintes temporelles qui conditionnent les réponses. En fait, il n'existe aucune sous-caractéristique du modèle standard qui couvre la première propriété. Par contre, les réponses à temps peuvent être assurées par la sous-caractéristique comportement temporel (*le degré de capacité qu'a un produit logiciel d'offrir des temps de réponse adéquats* [ISO11]) [Mar14a].
- **La pro-activité :** après l'analyse de différentes sous-caractéristiques du modèle ISO/IEC 25010 on ne trouve aucune parmi elles qui fait référence à la capacité du logiciel de prendre l'initiative.

Pour conclure, on peut remarquer que l'autonomie et la sociabilité sont totalement couverts par les sous-caractéristiques du modèle ISO/IEC 25010, la pro-activité et la situation ne sont pas couvertes par ce modèle tandis que la réactivité est partiellement couverte [Mar14a]. En conséquence, nous présenterons dans la section suivante, une proposition pour couvrir les insuffisances de ce modèle à l'égard de la modélisation de la qualité des SMA.

3.3. Les extensions proposées

Après l'analyse de différentes caractéristiques des SMA par rapport aux sous-caractéristiques du modèle ISO/IEC 25010 présentées précédemment, nous avons conclu que les propriétés suivantes ne figurent pas dans ce modèle [Mar14a] :

- La capacité de prendre l'initiative (la pro-activité) ;
- La capacité de percevoir et d'agir sur l'environnement (la situation) ;
- La capacité de percevoir les changements de l'environnement (une propriété de la réactivité).

Naturellement, nous devons ajouter ces propriétés au modèle standard afin de couvrir les spécificités des SMA. Cependant, cette étape doit être précédée par la reformulation de ces propriétés afin d'éliminer les chevauchements (par exemple, le caractère situé des agents et la réactivité indiquent les deux la capacité de percevoir l'environnement). En plus, cette étape permet de présenter des notions compatibles avec le standard ISO/IEC 25010. En conséquence, nous proposons les sous-caractéristiques suivantes [Mar14a]:

- **La pro-activabilité** : le degré de capacité qu'a un agent pour prendre l'initiative et générer des comportements orientés-buts ;
- **La perceptionnabilité :** le degré de capacité qu'a un agent pour percevoir son environnement et les changements apparues sur ce dernier ;
- **L'agissabilité :** le degré de capacité qu'a un agent pour exécuter des actions sur son environnement.

Finalement, ces sous-caractéristiques sont ajoutées au modèle ISO/IEC 25010 dans les emplacements jugés adéquats. Ainsi, la pro-activabilité est ajoutée à la caractéristique pertinence fonctionnelle. La perceptionnabilité et l'agissabilité sont considéré comme des sous caractéristiques de la compatibilité [Mar14a]. La figure 4.3 présente le nouveau modèle qu'on propose pour la modélisation de la qualité des SMA.

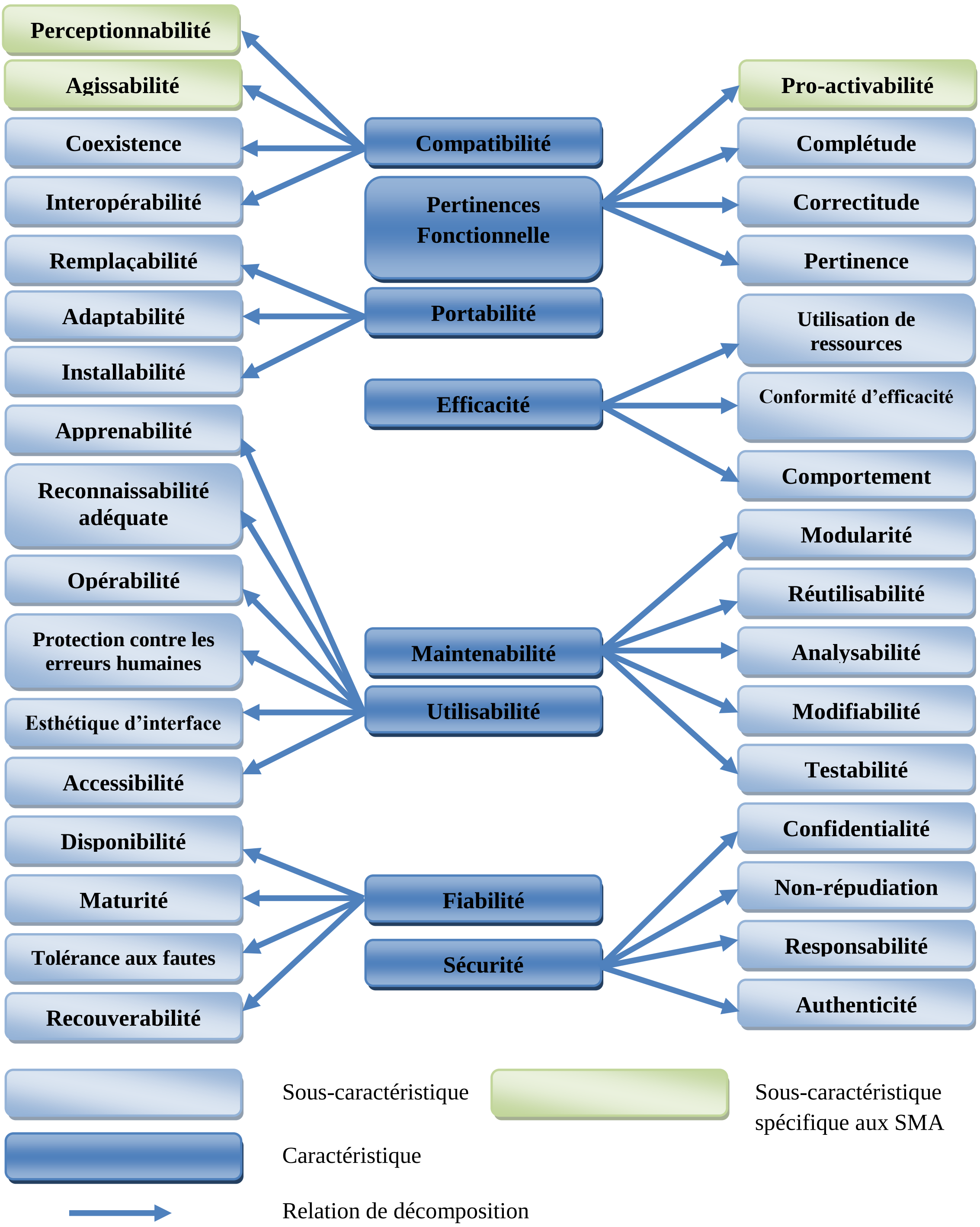


Figure 4.3 : Une version du modèle ISO/IEC 25010 adaptée aux SMA.

Nous considérons cette proposition juste comme un premier pas vers l'étude de la qualité des SMA en utilisant le nouveau modèle standard, parce que ce dernier est présenté dans le cadre d'une série complète de standards pour la qualité (ISO/IEC 25000). En conséquence, une étude profonde et complète doit prendre en compte l'ensemble de la série standard.

4. Conclusion

Ce chapitre est consacré à la présentation de deux contributions présentées dans le cadre de notre thèse. Ces deux contributions partagent le même objectif : ***le développement d'un modèle global de qualité des SMA en utilisant les modèles de qualité standards.*** Dans la première partie, nous avons présenté un modèle de qualité (appelé *QM4MAS*) basé sur le modèle ISO/IEC 9126 pour la spécification de la qualité des SMA. Par contre, nous avons étendu le modèle ISO/IEC 25010, dans la deuxième partie, pour supporter les SMA basés sur la notion d'agent faible. Considérant les trois objectifs de base des modèles de qualité (la spécification, l'évaluation et/ou la prédiction de la qualité), il semble indispensable de mettre nos modèles proposés dans leur cadre adéquat, essentiellement, pour la spécification de la qualité des SMA. L'évaluation de la qualité n'a été touchée que légèrement dans notre première contribution. Notre deuxième contribution n'a pas touché à cet objectif parce qu'elle représente seulement un pas vers l'application du nouveau modèle standard sur les SMA.

Nous avons indiqué dans ce chapitre la difficulté de la proposition des métriques génériques pour les SMA. En fait, les métriques sont fortement liées au paradigme de développement de SMA, du modèle d'agent ou du langage de programmation. Pour cette raison, nous avons proposé dans le troisième niveau seulement des directives abstraites pour la mesure de différentes sous-caractéristiques. Dans le chapitre suivant nous proposerons l'application du modèle QM4MAS sur deux plateformes multi-agents à travers la proposition d'un ensemble de métriques.

Chapitre 05

Applications du Modèle QM4MAS

« J'entends et j'oublie; je vois et je me souviens; je fais et je comprends »

1. Introduction

Le modèle de qualité QM4MAS a été développé pour la spécification de la qualité des SMA. En addition, ce modèle peut servir pour l'évaluation de la qualité de tels systèmes. Comme nous l'avons expliqué dans le chapitre précédent, la focalisation sur l'objectif de généricité nous conduit à la proposition de métriques abstraites appelées directives abstraites. Ces directives peuvent être utilisées pour la proposition de métriques concrètes en fonction de particularités de la plateforme, de modèle d'agent ou de langage de programmation. Dans ce chapitre, nous montrons l'application du modèle QM4MAS sur deux plateformes multi-agents. L'application du modèle proposé passe essentiellement par la définition de métriques concrètes. Nous notons que l'évaluation de ces métriques est automatique grâce à des outils développés pour cet objectif. Ainsi, nous présentons pour chaque plateforme l'outil développé. Cependant, nous commencerons ce chapitre par la présentation de démarche d'application du modèle proposé.

2. Démarche d'application du modèle QM4MAS

Malgré que le modèle QM4MAS a été développé pour être indépendant de plateformes multi-agents, nous devons prendre les spécificités de la plateforme cible durant l'application de notre modèle. L'application de notre modèle passe par deux étapes :

- **La sélection de sous-caractéristiques ciblées :** la première étape consiste à la sélection de sous-caractéristiques du modèle QM4MAS pertinentes à la plateforme cible. En fait, la couche de sous-caractéristiques est constitué, en plus de sous-caractéristiques du modèle de base ISO/IEC 9126, de notions importantes des SMA. Cependant, comme nous l'avons mentionné dans le chapitre 02, les notions introduites par le paradigme agent n'ont pas le même degré d'importance. En fait, certaines notions sont des notions fondamentales qui donnent naissance à ce nouveau paradigme. L'autonomie est un exemple de ces notions. Par contre, le paradigme agent présente plusieurs notions optionnelles, comme la rationalité et l'adaptabilité. En conséquence, chaque plateforme focalise sur des concepts donnés. La difficulté de mettre toutes les notions présentées dans ce paradigme de développement logiciel dans un framework *unifié* [Gue01, Bor07a] a même conduit les développeurs de certaines plateformes à ignorer certaines notions. A titre illustratif, la plateforme JADE ignore les notions d'organisation et de rôle. En plus, la conception d'une plateforme pour des applications spécifiques guide les développeurs à la focalisation sur des concepts au détriment des autres. Cette situation nous oblige de spécifier les sous-caractéristiques à évaluer afin d'éviter l'évaluation de sous-caractéristiques impertinentes. Par exemple, il n'est pas approprié de mesurer l'adaptabilité d'un SMA développé en utilisant une plateforme qui n'offre pas les mécanismes nécessaires pour la production des comportements adaptatifs.

- **La spécification de mesures :** après l'identification de sous-caractéristiques ciblées, nous devons spécifier des mesures pour l'évaluation de chaque sous-caractéristique. Cette tâche est basée essentiellement sur l'utilisation de directives abstraites afin de proposer les mesures. En d'autre termes, les mesures doivent être formulées sous la lumière de directives abstraites proposées et en prenant en considération les mécanismes spécifiques de chaque plateforme. Pour expliquer cette tâche nous présentons l'exemple de l'autonomie. Cette sous-caractéristique peut être mesurée en utilisant la disponibilité de ressources ou les demandes de services. Si la plateforme visée offre la possibilité de la modélisation explicite des ressources disponibles et les ressources nécessaires pour atteindre un but, nous pouvons spécifier la mesure de l'autonomie en utilisant ces deux paramètres. Sinon, nous devrons analyser les communications entre les agents pour identifier les demandes de services qui représentent un autre paramètre de l'autonomie. Sachant que le modèle proposé ne présente pas une liste exhaustive de métriques, nous pouvons proposer des nouvelles métriques spécifiques pour certaines plateformes ou réutiliser des métriques proposées dans la littérature.

L'application du modèle QM4MAS passe sous silence sur le niveau de caractéristiques. En fait, nous pensons que ce niveau n'est pas concerné par la spécification vis-à-vis les plateformes multi-agents. Les caractéristiques du modèle de qualité sont indépendantes des plateformes ou des architectures. Les utilisateurs de notre modèle peuvent cependant spécifier le niveau d'importance de chaque caractéristique en utilisant la pondération. Naturellement, l'affectation de la valeur nulle au poids d'une caractéristique indique la négligence de cette caractéristique.

Dans la suite de ce chapitre, nous appliquons le modèle proposé sur deux plateformes différentes : la plateforme JADE [Bel07] et la plateforme DIMA [Gue03]. Malgré que les deux plateformes soient parmi les plateformes les plus répandues dans ce domaine, elles offrent la possibilité de l'application du notre modèle sur deux cas différents. Le critère locomotif qui guide notre choix de deux plateformes est *la relation à un modèle spécifique d'agent.* En fait, la plateforme JADE est indépendante des modèles d'agent. Par contre, la plateforme DIMA est développée sur la base d'un modèle spécifique des agents hybrides (appelé le modèle DIMA). Bien entendu, nous consacrerons une section dans chaque cas pour présenter brièvement la plateforme cible.

3. Application du modèle QM4MAS sur la plateforme JADE

Dans ce premier exemple, nous présentons l'application du modèle QM4MAS sur la plateforme JADE [Mar15]. Avant de présenter les différentes métriques, nous présentons brièvement la plateforme cible.

3.1. La plateforme JADE

JADE (*Java Agent Development Framework*) est une plateforme de développement des SMA initiée par le département de recherche et développement d'Italie Télécom. Actuellement, elle est distribuée en tant que logiciel open source. En plus, elle est considérée parmi les plateformes les plus répandues pour le développement des SMA [Bel07].

La plateforme JADE est un middleware distribué et extensible écrit totalement en Java. En effet, la plateforme JADE a bénéficié de caractéristique du langage Java. En addition, cette plateforme offre un niveau d'abstraction permettant le développement des SMA sans la nécessité d'être un expert dans la théorie des agents. Un autre avantage de cette plateforme consiste en sa conformité aux spécifications de FIPA. La FIPA (*Foundation for Intelligent Physical Agents*) est une organisation à but nonlucratif pour la production des standards qui favorisent l'interopérabilité des agents logiciels hétérogènes [Bel07].

Conformément aux standards FIPA, la plateforme JADE est composée d'un ou plusieurs conteneurs (*Container*) dont un est un conteneur principal (*Main Container*). Le conteneur principal possède les responsabilités suivantes [Bel07] :

- La gestion de table de conteneurs (CT pour *Container Table*) : cette table enregistre toutes les adresses et les références de conteneurs composant la plateforme ;
- La gestion de la table de description des agents globale (GADT pour *Global Agent Descriptor Table*) : qui sauvegarde la liste de tous les agents existants dans la plateforme avec leurs états et adresses ;
- L'hébergement de deux agents spéciaux : le système de gestion d'agents (AMS pour *Agent Management System*) et le répertoire de facilitateur (DF pour *Directory Facilitator*) qui offrent respectivement les services de pages blanches et pages jaunes. Le répertoire de facilitateur offre le service de pages jaunes. Ce répertoire met en relation les agents avec les services qu'ils offrent. Par contre, le système de gestion d'agents permet l'authentification (c'est-à-dire l'offre d'un identificateur unique pour chaque agent) et le contrôle des agents (l'utilisation de la plateforme, la migration et le désenregistrement). Ce service enregistre les identificateurs des agents et leurs adresses.

Le conteneur principal peut devenir le goulot d'étranglement à cause de l'hébergement de la table de description des agents globaux (GADT) que tous les agents de la plateforme exploitent. Afin d'éviter cet inconvénient, chaque conteneur héberge une version cache de cette table. En plus, pour envoyer un message à un autre agent, l'agent émetteur commence la recherche de l'adresse de cet agent dans une table de description des agents locaux (LADT pour *Local Agent Descriptor Table*). La connexion au conteneur principal pour l'obtention de la version actualisée de la table de description des agents globaux (GADT) est conditionnée par l'échec de recherche d'un agent localement. Nous présentons dans la figure 5.1 l'architecture de la plateforme JADE en se basant sur la description présentée ci-dessus.

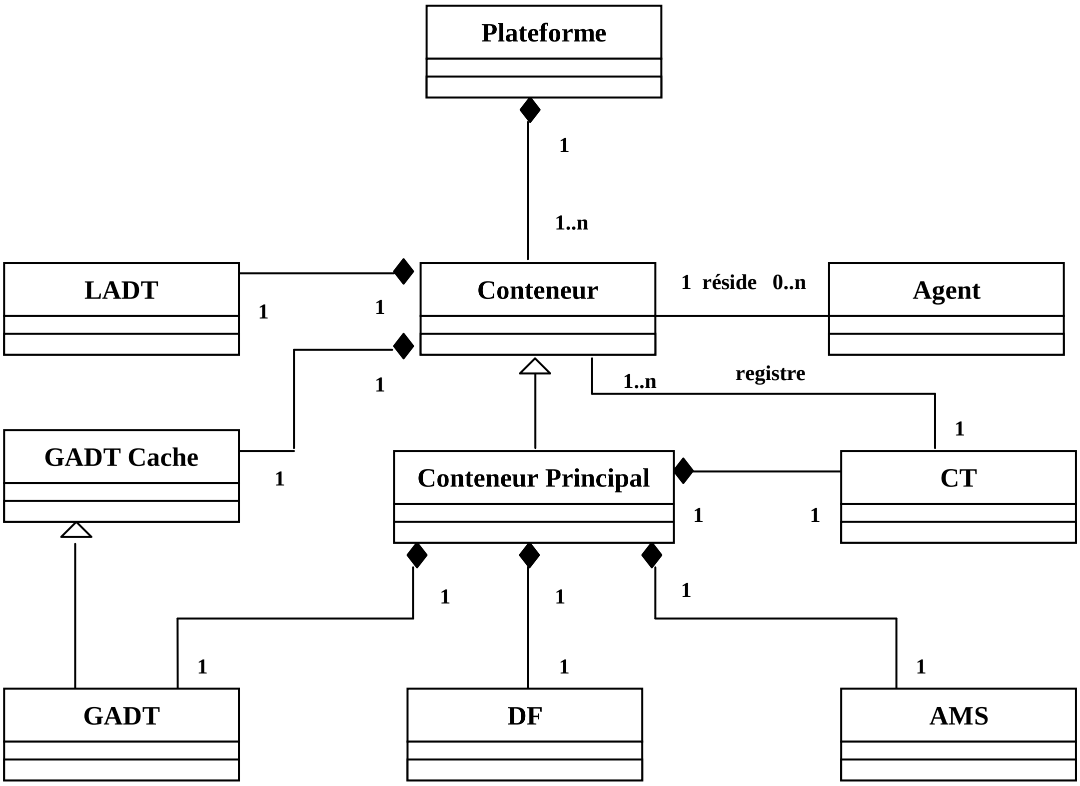


Figure 5.1 : L'architecture de la plateforme JADE.

Un agent JADE est une simple classe héritée de la classe ***jade.core.Agent*** et qui implémente la méthode ***setup().*** La figure 5.2. montre un exemple d'un agent qui affiche le message *(je suis agent).*

```
import jade.core.Agent;
public class Exemple_Agent extends Agent {
      protected void setup(){
          // Printout a welcome message
           System.out.println("je suis agent!");
         }
   }
```

Figure 5.2 : Code d'un agent JADE.

Bien entendu, les tâches de l'agent n'ont pas tous la simplicité apparue dans l'exemple précédent. En fait, la plateforme JADE propose l'implémentation de tâches des agents comme des objets de la classe ***jade.core.behaviours.Behaviour.*** Afin de permettre à l'agent d'exécuter un comportement, ce dernier doit être ajouté par la méthode ***addBehaviour().*** La tâche à exécuter par le comportement doit être implémentée dans la méthode ***action().*** En plus, la méthode ***done()*** exprime les conditions de terminaison du comportement.

La plateforme JADE offre trois catégories de comportements permettant le développement de comportements simples jusqu'aux comportements les plus complexes :

- **Les comportements simples :** cette catégorie permet le développement des comportements *one-shot* (c'est-à-dire des comportements à exécuter une seule fois), des comportements cyclique (c'est-à-dire des comportements à exécuter à chaque fois) ou des comportements génériques dont la fonction ***done***() est implémentée par le développeur.
- **Les comportements planifiés :** cette catégorie permet le développement des comportements exécutés dans des moments spécifiques. Le comportement ***WakerBehaviour*** s'exécute après un temps défini. Par contre, le comportement ***TickerBehaviour*** est un comportement répétitif qui s'exécute à chaque fois après une période d'attente.
- **Les comportements composés :** cette catégorie permet le développement de comportements composés de plusieurs comportements. Ces comportements peuvent s'exécuter séquentiellement, en parallèle ou selon un diagramme de machine à états finis.

Finalement, nous notons que les agents de la plateforme JADE communiquent à travers l'utilisation du langage FIPA-ACL.

3.2. La méthode de mesure

Selon l'exécution du logiciel, nous distinguons des métriques statiques et des métriques dynamiques. Les métriques statiques sont des métriques basées sur le code source du produit logiciel. Par contre, les métriques dynamiques sont calculées à la base de comportements du produit logiciel.

Malgré l'importance des métriques dynamiques, les métriques statiques sont les plus appliquées à cause de la difficulté relative de la mise en œuvre des métriques dynamiques [Tah10]. D'autre part, nous pensons que l'utilité des métriques statiques soit limitée à cause de la nature flexible des comportements des SMA. Il est difficile de collecter des informations utiles sur le comportement des agents à partir du code source. En conséquence, nous favorisons l'application de métriques dynamiques, quand il est possible, par rapport aux métriques statiques.

Plusieurs techniques ont été appliquées pour le calcul des métriques dynamiques [Gup08]. Généralement, nous faisons recours à des profils qui envoient les évènements captés afin de tracer l'exécution du logiciel. Ensuite, nous appliquons les métriques sur cette trace d'exécution. Cependant, la trace d'exécution donne une quantité d'informations importante ce qui nécessite l'application de techniques de compression afin de générer seulement des informations concises [Gup08].

Nous optons pour l'utilisation de la programmation orientée-aspect pour calculer les métriques dynamiques des SMA. Le paradigme aspect peut être utilisé pour capter seulement les informations pertinentes pour le calcul des métriques. En plus, grâce au développement des métriques indépendamment du logiciel, les métriques appliquées en utilisant ce paradigme n'affectent pas l'exécution du logiciel [Mar15].

La programmation orientée-aspect a été introduite en 1997 [kic97] afin d'augmenter la modularité des logiciels et maîtriser la complexité de ceux-ci. La figure 5.3 présente le principe de ce paradigme. Ainsi, la programmation orientée-aspect consiste en la séparation de préoccupations transversales de préoccupations métier. Les préoccupations métier sont développées comme le noyau du système. Par contre, les préoccupations transversales sont développées comme des aspects. Ensuite, les aspects s'intègrent au système dans les emplacements adéquats grâce au *tisseur (Waver)*.

Parce que la plateforme JADE est une plateforme basée-Java pour la programmation des SMA, nous pouvons appliquer *AspectJ* [Lad03] qui est une extension du langage Java pour supporter la programmation orientée-aspect. En fait, l'aspect représente l'unité centrale de l'*AspectJ*. Par rapport à la programmation orientée-objet, le paradigme aspect a introduit les concepts suivants [Lad03] :

- **Aspect** : est une entité logicielle qui capture une fonctionnalité transversale d'une application ;

- **Point de Jonction (joinpoint)) :** est un point dans l'exécution d'un programme (par exemple, appel à une méthode, à un constructeur ou lecture d'une variable) autour duquel un ou plusieurs aspects peuvent être ajoutés ;
- **Coupe (Crosscut) :** un constructeur du programme qui permet la sélection d'un point de jonction et collecter le contexte ;
- **Greffon (advice) :** un bloc de code définissant le comportement d'un aspect ;
- **Mécanisme d'introduction (inter-type declaration) :** permet d'étendre la structure du logiciel par l'ajout des attributs (des méthodes ou des variables).

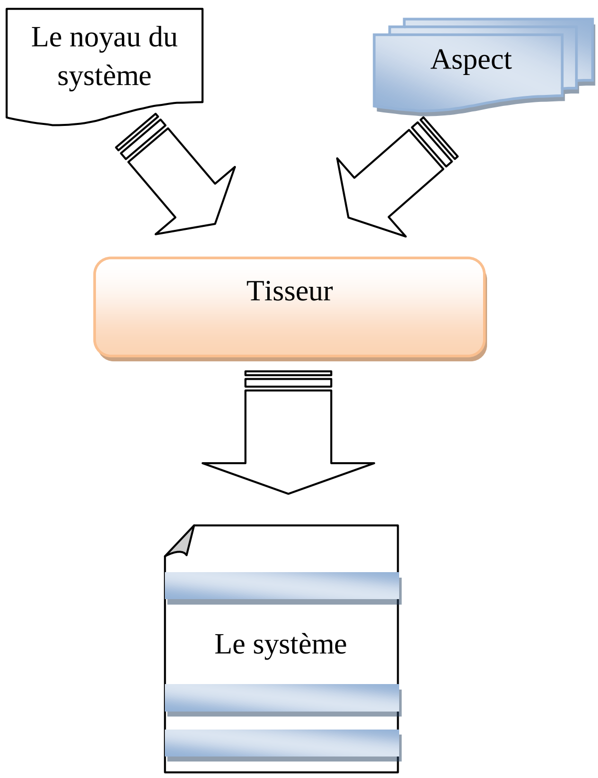


Figure 5.3 : Le principe de la programmation orientée-aspect.

La Figure 5.4 montre un exemple d'un code orienté-aspect. Dans la première partie de la figure (5.4 – A) nous trouvons le code d'une classe ordinaire écrite en Java qui englobe une méthode appelée ***`SimpleMethod()`*** qui affiche le message (*Je suis une méthode*). La partie (5.4 – B) présente un code aspect. Ce code spécifie une coupe (appelé ***`Point1()`***) en appelant la méthode ***`SimpleMethod()`***. Le greffon de cet aspect consiste en l'affichage du message (***Je suis un aspect***). En exécutant l'*AspectJ*, nous pouvons remarquer l'exécution de l'aspect avant l'exécution de la méthode.

```
public class SimpleClass{
   //Une classe Simple
   //...
   public void SimpleMethod(){
    //Une Methode Simple
   System.out.println("Je suis une methode");
  }
}
```

(A)

```
public aspect SimpleAspect {
  //Un aspect
    //Une coupe
  pointcut   Point1()   :   call   (public   void
   SimpleMethod ());
    //Un greffon
  before() : Point1() {
     System.out.println("Je suis un aspect");
     }
}
```

(B)

Figure 5.4 : Un exemple de code aspect.

3.3. Les mesures proposées

Afin d'appliquer notre modèle, nous devons identifier les sous-caractéristiques à mesurer. Ainsi, nous avons choisi les sous-caractéristiques suivantes : l'autonomie, la pro-activité, la réactivité, l'aspect social, la granularité et la spécialisation. En fait, notre objectif n'est pas la présentation de métriques pour toutes les sous-caractéristiques mais seulement la présentation de l'application de notre modèle.

Afin de proposer des métriques pour la plateforme JADE, nous avons étudié cette dernière pour identifier les constructeurs qui affectent les sous-caractéristiques présentées ci-dessus. Nous notons qu'en plus des métriques dynamiques mesurées en utilisant la programmation orientée-aspect, nous avons proposé des métriques statiques mesurées à partir de code source. Les métriques proposées sont [Mar15] :

- **La spécialisation :** comme nous l'avons indiqué dans les directives abstraites, la spécialisation peut être mesurée par le moyen de nombre de rôles joués par un agent. Cependant, la plateforme JADE n'offre pas le concept de rôle. En effet, nous avons proposé la mesure de la spécialisation par rapport au nombre de comportements implémentés par agent (ABA). Afin de normaliser les valeurs de cette métrique dans l'intervalle [0,1], nous proposons la formule de cette métrique comme suit : $ABA = \frac{N}{\sum_{i=1}^{N} B_i}$, tel que N est le nombre des agents et B_i est le nombre de comportements implémentés dans l'agent *i* [Mar15].
- **La granularité :** nous avons vu précédemment que la plateforme JADE propose des comportements simples ou des comportements composés. La granularité d'un agent peut être mesurée par la complexité comportementale de l'agent. En effet, nous avons proposé le taux de composition des comportements (RBC). Cette métrique est calculée en utilisant le code source de l'agent. Ensuite, nous construisons un arbre de comportements de l'agent. Chaque nœud de l'arbre est un comportement. Les nœuds fils représentent des comportements qui composent le père. Ainsi, les feuilles sont les comportements les plus simples. Considérant H_i la profondeur du nœud *i* et B le nombre de comportements,

  $RBC = 1 - \frac{B}{\sum_{i=1}^{B} H_i}$ [Mar15].

- **L'autonomie :** afin de mesurer l'autonomie dans les applications JADE, nous avons suivi la tendance basée sur les demandes de services. En fait, les agents demandent des services en envoyant des messages avec des performatives spécifiques (comme *CFP*, *REQUEST*, *QUERY* …etc.). Ainsi, nous avons utilisée le paradigme aspect pour capturer l'envoie de ce types de messages. Ainsi, l'autonomie peut être mesurée par le taux d'absence de demande de service (RLRS). $RLRS = 1 - \frac{RS}{EB}$, tel que *RS* est le nombre de demande de services et *EB* est le nombre de comportements exécutés. Pour assurer les exigences de la normalisation de valeurs dans l'intervalle [0,1], nous supposons que l'agent ne fait pas plus d'une demande dans chaque comportement exécuté. Dans le cas échéant, nous considérons le comportement exécuté comme un ensemble de comportements abstraits. Dans ce cas, EB devient le nombre de demandes de services dans le comportement exécuté [Mar15].

- **La réactivité :** la plateforme JADE offre plusieurs constructeurs pour permettre aux agents de changer leurs états, comme : *block(), restart(), changeStateTo(), doActivate(), doSuspend(), doWait() ,doWake(), restore()…etc*. En exécutant ces constructeurs, l'agent peut changer son état. En conséquence, nous pouvons utiliser le paradigme aspect afin de calculer le taux de changement de l'état de l'agent (RSC). $RSC = 1 - \frac{CB}{EB}$, tel que *CB* est le nombre d'instructions exécutées pour changer le comportement et *EB* est le nombre de comportements exécutés. Comme le cas de la mesure de l'autonomie, si l'agent a changé son état plusieurs fois au sein d'un seul comportement alors nous considérons ce dernier comme plusieurs comportements afin d'assurer la contrainte de normalisation [Mar15].
- **La pro-activité :** les comportements dans la plateforme JADE ne cessent d'être exécutés jusqu'à la réalisation de leurs buts. Généralement, le but d'un comportement est exprimé dans la fonction *done()*. Cette dernière retourne une valeur *true* si le comportement atteint son objectif. Ainsi, la métrique *taux de réalisation de buts* (*RAG*) est calculée par la formule $RAG = \frac{DB}{EB}$, tel que *DB* est le nombre de comportement qui ont atteints leurs buts et EB est le nombre de buts exécutés [Mar15].
- **La sociabilité :** la sociabilité peut être mesurée par plusieurs métriques :
  1. **Le taux d'intention d'interagir :** après la réception d'une requête (CFP, *REQUEST*, *QUERY* …etc.), l'agent peut répondre par *REFUSE* ou *AGREE* pour exprimer son intention de coopérer. Cependant, comme nous l'avons expliqué dans le chapitre précédent, cette intention ne reflète pas la coopération effective parce qu'il existe des raisons qui peuvent empêcher les agents d'atteindre l'objectif de coopération. Cette métrique mesure l'intention de la coopération. $RII = \frac{PIR}{SR}$, tel que PIR est le nombre de messages exprimant l'acceptation de coopération et SR est le nombre de messages exprimant la demande de services [Mar15].

2. **Le taux d'utilité de l'interaction :** une situation d'interaction est considérée utile si elle atteint son objectif. Généralement, un agent couronne la coopération par informer les autres par les résultats de son exécution à travers un message (*INFORM*). En conséquence, le taux d'utilité de l'interaction représente le nombre de message avec le performative *INFORM* par rapport au nombre total de demandes de services [Mar15].
3. **Le taux de compréhensibilité :** si l'agent ne comprend par un message reçu, il répond par le message *NOT_UNDERSTOOD.* En effet, le taux de compréhensibilité représente le nombre de ces messages par rapport au nombre total de messages reçus [Mar15].

3.4. Outil implémenté et étude de cas

Dans le cadre de notre travail, nous avons développé une application qui permet la mesure automatique de différentes métriques citées auparavant. Cette application consiste essentiellement en une bibliothèque d'aspects qui captent le comportement de SMA et calculent les différentes mesures [Mar15]. La figure 5.5 présente une partie d'un aspect parmi les aspects développés. Cet aspect est utilisé pour capter le changement d'états d'un agent. Cette information sera utilisée pour le calcul du niveau de la réactivité de l'agent.

```
public aspect ReactivityMeasure {
/* Un aspect pour la mesure de la reactivité */
  pointcut ChangeBehaviourState() :
     (call (* *.block())||
      call (* *.restart()) ||
      call (* *.changeStateTo(*))||
       /* Autres constructeurs pour le
          changement de l'état d'un agent */
             ) ;
  before() : ChangeBehaviourState(){
       /* Ici on calcule le nombre de fois qu'on
          change l'état de l'agent */}
     }
```

Figure 5.5 : Un aspect pour calculer la réactivité d'un agent.

Comme nous l'avons mentionné ci-dessus, l'un des avantages de l'utilisation de la programmation orientée-aspect pour la mesure de différentes métriques est l'indépendance totale entre l'outil de mesure et le système à mesurer. En fait, notre outil peut être appliqué sur n'importe quel SMA implémenté en JADE sans la nécessité de changer son code. Afin de montrer cet avantage, nous avons validé notre outil sur un code qui existe dans le web. L'exemple choisi est l'exemple de *l'agent de portail* présenté dans le site web *Codes-Sources*[1]. Cet exemple est composé de trois agents : l'agent portail, l'agent vendeur et l'agent acheteur. Dans les figure 5.6, 5.7, 5.8, 5.9 nous avons présenté les comportements individuels de différents agents composant le système ainsi que le comportement collectif afin d'améliorer la compréhensibilité de l'exemple.

Dans la figure 5.6, le comportement de l'agent portail a été présenté. Cet agent commence son comportement par la création d'une interface graphique et l'exécution de deux autres agents. Ensuite, il passe à l'état d'attente jusqu'à la réception d'un message. Quand l'agent portail reçoit le message, il affiche la liste des produits reçue et il passe à l'état final.

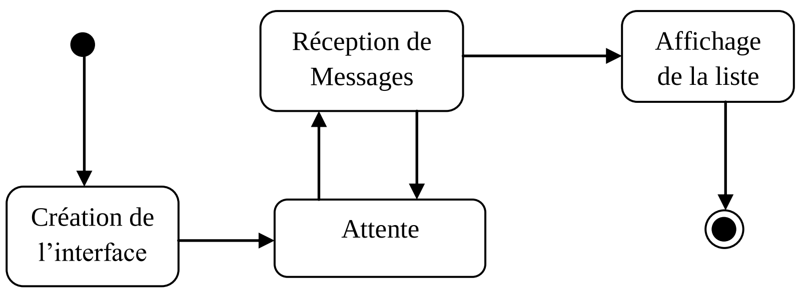


Figure 5.6: Description du comportement de l'agent *portail.*

L'agent acheteur commence son comportement par la demande de la liste des produits de l'agent vendeur. Ensuite, il passe à l'état d'attente jusqu'à la réception de la réponse de l'agent vendeur. A la réception de la liste des produits, l'agent acheteur la transmet à l'agent portail puis il passe à l'état final. La figure 5.7 présente son comportement.

[1] Le lien de l'exemple : http://www.javafr.com//code.aspx?ID=49974

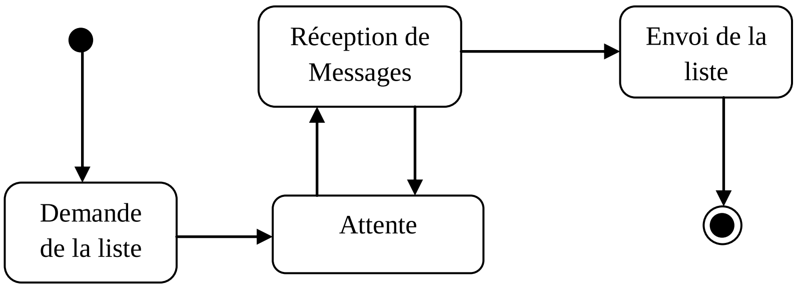


Figure 5.7: Description du comportement de l'agent *acheteur*.

L'agent vendeur commence son comportement par le passage à l'état d'attente (Figure 5.8). A la réception de la liste des produits, l'agent vendeur l'envoie et passe à l'état final.

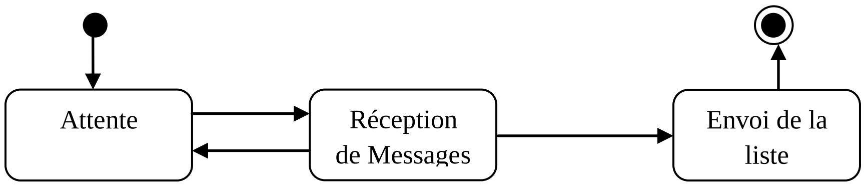


Figure 5.8: Description du comportement de l'agent *vendeur*.

Le diagramme d'interaction présenté dans la figure 5.9 représente le comportement collectif de différents agents composant le système. Ainsi, l'acheteur commence par l'envoi d'une requête pour demander la liste des produits. Le vendeur peut répondre par l'envoi de message *Not_Understood* si le message est illisible ou l'envoi de la liste. Finalement, l'acheteur envoie la liste des produits reçue à l'agent portail.

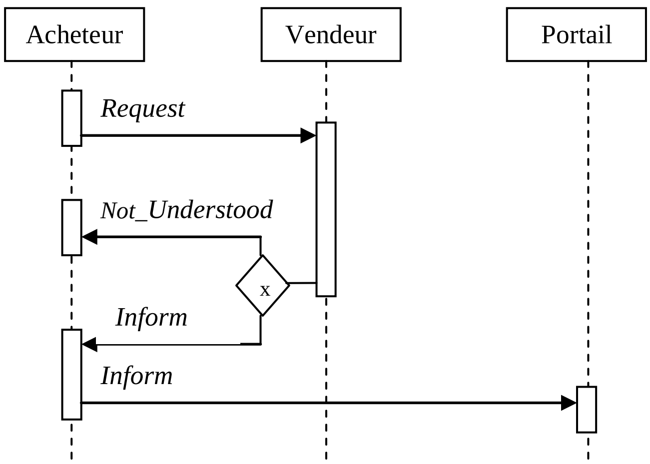


Figure 5.9: Le diagramme d'interaction du système.

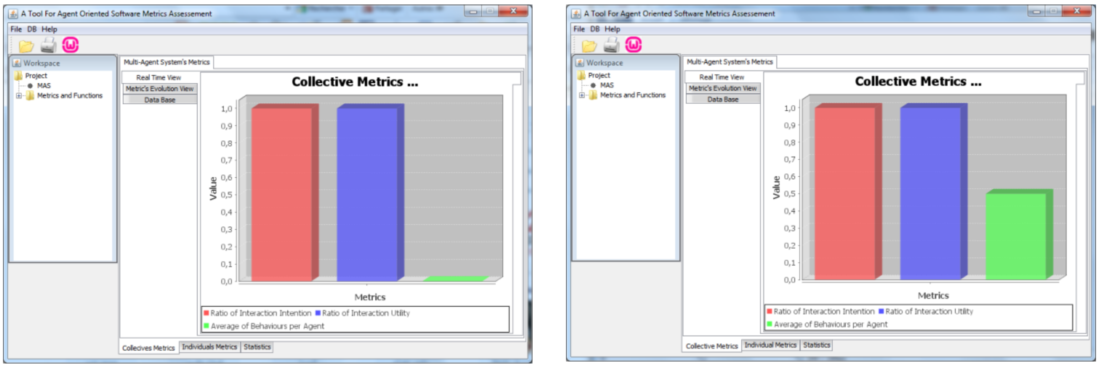
La figure 5.10 présente une partie du comportement de l'agent portail écrit en JADE. Cet agent est composé de deux comportements : *la création de l'interface* et la *production de la liste de produits*.

```
public class Portal extends Agent {
  private MyInterface Interface1 = new
      MyInterface();
  protected void setup() {
     /* Les comportements de l'agent Portail */
    addBehaviour(new CreateInterface());
    addBehaviour(new ProductListReception());}
      /*  On  implémente  ici,  les  different
         comportements des agents */
}
```

Figure 5.10 : Le squelette du code de l'agent *Portail*.

En exécutant l'exemple, notre outil commence la mesure de différentes métriques en accord avec leurs types : des métriques collectives et des métriques individuelles. Les métriques collectives sont des métriques communes pour tous les agents. Par contre, les métriques individuelles sont des métriques spécifiques pour chaque agent. Notre outil permet la présentation de résultats sous plusieurs formes : une présentation instantanée de différentes métriques, une présentation de l'évolution des métriques au cours de l'exécution, un rapport textuel et la sauvegarde dans une base de données [Mar15].

La présentation instantanée permet la présentation de la valeur de métriques dans l'instant courant de l'exécution. La figure 5.11 montre ce cas. Dans la partie (5.11 – A), la figure montre les métriques collectives au démarrage de l'exemple. Cette figure montre les trois métriques : *le taux d'intention d'interagir* (en rouge), *taux d'utilité de l'interaction* (en bleu) et *le nombre moyen de comportements par agent* (en vert). Parce que ces métriques sont mesurées au démarrage de l'exemple, elles représentent les valeurs par défaut (les deux premières métriques égales à 1 et la deuxième à 0). Après l'exécution de l'exemple, l'agent portail a été crée. Comme cet agent implémente deux comportements, la valeur de la métrique (*le nombre moyen de comportement par agent)* devient 0.5 [Mar15].



(A) **(B)**

Figure 5.11 : La présentation instantanée de métriques collectives.

L'outil développé permet aussi la présentation de l'évolution des métriques. La figure 5.12 présente l'évolution de trois métriques individuelles de l'agent *Acheteur : le taux d'absence de demande de services* (en rouge), *le taux de réalisation de buts* (en bleu) et *le taux de changement de l'état de l'agent* (en jaune). Nous expliquons ici, *le taux d'absence de demande de services (RLRS).* En fait, l'agent *Acheteur* possède trois comportements : un comportement pour la demande de la liste des produits, un comportement pour la recevoir et un comportement pour envoyer la liste à l'agent *Portail*. Durant l'exécution du premier comportement, l'agent demande la liste des produits. Ainsi, la métrique RLRS devient 0 ($RLRS = 1 - \frac{1}{1}$) à l'instant 43 secondes. A l'instant 44 secondes, l'agent exécute son deuxième comportement et la métrique RLRS devient 0.5. En attendant la réception de la liste des produits, l'agent *Acheteur* passe à l'état d'attente jusqu'à l'instant 66 secondes où il a été réveillé pour ré-exécuter le comportement de réception de la liste. En conséquence, la métrique RLRS devient 0.66. Finalement, l'agent exécute le comportement responsable de l'envoi de la liste à l'agent *Portail* et sa métrique RLRS devient 0.75 [Mar15].

Nous notons que l'outil offre aussi les possibilités de sauvegarder l'historique des métriques dans une base de données ou la génération d'un rapport textuel [Mar15].

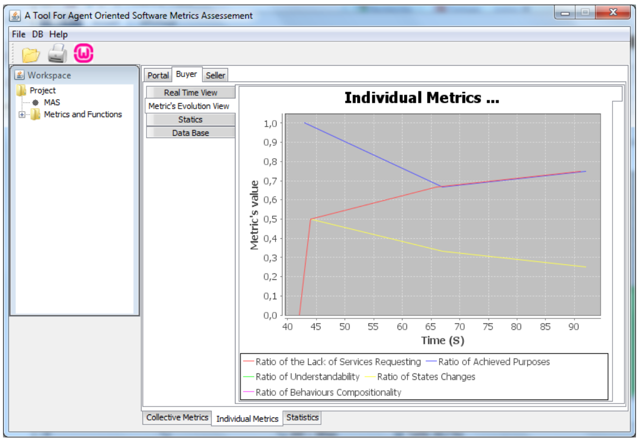


Figure 5.12 : L'évolution des métriques de l'agent *Acheteur*.

4. Application du modèle QM4MAS sur la plateforme DIMA

La partie précédente montre l'application du modèle QM4MAS sur la plateforme JADE. Cette plateforme se caractérise par son indépendance de modèles d'agents. Dans cette partie, nous appliquons notre modèle de qualité sur la plateforme DIMA [Gue03] qui est spécifique au développement des agents basés sur l'architecture DIMA.

4.1. La plateforme DIMA

La diversité de définition du concept agent a conduit à l'apparition de plusieurs modèles. Le modèle DIMA (pour Développement et Implémentation des Systèmes Multi-Agents) est l'un des modèles les plus connus dans le domaine des SMA [Gue03]. Comme il est montré dans la figure 5.13, un agent DIMA est un composant simple ou composé qui interagit avec son environnement. L'environnement représente toutes les autres entités dont les agents inclus.

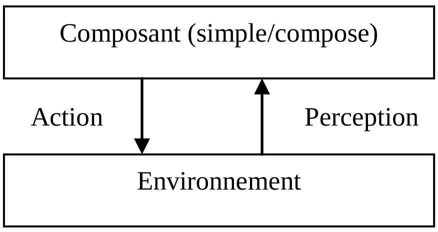


Figure 5.13 : L'architecture de l'agent DIMA [Gue03].

Sachant que l'agent est une entité autonome, l'agent DIMA peut être considéré, essentiellement, comme une entité proactive. Cette dernière modélise l'autonomie et la pro-activité de l'entité *agent*. L'entité proactive est une entité composée de (Figure 5.14) [Gue03] :

- Le but de l'entité qui est décrit implicitement ou explicitement dans la méthode *IsAlive() ;*
- La méthode *step()* utilisée pour la spécification et le contrôle du comportement de l'agent afin d'atteindre son objectif.

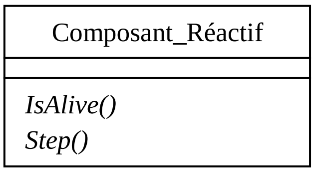


Figure 5.14 : La classe modélisant le composant proactif [Gue03].

La plateforme DIMA a été conçue comme une extension de la programmation orientée-objet. En effet, DIMA est développée comme un API du langage *Java*. Cet API consiste en une bibliothèque riche permettant le développement de simples agents jusqu'à des agents plus complexes comme les agents communicants ou les agents adaptatifs. Grâce au principe de la modularité sur lequel la plateforme DIMA a été fondée, on peut enrichir les composants proactifs afin de développer des agents plus complexes [Gue03].

Les agents DIMA ne peuvent pas être considérés seulement comme des *agents hybrides*. En fait, l'agent DIMA peut être considéré comme un modèle ouvert permettant le développement de plusieurs types d'agents [Gue03]. En fait, la plateforme DIMA permet le développement de plusieurs types d'agents hétérogènes. Tout d'abord, il est possible de développer des agents réactifs, cognitifs ou hybrides en utilisant la plateforme DIMA. En plus, nous pouvons développer des agents adaptatifs.

Les comportements adaptatifs sont assurés en introduisant le niveau méta-comportement proposé dans cette architecture [Gue03]. Ce niveau a été introduit pour représenter explicitement et séparément le contrôle permettant aux agents de produire des comportements autonomes et adaptatifs selon la situation courante et l'objectif visé. Nous notons qu'une partie de l'hétérogénéité des agents DIMA est due à la richesse de techniques utilisées pour le développement de ce niveau. En fait ce module peut être décrit en utilisant l'ATN, le raisonnement à base de connaissance ou les systèmes à base de connaissances [Gue03].

Concernant les interactions entre les agents, la plateforme DIMA supporte la communication basée sur le langage KQML et le langage FIPA-ACL [Gue03]. En plus, la plateforme donne la possibilité de la définition de messages de communications spécifiques.

Plusieurs raisons ont motivé l'utilisation de cette plateforme pour l'application de notre modèle. Tout d'abord, la modularité sur laquelle la plateforme repose permet la maîtrise de développement des agents complexes. En plus, la richesse de techniques de raisonnement supportées par cette plateforme augmente son applicabilité. Finalement, la plateforme est en pleine évolution. En fait, une version de la plateforme DIMA pour le développement des SMA tolérants aux pannes est en cours de développement [Fac15].

4.2. Les mesures proposées

Suivant la démarche de l'application citée dans la première partie de ce chapitre, nous devons cibler les sous-caractéristiques à mesurer avant la proposition de mesures concrètes. En conséquence, nous avons choisi la mesure de sous-caractéristiques suivantes : l'autonomie, la pro-activité, la réactivité, l'aspect social et la rationalité. Ensuite, nous avons proposé des métriques pour chaque sous-caractéristique. Bien entendu, nous avons appliqué le paradigme aspect pour la mesure de ces métriques dynamiques. Cependant, le modèle de la qualité proposé ne posant aucune contrainte sur le niveau des métriques, nous avons enlevé la contrainte de normalisation de valeurs de meures. En fait, la normalisation de valeurs a le seul objectif d'aider à l'interprétation des valeurs obtenues. En conséquence, nous pouvons, comme il est appliqué dans cette partie, enlever cette contrainte et donner la signification de chaque métrique.

Après l'analyse de la plateforme DIMA, nous avons identifié un ensemble de constructeurs qui influent sur les sous-caractéristiques visées. En conséquence, nous avons proposé les mesures suivantes :

- **L'autonomie :** comme dans le cas de la mesure de l'autonomie des applications JADE, nous mesurons l'autonomie des agents DIMA par référence aux demandes de services. Cependant, nous calculons ici le nombre moyen de demandes de services par comportement exécutés (ARS). Naturellement, cette mesure est en relation négative avec l'autonomie.
- **La pro-activité :** cette sous-caractéristique est mesurée, comme dans le cas de JADE, par le taux de buts réalisés par rapport aux comportements exécutés. La réalisation de buts dans les agents DIMA est peut être capturée par la valeur retournée par la méthode *IsAlive().* Nous pouvons calculer le nombre de comportements exécutés par le nombre d'exécution de la méthode *step()*.
- **La réactivité :** un comportement d'un agent peut être en exécution ou en état de suspension. Un agent peut suspendre un comportement pour attendre un évènement (donc, il passe à l'état d'attente) ou pour exécuter un autre comportement jugé plus important. En conséquence, cette suspension de comportements est un indicateur de réactivité de l'agent. Ainsi, nous mesurons la réactivité de l'agent par le nombre moyen de comportements suspendus par rapport au nombre de comportements exécutés. La suspension de comportement est le résultat de l'exécution de fonctions *wwait()* ou *wwait(long n).*
- **La rationalité :** on peut mesurer la rationalité par :
    1. **Le nombre moyen de comportements exécutés :** afin de réaliser ses buts, un agent exécute des comportements. Cependant, l'exécution de comportements n'est pas *gratuite*. En exécutant des comportements, l'agent consomme des ressources différentes (comme le temps, l'espace mémoire, …etc.). En conséquence, nous pouvons calculer la rationalité de l'agent en calculant le nombre de comportements exécutés par agent.
    2. **L'accélération de la réalisation des buts :** un agent rationnel est un agent qui contrôle son exécution par rapport à la satisfaction de ses buts. Naturellement, un agent rationnel peut suspendre un comportement s'il juge que la quantité de ressources consommées est plus grande que le but visé. Dans cette mesure, nous calculons

la progression des buts par rapport aux ressources consommées (GAA). $GAA = \frac{NBR_{t'} - NBR_t}{RC_{t'} - RC_t}$, tel que $NBR_{t'}$ (respectivement NBR_t) représente le nombre de buts réalisés à l'instant *t'* (respectivement *t*) et $RC_{t'}$ (respectivement RC_t) est la quantité de ressource consommée à l'instant *t'* (respectivement *t*). Selon la nature des ressources, nous pouvons distinguer plusieurs formes de cette mesure. Généralement, nous pouvons calculer cette mesure par unité de temps ou par comportements exécutés.

- **L'aspect social :** en plus de trois mesures de la sociabilité définies pour la plateforme JADE que nous pouvons appliquer ici, nous proposons aussi pour la plateforme DIMA les mesures suivantes :
  1. **Le niveau d'hétérogénéité :** la plateforme DIMA propose plusieurs types hétérogènes d'agents. Il semble clair que l'hétérogénéité de SMA génère des difficultés pour la compréhension du système durant la phase de maintenance. En plus, il n'est pas simple de mettre en coopération des agents hétérogènes. En conséquence, nous proposons cette métrique qui représente le nombre de types d'agents implémentés dans le système par rapport au nombre total de types d'agents proposés dans la plateforme.
  2. **Le moyen de la charge de communication :** la communication entre les agents est un processus lourd qui consomme beaucoup de ressources comme le temps d'exécution, l'espace mémoire pour la sauvegarde de messages, la bande passante et l'énergie dans certains types de systèmes. Malgré que la communication est le moyen de l'interaction, il est important de rationaliser le recours à la communication. Cette mesure propose l'estimation de la charge de communication par le nombre moyen de messages envoyés.
  3. **Le taux de l'utilisation de communications basées sur des langages-standards :** la plateforme DIMA, comme nous l'avons mentionné précédemment, propose plusieurs langages de communication. En fait, cette diversité de langages de communication a son impact négatif sur l'interopérabilité surtout

dans le cas des systèmes ouverts. En conséquence, l'utilisation de langages standards comme FIPA-ACL est plus utile. Cette métrique est le nombre de messages envoyés qui sont basés sur des langages standards par rapport au nombre total de messages envoyés.

4.3. Outil implémenté et étude de cas

Afin de mesurer automatiquement les applications basées sur la plateforme DIMA, nous avons développé un outil de mesure. La figure 5.15 présente l'architecture de notre outil. Naturellement, les applications à mesurer représentent une partie indépendante de l'outil. L'outil consiste essentiellement en un ensemble de métriques développées en tant qu'aspects en utilisant *AspectJ*. Grâce au tisseur, nous pouvons capter les évènements pertinents au calcul des mesures en utilisant les différents aspects. Ces évènements représentent la trace d'exécution utilisée ensuite pour mesurer les différentes mesures. Finalement, les résultats peuvent être sauvegardés dans une base de données, représentés graphiquement ou dans un rapport textuel.

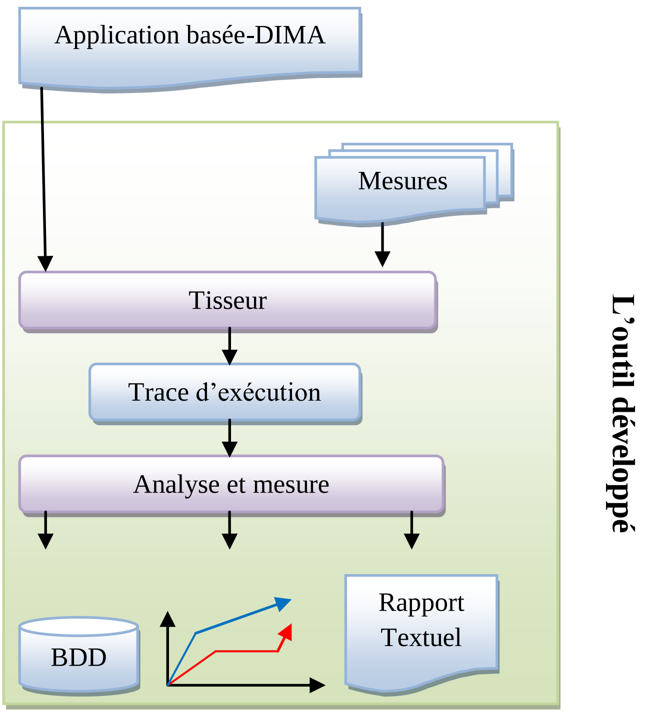


Figure 5.15 : L'architecture de l'outil développé.

L'outil est validé sur l'exemple d'enchère. Ce système multi-agents est composé de deux types d'agents (Acheteur et Vendeur) hérités de la classe des agents communicants (*BasicCommunicatingAgent*) qui est aussi héritée de la classe des agents réactifs (*BasicReactiveAgent*) (Figure 5.16).

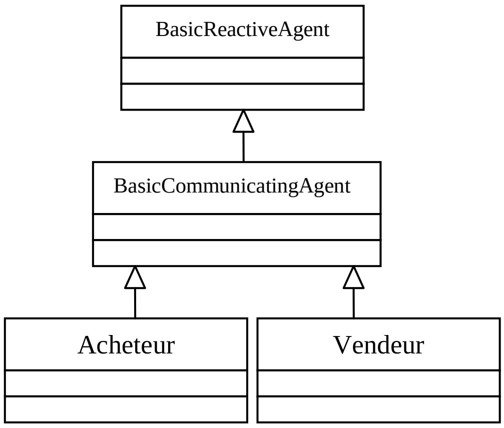


Figure 5.16: Diagramme de classe de l'exemple d'enchère.

Les deux figures 5.17 et 5.18 représentent les comportements des agents du système. Dans la figure 5.17, nous pouvons remarquer que l'agent *Acheteur* commence son comportement par l'attente de messages. En fonction du message reçu, l'agent peut passer à l'évaluation de l'appel d'offre si le message est un CFP, l'achat de l'objet si le message est une acceptation de proposition ou à l'état final si le message est une refuse de sa proposition. En évaluant l'appel d'offre, l'acheteur peut passer à l'état final si l'appel n'est pas intéressant ou l'envoi d'une proposition et le passage à l'état d'attente de réponse.

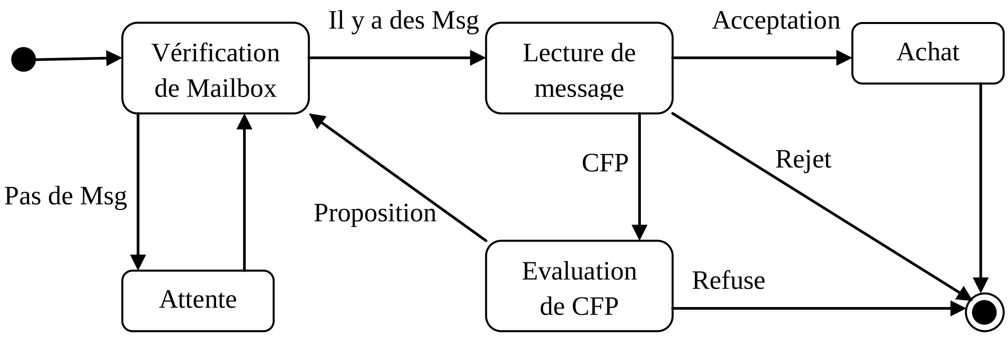


Figure 5.17 : Le comportement de l'agent *Acheteur*.

L'agent *Vendeur* (Figure 5.18), par contre, commence son comportement par l'envoi d'un appel d'offre puis il passe à l'état d'attente de réponses. A la réception de réponses, l'agent les évalue et il envoie ses réponses. Ensuite, il passe à l'état d'attente de la confirmation de l'agent gagnant. Après la réception de la confirmation, l'agent passe à son état final.

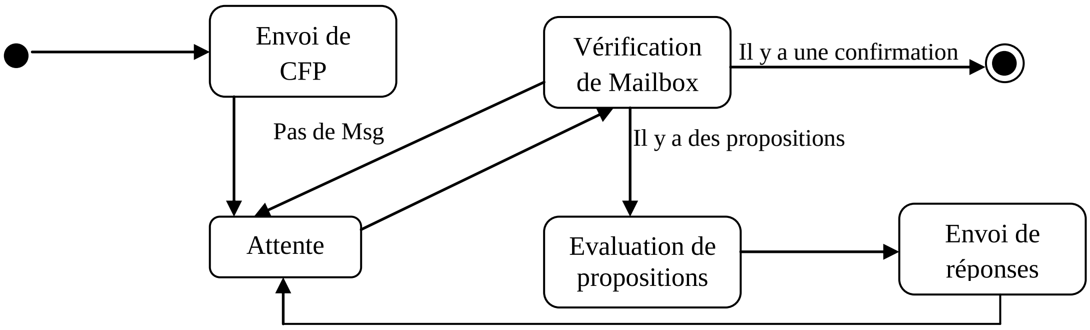


Figure 5.18 : Le comportement de l'agent *Vendeur*.

Le squelette du code de l'agent *Acheteur*, conformément au diagramme présenté dans la figure 5.17, est présenté dans la figure 5.19.

```
public class Buyer extends BasicCommunicatingAgent{
  public void step() {
       if(hasMail()){
         FIPAACLMessage m = (FIPAACLMessage)this.getFirstMessage();
            if(m.getPerformative()== "CallForProposal"){
                  // Evaluate the CFP, then send propose
            }
            if(m.getPerformative()== "RejectProposal"){
                Active = false ;
            }
            if(m.getPerformative()== "AcceptProposal"){
                  // Buying, then send InformDone Message
                  Active = false ;
            }
            readMailBox();
       }
       else{
            wwait();
            }
       }
  public boolean isActive(){
       return Active ;
  }
}
```

Figure 5.19: Le squelette du code de l'agent *Acheteur*.

Nous avons exécuté l'exemple avec un agent vendeur (*Seller*) et trois agents acheteur (*Buyer1*, *Buyer2* et *Buyer3*). Le tableau 5.1 représente la trace de l'exécution de cet exemple.

Tableau 5.1 : La trace d'exécution de l'exemple d'enchère.

Temps	Evènement
22	Création des agents: *Buyer1*, *Buyer2*, *Buyer3* et Seller
24	Les agents (*Buyer1*, *Buyer2* and *Buyer3*) exécutent respectivement ***step***() puis ***wwait***()
24	L'agent *Seller* exécute ***step***(), envoie ***CFP*** aux agents acheteurs, puis il exécute ***wwait***()
37	Les agents acheteurs (*Buyer1*, *Buyer2* and *Buyer3*) exécutent ***step***(). Les agents (*Buyer1* and *Buyer3*) reçoivent ***CFP*** de *Seller*, envoient leurs propositions, et exécutent ***step***() suivi par l'exécution de ***wwait***()
38	L'agent *Seller* exécute ***step***() et reçoit la proposition de *Buyer1*, puis il exécute ***step***() et reçoit la proposition de *Buyer3*.
38	L'agent *Buyer2* reçoit ***CFP***, envoie une proposition à l'agent *Seller*, exécute ***step***(), puis exécute ***wwait***()
39	L'agent *Seller* exécute ***step***(), reçoit la proposition de *Buyer2*, envoie une réponse aux acheteurs (***Reject_Proposal*** à *Buyer1* et *Buyer2*, et ***Accept_Proposal*** à *Buyer3*), exécute ***step***(), puis exécute ***wwait***()
51	L'agent *Buyer1* exécute ***step***(), reçoit le refus du vendeur et passe à l'état final
51	L'agent *Buyer3* exécute ***step***(), reçoit l'acceptation, envoie un message (***INFORM_Done***) au vendeur et passe à son état final
52	L'agent *Buyer2* exécute ***step***(), reçoit la refuse du vendeur et passe à l'état final
52	L'agent *Seller* exécute *step*(), reçoit la confirmation de l'agent *Buyer3* puis il passe à son état final

A base de cette trace, les différentes métriques sont calculées. Nous présentons deux exemples de mesures calculées. Le premier exemple (Figure 5.20) présente l'évolution de la mesure de la réactivité (*le nombre moyen de comportements suspendus*) de l'agent *Buyer2*. Rappelons que cette mesure est basée sur le nombre de suspension de comportements par rapport aux comportements exécutés, cette mesure commence par la valeur 1 dans l'instant 24 secondes parce que l'agent exécute un comportement (***step()***) suivi par ***wwait().*** Ensuite, l'agent Buyer2 exécute un comportement à l'instant 37 et la mesure de réactivité devient 0.5. A l'instant 38, la métrique (*le nombre moyen de comportements suspendus*) devient 0.33 suivi par la valeur 0.66 parce que l'agent a exécuté ***step()*** puis ***wwait()*** dans le même moment. Finalement, l'agent exécute un comportement à l'instant 52, et la valeur de cette métrique devient 0.5.

Le deuxième exemple présenté dans la figure 5.2 montre les deux formes de la mesure de l'accélération de la réalisation des buts de l'agent *Buyer3*. La première forme est calculée par rapport au temps. Ainsi, cet agent ne réalise son but qu'après 29 unités de temps (date de réalisation de but – date de création = 29). En conséquence, cette mesure passe de 0 vers 0.039 [buts/unité de temps]. Par contre, en calculant cette mesure en utilisant le nombre de comportements exécutés, nous pouvons remarquer que l'agent exécute 04 comportements avant d'atteindre son but. En effet, cette métrique passe de 0 vers la valeur 0.25 [but/comportement] à la fin de son exécution.

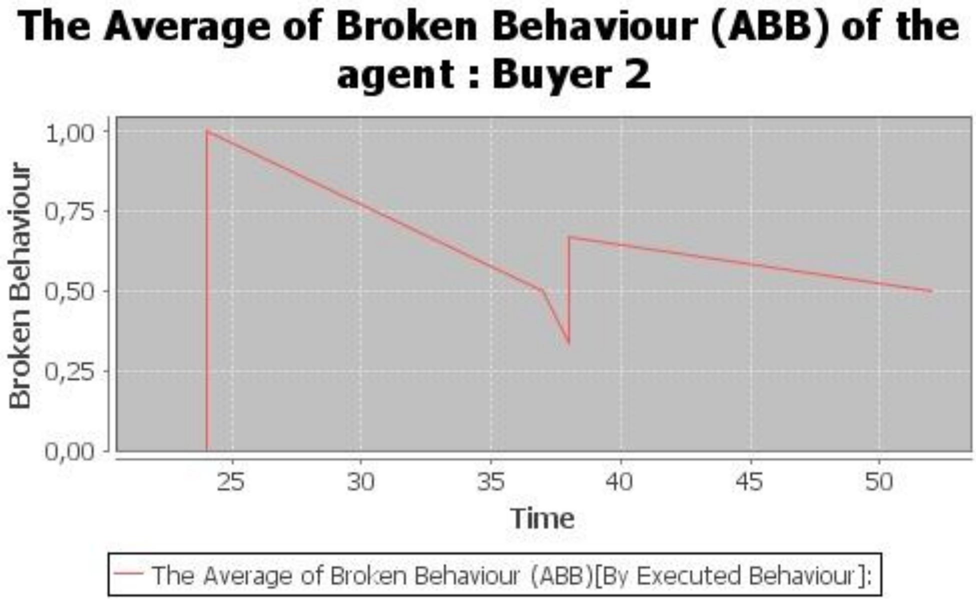


Figure 5.20 : La mesure de la réactivité de l'agent *Buyer2*.

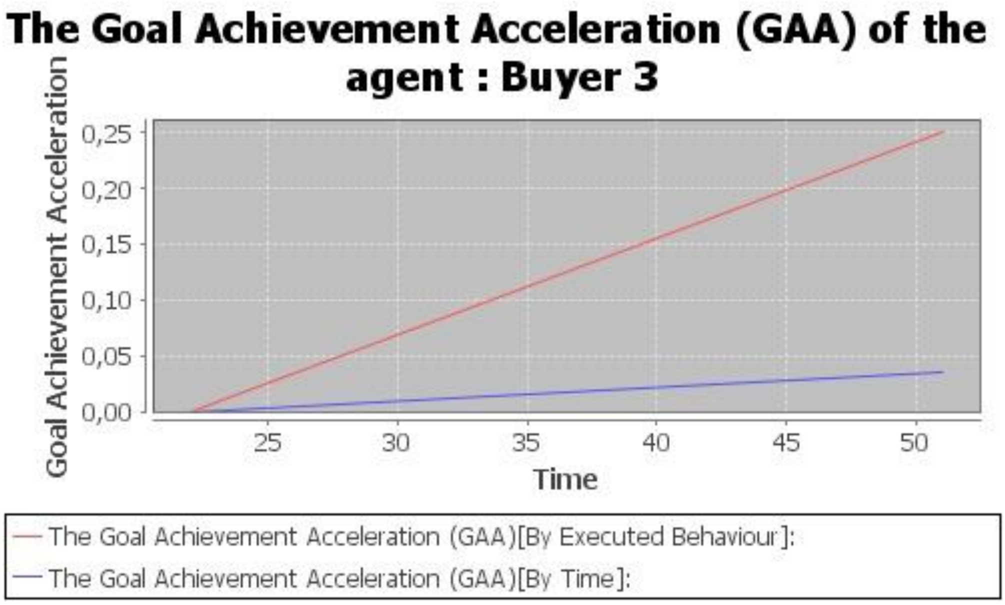


Figure 5.21 : Exemple de mesure de la rationalité de l'agent *Buyer3*.

5. Conclusion

Dans ce chapitre nous avons appliqué notre modèle de qualité spécifique aux SMA (QM4MAS) sur deux plateformes multi-agents. Avant d'appliquer le modèle, nous avons proposé une démarche d'application qui s'articule autour de deux étapes. La première étape consiste en la spécification de sous-caractéristiques jugées pertinentes à la plateforme parce que le modèle proposé englobe des sous-caractéristiques non fondamentales pour tous les agents. Ensuite, nous devons proposer des métriques concrètes sous la lumière de directives abstraites et qui doivent prendre en considération les spécificités de chaque plateforme. Favorisant l'application de métriques dynamiques, nous avons proposé l'utilisation du paradigme aspect pour cet objectif vu les avantages qu'il procure. En fait, nous avons appliqué cette démarche sur deux plateformes différentes : la plateforme JADE et la plateforme DIMA. Deux outils pour la mesure automatique de différentes mesures proposées ont été développés dans le cadre de notre travail. Les outils développés ont été présentés à travers deux études de cas.

Dans le chapitre suivant, nous présentons un autre aspect de notre travail qui consiste en l'étude détaillée d'un facteur qui affecte la qualité des SMA.

Chapitre 06

Evaluation de la Complexité des Systèmes Multi-Agents

« Mesure ce qui est mesurable, et rend mesurable ce qui ne peut être mesuré »

1. Introduction

Dans le chapitre 03, nous avons classifié les études ciblant la qualité des SMA selon deux axes : des études ciblant l'objectif de développement d'un modèle global pour la qualité des SMA et des études focalisant sur un attribut spécifique qui influe sur la qualité des SMA. Dans le deuxième axe, nous pouvons distinguer des approches consacrées à l'étude d'un attribut spécifique aux SMA (comme l'autonomie, la réactivité, l'intelligence, …etc.) et des approches qui traitent des attributs génériques (c'est-à-dire indépendants de paradigmes) selon les spécificités de paradigme agent (comme la performance et la complexité). Notre modèle de qualité (QM4MAS) est une contribution proposée dans le contexte du premier axe. Dans ce chapitre, nous présentons l'étude détaillée de la complexité des SMA en tant qu'attribut influant sur la qualité. Nous commençons ce chapitre par la modélisation du concept étudié. Ensuite, nous proposons un ensemble de mesures pour l'évaluation de la complexité des SMA. Finalement, nous présentons l'outil développé pour la mesure de ces attributs à travers des études de cas.

2. Modélisation de la complexité des SMA

Le paradigme agent représente maintenant l'un des paradigmes les plus appliqués. En fait, le spectre de son applicabilité allant du simple jeu d'enfants jusqu'aux systèmes extrêmement critiques comme les navettes spatiales. Particulièrement, ce paradigme semble adéquat pour le développement de systèmes complexes. A notre avis, plusieurs propriétés des SMA encouragent les développeurs à l'adopter durant le développement de systèmes complexes. La distribution d'exécution, la flexibilité de comportements et la richesse d'interactions sont des exemples de ces propriétés. Cependant, nous sommes persuadés que ces propriétés ont leur impact négatif sur la complexité des systèmes développés [Mar14b]. Par exemple, la flexibilité rend les comportements des agents imprévisibles. En conséquence, la compréhension de ces derniers, durant la phase de maintenance par exemple, devient une tâche complexe.

Malgré l'importance de la complexité, cet attribut est peu étudié dans le contexte des SMA par rapport aux attributs spécifiques aux agents ou par rapport à la performance. En plus, le nombre limité d'études abordant cet attribut est proposé pour des domaines restreints comme il est présenté dans le chapitre 03. Cette restriction limite l'applicabilité de différentes approches [Mar14b]. En conséquence, nous avons visé l'évaluation de cet attribut. Nous notons que l'évaluation de cet attribut permet, entre autres, la comparaison de produits logiciels partageant le même objectif ou comme un indicateur pour l'estimation de l'effort nécessaire durant la phase de maintenance.

Mettant l'étude de la complexité dans le contexte de notre travail qui consiste en la modélisation de la qualité des SMA, une question intrinsèque émerge : *si la complexité est à ce niveau d'importance, pourquoi le modèle QM4MAS ne la présente pas comme une sous-caractéristique ?* En fait, des propriétés comme la performance ou la complexité sont généralement omises des modèles de qualité parce qu'elles ont des relations avec plusieurs sous-caractéristiques. La complexité d'un logiciel, par exemple, est fortement attachée à la modularité de ce dernier et à ses fonctionnalités. Dans le cas du modèle QM4MAS, nous pouvons considérer la complexité comme un attribut dispersé sur plusieurs sous-caractéristiques. Comme dans le cas de tous les logiciels, la modularité qui est une sous-caractéristique dans notre modèle de qualité influe significativement sur la compréhension d'un logiciel et de sa complexité. En plus, la granularité représente le niveau de complexité d'un agent. Parce que nous visons la complexité du système et non pas seulement celle d'un agent *isolé*, le terme complexité devient plus approprié que la granularité. Ce point de vue est justifié par l'existence de sous-caractéristiques représentant des aspects collectifs des systèmes qui influent sur la complexité des SMA. L'interaction et l'organisation sont des exemples de ces aspects.

Avant la proposition de mesures pour la complexité, il est important de la spécifier de manière claire et précise afin d'éviter l'ambigüité associée généralement à ce genre de concepts. En plus, la spécification de ce concept doit prendre en considération les particularités des SMA.

Le glossaire standard des terminologies du génie logiciel [IEEE90] définit la complexité par *le degré de difficulté de compréhension et de vérification d'un système ou d'un composant.* Cette définition met en évidence la relation entre la complexité et l'effort consacré pour comprendre un logiciel. En conséquence, nous cherchons à identifier les propriétés qui peuvent influer sur la compréhensibilité d'un

SMA. Nous mettons ces propriétés ensemble afin de développer un modèle pour la compréhension et l'étude de la complexité des SMA [Mar14b].

Un système multi-agents est composé d'un ensemble d'agents en interaction entre eux. Evidement, le nombre d'agents et les interactions entre eux influent sur la complexité d'un tel système. Les entités formant le système (les agents) ne sont pas des entités *atomiques*. En fait, un agent est peut être composé de plusieurs composants internes. En conséquence, le nombre de composants internes d'un agent et les relations entre eux ont leur impact sur la complexité de l'agent. Pour conclure, nous pouvons remarquer que la complexité des SMA peut être étudié dans deux niveaux différents : le niveau agent et le niveau système [Mar14b]. Nous entendons par la complexité au niveau agent, la complexité de l'agent en tant qu'entité *isolée.* Par contre, la complexité au niveau système cible la complexité de tous les agents avec leurs interactions. Nous notons que la complexité de l'environnement fait partie du niveau système [Mar14b].

Au niveau agent, la complexité est étudiée selon deux critères orthogonaux : la complexité de la structure de l'agent et la complexité de son comportement. La complexité de la structure est la complexité des composants internes de l'agent dont les connaissances font partie. Par contre, la complexité de comportement est la complexité de tâches exécutées par l'agent afin de réaliser ses buts [Mar14b].

La complexité au niveau système est composé de la complexité de la structure sociale du système et des interactions entres les agents qui le forment. La structure sociale du système indique la structure globale de SMA. Elle englobe tous les agents composant le SMA et éventuellement les objets existants dans l'environnement. Les interactions entre les agents représentent le comportement collectif des SMA. Ce comportement collectif peut être assuré par l'interaction directe à travers l'échange de messages ou l'interaction indirecte à travers la manipulation des objets composant l'environnement [Mar14b].

La figure 6.1 présente le modèle proposé pour la complexité des SMA. Après le développement du modèle, nous devons proposer des mesures pour chaque critère proposé.

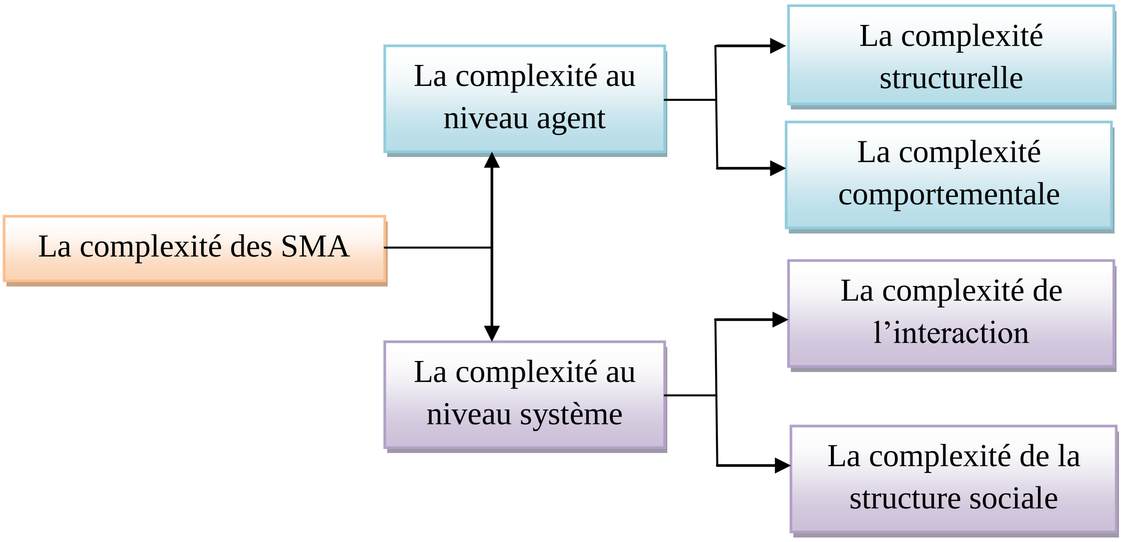


Figure 6.1 : Modèle de la complexité des SMA.

3. Mesure de la complexité des SMA

Le modèle proposé dans la section précédente est utilisé pour la spécification de la complexité des SMA en montrant les différents critères qu'ils peuvent l'affecter. Cependant, il est important de mesurer, le plus objectivement possible, la complexité de tels systèmes. Comme il est indiqué dans cette thèse, la diversité de paradigmes pouvant être utilisés pour l'implémentation des SMA complique l'objectif de la proposition des métriques génériques. En conséquence, nous avons décidé dans l'étude de la complexité de viser les SMA implémentés en utilisant le paradigme orienté-objet [Mar14b].

Afin de mesurer les différents critères, nous avons combiné les mesures statiques et les mesures dynamiques. Les mesures statiques sont calculées avant l'exécution du système en utilisant son code source. Par contre, le calcul des mesures dynamiques est basé sur le comportement du logiciel. Suivant la même approche pour la mesure de métriques dynamiques présentée précédemment, nous avons appliqué la programmation orientée-aspect dans cette étude [Mar14b].

Il est important de noter que la liste des mesures proposées dans le cadre de ce travail n'est pas exhaustive. En conséquence, il est possible d'étendre les mesures proposées par des nouvelles mesures. Les mesures proposées sont [Mar14b] :

- **Les mesures de la complexité structurelle de l'agent :** pour mesurer la complexité structurelle d'un agent nous proposons :
  1. **La taille de la structure de l'agent (SAS) :** la structure de l'agent est représentée par un ensemble d'attributs. Généralement, ces attributs sont utilisés pour la spécification de l'état interne de l'agent. En d'autres termes, l'état global de l'agent est distribué sur plusieurs attributs. Naturellement, l'augmentation du nombre d'attributs d'un agent complique la compréhension et l'analyse de l'état de l'agent. Sachant que le comportement de l'agent est basé, entre autres, sur son état, l'analyse du comportement de l'agent, aussi, devient difficile. En effet, nous utilisons la taille de la structure de l'agent en nombre d'attributs comme un indicateur de la complexité de la structure de l'agent [Mar14b].
  2. **La granularité de la structure de l'agent (ASG):** les attributs de la structure interne de l'agent n'ont pas tous le même niveau de complexité. Il est possible de trouver des attributs sous forme d'objets composé d'autres attributs. Evidement, l'existence de ces attributs composés influe sur la complexité de la structure de l'agent. Afin de mesurer la granularité de la structure de l'agent, nous modélisons les différents attributs sous forme d'un arbre dont la racine est l'agent. Les différents nœuds représentent des attributs et les feuilles représentent les attributs primitifs. Ainsi, $= \frac{\sum_{i=1}^{K} N_i}{K-1}$, sachant que K est le nombre de nœuds (K>1) et N_i est la profondeur de l'attribut i. Nous avons utilisé (K-1) parce que la racine de l'arbre n'est pas un attribut. Naturellement, un agent sans attribut a une granularité de structure égale à 0 [Mar14b].
  3. **La dynamicité de la structure de l'agent (DAS) :** les attributs de l'agent ne peuvent pas être seulement des attributs composés mais ils peuvent être de nature dynamique. Par un attribut dynamique, nous entendons par attribut un attribut auquel nous pouvons ajouter ou enlever des éléments. Bien entendu, connaitre la taille de ces structures nous donne précisément la

taille de la structure de l'agent. En plus, un changement fréquent de ces attributs est un signe de l'instabilité de la structure de l'agent. Ainsi, la dynamicité de la structure de l'agent est basée sur le changement de la structure entre deux moments. $DAS = \frac{CS_{t'} - CS_t}{t' - t}$, sachant que $CS_{t'}$ (respectivement CS_t) représente la taille de la structure dynamique au moment t' (respectivement t) [Mar14b].

- **Les mesures de la complexité comportementale de l'agent :**

1. **La taille comportementale de l'agent (BSA) :** un agent peut assurer plusieurs comportements différents. Bien entendu, un agent qui assure plusieurs comportements est plus compliqué que les agents spécialisés. En conséquence, nous calculons dans cette mesure le nombre de comportements implémentés au sein d'un agent [Mar14b].
2. **La complexité de comportements de l'agent :** la mesure précédente semble insuffisante. En fait, un agent composé de plusieurs comportements simples peut être plus maîtrisable qu'un agent qui assure un seul comportement complexe. La plateforme JADE propose le développement de comportements complexes selon une forme d'une machine à états finis. En conséquence, nous proposons l'utilisation du nombre *cyclomatique* proposé par McCabe [McC76]. Ce nombre est proposé pour la mesure de la complexité des logiciels en se basant sur la théorie des graphes. Nous avons adopté cette mesure pour les comportements de l'agent. Prenant un comportement composé représenté par un graphe (G), la complexité *cyclomatique* d'un comportement (CC_B) est calculée par la formule $CC_B = (E_G - N_G) + 2P_G$, tel que E_G est le nombre de liens du graphe, N_G est le nombre de nœuds du graphe et P_G est le nombre de composants connexes. La complexité *cyclomatique* d'un comportement simple égale 1. La complexité de comportements de l'agent peut être calculée par le moyen de complexités de ces comportements [Mar14b].

3. **Le nombre moyen de comportements planifiés (ANSB) :** un agent peut exécuter plusieurs comportements simultanément. Les comportements démarrés sont mis dans une liste de scheduling. L'exécution simultanée de comportements d'un agent assure la concurrence de fonctionnalités de l'agent. Cependant, cette concurrence nécessite l'introduction des mécanismes de synchronisation pour assurer un comportement cohérent de l'agent. Ces mécanismes compliquent la compréhensibilité de comportement de l'agent. En conséquence, nous pouvons considérer le nombre moyen de comportements planifiés comme étant une mesure de la complexité de comportements des agents [Mar14b].

- **Les mesures de la complexité de la structure sociale du système :** un SMA est composé d'un ensemble d'agents existants dans un environnement. Ce dernier est composé d'un ensemble d'objets. En conséquence, nous pouvons mesurer la complexité de la structure sociale du système par les mesures suivantes :

1. **L'hétérogénéité des agents (HA) :** cette mesure indique le nombre de classes d'agents. Nous croyons que l'hétérogénéité du système complique la compréhension de ce dernier parce qu'on doit comprendre plusieurs classes d'agents. En plus, la maintenance de plusieurs classes est plus compliquée que la maintenance d'une seule [Mar14b].
2. **L'hétérogénéité des objets de l'environnement (HEO) :** l'environnement d'un SMA est composé d'un ensemble d'objets. Comme dans la mesure précédente, nous pensons que l'hétérogénéité a son impact sur la complexité de la structure de SMA. En conséquence, nous proposons cette mesure qui représente le nombre de classes des objets composants l'environnement [Mar14b].
3. **La taille de la population d'agents (SAP) :** les agents s'exécutent en concurrence. Donc, le nombre des agents influe sur la complexité du SMA. En plus, l'augmentation du nombre d'agents implique une augmentation dans les interactions des

agents. Sans doute, ces interactions compliquent la compréhensibilité du comportement collectif des agents. Sachant que le nombre des agents n'est pas fixe parce que les agents peuvent être exécutés ou détruits durant l'exécution du SMA, nous proposons l'utilisation de mesure dynamique pour calculer le nombre des agents à chaque instant [Mar14b].

- **Les mesures de la complexité de l'interaction des agents :** l'interaction entre les agents peut être assurée directement par les messages ou indirectement par la manipulation des objets. En effet, nous proposons les mesures suivantes :

1. **Le taux de code consacré à l'interaction (RIC) :** cette mesure représente le nombre de lignes de code consacrées à la communication entre les agents par rapport au nombre total de lignes de code de leurs comportements. Les lignes de code consacrées à la communication englobent toutes les opérations possibles, comme le traitement, l'envoi et la réception de messages. Bien entendu, cette mesure ne présente pas le taux d'interaction parce que ce code peut être exécuté plusieurs fois à cause de boucles. Cependant, cette mesure nous permet d'estimer l'effort pour comprendre le comportement collectif des agents à partir de leurs codes sources. En plus, un niveau élevé de cette mesure indique un couplage fort entre les agents. Ce couplage nécessite le changement de tous les agents du système si on veut maintenir un agent [Mar14b].
2. **Le nombre moyen de messages échangés par agent (AEMA) :** comme nous l'avons indiqué avant, le taux de code consacré à l'interaction ne donne qu'une information partielle sur la complexité du comportement collectif des agents. Afin de comprendre le comportement global du système, nous devons prendre en considération les interactions entre les agents. En effet, cette mesure présente le nombre moyen de messages envoyés par agent [Mar14b].
3. **Le taux d'accessibilité de l'environnement (REA) :** les agents peuvent interagir indirectement par la manipulation des objets de

l'environnement. La manipulation des objets est conditionnée par l'existence des attributs publics. L'existence de ces attributs indique un couplage fort entre les agents. En conséquence, il sera difficile de changer le code d'un agent sans la modification des codes des autres agents. Cette mesure représente le nombre des attributs publics dans les objets de l'environnement par rapport au nombre total des attributs [Mar14b].

Cette liste de mesures n'est pas exhaustive. Il est possible d'étendre la liste des mesures de chaque critère pour supporter de nouveaux aspects. Comme les SMA sont des produits logiciels, nous pouvons appliquer des mesures de complexité connues dans le domaine du génie logiciel. Particulièrement, nous pouvons appliquer les mesures de génie logiciel orienté objet parce que nous avons visé les SMA basés sur le paradigme objet. En plus, il est possible de pondérer les différentes mesures proposées pour exprimer l'importance de chacune [Mar14b].

4. Outil développé et études de cas

Afin de mesurer la complexité des SMA basés sur la plateforme JADE, nous avons développé un outil permettant la mesure automatique de différentes mesures citées ci-dessus [Mar14b]. La figure 6.2 présente l'architecture de l'outil développé.

Le processus de mesure passe par deux étapes. Tout d'abord, l'outil analyse le code source de l'application afin de calculer les mesures statiques. Ensuite, le tisseur de l'*AspectJ* insère des aspects durant l'exécution de l'application afin de générer sa trace d'exécution. En se basant sur cette trace, nous pouvons calculer les mesures dynamiques. Enfin, les résultats des mesures peuvent être présentés sous forme textuel ou graphique comme ils peuvent être sauvegardés dans une base de données [Mar14b].

La trace d'exécution est générée grâce à la captation de différents évènements. La figure 6.3 montre un aspect utilisé pour capter la création d'un nouvel agent. Avant d'exécuter la méthode responsable de la création d'un agent, nous devons capter les informations pertinentes, comme : le temps de création, le nom d'agent,…etc, pour le calcul de différentes mesures.

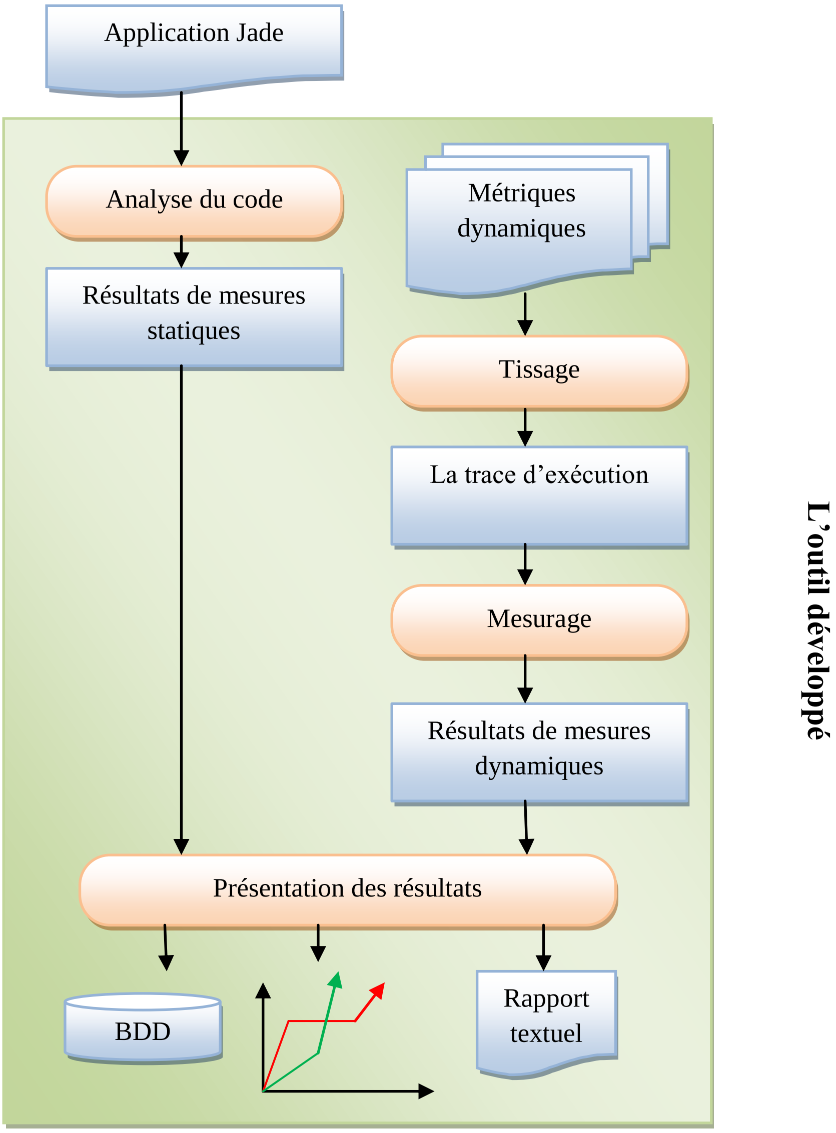


Figure 6.2 : L'architecture de l'outil développé.

```
public aspect CreationOfNewAgent {
    pointcut CreationOfAgent():execution(*
          *createNewAgent(..));
    before()  : CreationOfAgent(){
      // Sauvegarder des informations concernant l'agent
      }
}
```

Figure 6.3 : Un aspect pour capter la création d'un agent.

L'outil développé est validé sur deux exemples open source disponibles sur le site officiel de la plateforme JADE[1] [Mar14b]. Le premier exemple (appelé *FSMAgent*) représente un exemple de comportements composés. Comme il est présenté dans la figure 6.3, l'agent *FSMAgent* exécute un comportement composé décrit par une machine à états finis. En fait, deux comportements simples sont utilisés pour la formation du comportement complexe.

```
public class FSMAgent extends Agent {
  protected void setup() {
    FSMBehaviour fsm = new FSMBehaviour(this) {
      public int onEnd() {
           // la méthode onEnd
      }
    };
  // Le comportement compose en utilisant FSM
    addBehaviour(fsm);
  }
  private class NamePrinter extends OneShotBehaviour {
    // Le code du comportement NamePrinter
  }
  private class RandomGenerator extends NamePrinter {
     //Le code du comportement RandomGenerator
  }

}
```

Figure 6.4: Le squelette du code de l'agent *FSMAgent*.

Le deuxième exemple est une implémentation du protocole d'interaction *Contract Net* en JADE. Ce système est composé de deux classes d'agents : *Initiator* et *Participant*. L'initiateur commence son comportement par l'envoi d'un message *CFP* à tous les participants. Ensuite, il attend leurs réponses afin de sélectionner la meilleure proposition. A la réception de *CFP*, le participant formule et envoie sa proposition s'il veut participer, sinon il envoie un message *Refuse*. Selon la réponse de l'initiateur, le participant termine son exécution si la réponse est un message *Reject*, sinon il exécute l'action demandée afin de fournir les résultats à l'initiateur.

Le tableau 6.1 représente les mesures statiques de deux exemples. Nous notons que les deux mesures : *l'hétérogénéité des objets de l'environnement (HEO)* et *le taux d'accessibilité de l'environnement (REA)* sont omises parce que leurs valeurs sont

[1] http://jade.tilab.com/

nulles (il n'existe pas des objets composant l'environnement dans les deux exemples). L'agent *FSMAgent* est composé de trois comportements ce qui représente dans la mesure la taille comportementale de l'agent (BSA). Parce que l'un de ces comportements est un comportement complexe (modélisé par une machine à états finis avec huit liens et six nœuds), la moyenne de la complexité de comportements devient 02 [Mar14b].

Par contre, l'exemple *Contract Net* est composé de deux classes d'agents (donc, l'hétérogénéité des agents (HA) est 02), dont chacun exécute un seul comportement. En conséquence la taille comportementale de l'agent est 01. Cette valeur est égale à la mesure de la moyenne de la complexité de comportements parce que les deux comportements implémentés dans les agents de cet exemple sont des comportements simples. Nous remarquons que la mesure du taux du code consacré à l'interaction (RIC) est égale à 0.46, c'est-à-dire presque la moitié du code source de cet exemple est consacrée à l'interaction entre les agents [Mar14b].

Tableau 6.1 : Les résultats de mesures statiques.

Les mesures	FSMAgent	ContractNet
La taille de la structure de l'agent (SAS)	**06**	**0.5**
La granularité de la structure de l'agent (ASG)	**01**	**0.5**
La taille comportementale de l'agent (BSA)	**03**	**01**
Le moyen de la complexité de comportements	**02**	**01**
L'hétérogénéité des agents (HA)	**01**	**02**
Le taux de code consacré à l'interaction (RIC)	**00**	**0.46**

Après l'exécution des deux exemples, nous obtenons les mesures dynamiques. Nous notons que l'exemple *FSMAgent* est exécuté avec un seul agent et l'exemple de protocole *Contract Net* est exécuté avec un initiateur et trois participants. Les résultats de trois mesures (*Le nombre moyen de messages échangés par agent (AEMA), La taille de la population d'agents (SAP) et Le nombre moyen de comportements planifiés (ANSB))* sont présentés dans le tableau 6.2. Les autres mesures sont omises parce que leurs valeurs sont restées nulles durant l'exécution des deux exemples [Mar14b].

Prenant la mesure (*nombre moyen de messages échangés par agent (AEMA)*) de l'exemple *Contract Net*, nous pouvons remarquer que sa valeur au démarrage du système est nulle. Cette valeur reste inchangeable jusqu'à l'envoi du premier message

(CFP) par l'initiateur à l'instant 4672 ms, où la valeur de cette mesure devient 0.25 messages par agent. Bien entendu, cette mesure continue son évolution au rythme d'échange de messages jusqu'à l'obtention de la valeur 2.5 à la fin de l'exécution. De la même manière, les autres mesures évoluent selon l'exécution du système. La trace de l'exécution est présentée dans le même tableau afin de simplifier la lisibilité [Mar14b].

Tableau 6.2 : Les résultats de mesures dynamiques de deux exemples.

	Temps	Evènement	AEMA	SAP	ANSB
ContractNet	00	Démarrage de l'exécution	**00**	**00**	**00**
	4297	Création des agents: *Participant_01, Participant_02* et *Participant_03*	**00**	**03**	**00**
	4375	Création de l'agent *Initiator*	**00**	**04**	**00**
	4422	Planification de comportements des agents: *Participant_01* et *Participant_02*	**00**	**04**	**0.5**
	4485	Planification de comportement de l'agent *Participant_03*	**00**	**04**	**0.75**
	4672	L'agent *Initiator* envoie un message *CFP* aux participants: *Participant_01, Participant_02* et *Participant_03*	**0.75**	**04**	**0.75**
	4719	Planification de comportement de l'*Initiator*	**0.75**	**04**	**1**
	4766	Les participants (*Participant_01, Participant_02* et *Participant_03)* envoient leurs propositions	**1.5**	**04**	**1**
	4875	L' *Initiator* envoie des réponses aux participants (*Participant_01, Participant_02* et *Participant_03)*	**2.25**	**04**	**1**
	4938	Le *Participant_02* envoie le message *Inform* à l'agent *Initiator*	**2.5**	**04**	**1**
FSMAgent	00	Démarrage de l'exécution	**00**	**00**	**00**
	3407	Création de l'agent	**00**	**01**	**00**
	3422	Planification du comportement de l'agent	**00**	**01**	**01**

En analysant les résultats obtenus, nous remarquons que la source de la complexité dans l'exemple *FSMAgent* est due à son comportement individuel. Par contre, c'est l'aspect social qui est la source de la complexité de l'exemple *Contract Net.* Cet exemple consacre plus de la moitié de son code pour les interactions entre les agents. En plus, nous constatons que les deux types de mesures (statiques et dynamiques) ne donnent pas des informations contradictoires. Cependant, l'obtention des résultats de mesures statiques incompatibles avec des résultats de mesures dynamiques est fortement possible. Par exemple, il est possible de trouver un niveau de taux de code consacré à l'interaction (RIC) minimal avec un niveau de nombre moyen de messages échangés par agent (AEMA) élevé. Nous croyons que l'analyse approfondie de toutes les mesures proposées nous conduit à l'identification de la source de la complexité du système étudié [Mar14b].

5. Conclusion

Dans ce chapitre nous avons suivi une autre piste de l'étude de la qualité des SMA différente de celle suivie dans les deux chapitres précédents. Il s'agit de l'étude d'un seul attribut de la qualité des SMA au lieu du développement d'un modèle global de la qualité. Nous avons choisi l'étude de la complexité à cause de son impact sur la qualité du logiciel. En plus, cet attribut est peu étudié par rapport à la performance des SMA ou les attributs spécifiques de tel paradigme logiciel (comme l'autonomie, la réactivité et l'aspect social). Dans le cadre de cette thèse, nous avons proposé un modèle pour la complexité des SMA qui décompose ce concept en un ensemble de propriétés. Ensuite, nous avons proposé des mesures statiques et dynamiques pour la mesure de chaque propriété. Un outil a été développé pour la mesure automatique de mesures proposées. Bien entendu, ce travail peut être étendu dans plusieurs directions dont la plus intéressante à notre avis est la mesure de la complexité durant les phases de spécification ou de conception.

Conclusion et Perspectives

1. Bilan

Notre travail s'inscrit dans le domaine de génie logiciel-orienté agent. Particulièrement, il vise la problématique de l'assurance de la qualité des systèmes multi-agents. Nous nous sommes intéressés essentiellement au développement de modèles de qualité spécifiques aux SMA parce qu'ils représentent la base de la spécification et l'évaluation de la qualité.

Afin de proposer des contributions bénéfiques, nous avons entamé notre recherche par un état de l'art couvrant les domaines attachés à notre problématique. Ainsi, le premier chapitre n'a pas été spécifique à la qualité des SMA mais il donne un aperçu couvrant le domaine de la qualité des logiciels en général. L'étude de ce domaine nous a montré l'évolution des approches proposées depuis la proposition des modèles de qualité à la fin des années soixante-dix, la proposition des modèles spécifiques aux différents catégories de produits logiciels, la standardisation de modèles de qualité et l'extension de modèles standards afin de supporter les particularités de différentes catégories.

Dans le deuxième chapitre, nous avons présenté le deuxième domaine attaché à notre recherche, à savoir les systèmes multi-agents. En fait, ce domaine a été présenté sous un angle spécifique : l'ingénierie de tels systèmes. Cependant, nous avons entamé ce chapitre par la présentation de concepts de base des systèmes multi-agents. A cause de l'expansion de méthodes appliquées dans chaque phase de cycle de développement d'un SMA, nous nous sommes focalisés dans ce chapitre sur les modèles d'agents, les méthodologies de développement et l'implémentation de tels systèmes.

L'étude de travaux consacrés à la qualité des SMA a été présentée dans le chapitre 03. Nous avons classifié ces études en deux classes : les modèles de la qualité des SMA et les métriques de différents attributs de ces systèmes. Ce chapitre a été couronné par une étude comparative qui nous a montré les limites de travaux présentés. Ces limites consistent essentiellement en l'absence d'un modèle de qualité global qui englobe les caractéristiques de la qualité de logiciels et les propriétés des SMA et l'ignorance de modèles standards lors du développement des modèles spécifiques aux SMA. En plus, les méthodes de mesure sont généralement omises dans les différents travaux présentés. Nous avons basé nos contributions sur ces remarques afin de combler ces lacunes.

Dans le chapitre 04, nous avons présenté nos deux premières contributions qui se placent dans le contexte la modélisation de la qualité des SMA. Dans la première partie, un modèle appelé QM4MAS qui consiste à une extension du modèle standard ISO/IEC 9126 a été présenté. Dans ce modèle nous avons intégré les notions de base des SMA dans le niveau de sous-caractéristiques du modèle standard. Par ailleurs, nous avons proposé l'application de *directives abstraites* dans le niveau des mesures afin de caractériser notre modèle par la généricité nécessaire pour étendre sa portée d'application. Dans la deuxième partie de ce chapitre, nous avons proposé une extension du modèle standard ISO/IEC25010 pour supporter les spécificités des SMA. Ce pas vers l'utilisation du nouveau modèle standard est basé sur la notion de l'agent faible.

Dans le chapitre 05, nous avons présenté l'application du modèle QM4MAS sur deux plateformes multi-agents. Ainsi, une démarche d'application a été proposée. La démarche consiste en la proposition de mesures concrètes à partir de directives abstraites afin de mesurer les différentes sous-caractéristiques jugées pertinentes pour chaque plateforme. Par rapport aux travaux proposant des mesures pour les SMA, notre proposition est conforme au standard ISO/IEC 9126 parce qu'on spécifie en plus de mesures, la méthode de mesure et d'échelle. En fait, nous avons proposé l'utilisation du paradigme aspect afin de mesurer les métriques dynamiques. En plus de la popularité des deux plateformes choisies, ces dernières sont choisies par rapport au critère de l'indépendance envers les modèles d'agents. En effet, la plateforme JADE est indépendante de modèles d'agents, tandis que la plateforme DIMA est basée sur un modèle spécifique.

Dans un autre axe de domaine de la qualité des SMA, nous avons proposé l'étude et la mesure d'un attribut spécifique. Nous avons choisi la complexité comme un attribut cible à cause de son impact sur la qualité des logiciels. En plus, cet attribut n'a pas encore été étudié de manière profonde dans le contexte des SMA. Ainsi, nous avons proposé un modèle pour la spécification de la complexité des logiciels basés sur le paradigme agent. Ensuite, un ensemble de mesures ont été proposées pour l'évaluation de cet attribut.

2. Perspectives

Bien entendu, nos contributions restent ouvertes pour des extensions sur plusieurs directions. Tout d'abord, les modèles de qualité sont classifiés selon leurs objectifs en trois catégories : la spécification, l'évaluation et/ou la prédiction de la qualité. Nos contributions se focalisent principalement sur le premier objectif en touchant superficiellement l'évaluation de la qualité des SMA. En effet, une piste prometteuse consiste à développer des modèles d'évaluation et de prédiction de la qualité des SMA. Naturellement, l'évaluation ou la prédiction de la qualité sera basée sur la spécification présentée dans notre modèle proposé. Cependant, les modèles de l'évaluation ou de la prédiction augmentent l'utilité de l'étude de la qualité des SMA. En fait, un modèle d'évaluation va présenter un framework pour la mesure de la qualité globale et l'interprétation de résultats obtenues. Un modèle de prédiction permet l'estimation de la qualité des SMA avant la phase d'implémentation ou de déploiement.

Une autre entrave remarquée durant le développement de nos contributions est la diversité de modèles, de paradigmes et de plateforme d'implémentation des SMA. La généricité des modèles proposés restent limitée dans ce contexte. Dans notre modèle QM4MAS, nous avons proposé l'utilisation de directives abstraites afin de remédier à ce problème. Cependant, nous croyons que le recours à la méta-modélisation de la qualité des SMA représente la solution idéale. En conséquence, nous proposons de développer un méta-modèle de la qualité des SMA qui permet l'instanciation de modèles spécifiques selon l'architecture ou la plateforme multi-agents ciblé.

Dans l'axe de l'étude des attributs des SMA et leurs impacts sur la qualité, nous pensons que le paradigme agent est riche de concepts qui influent sur la qualité. Cette hypothèse doit être mise en étude à travers l'évaluation de l'impact de ces concepts sur la qualité des SMA. La rationalité, l'organisation et la flexibilité représentent des exemples candidats de tels concepts.

Bibliographie

[Alo98] F. Alonso, J. L. Fuertes, C. Montes, R. J. Navajo, *A Quality Model: How to Improve the Object Oriented Software Process*, IEEE International Conference on Systems, Man, and Cybernetics, 1998.

[Alo08] F. Alonso, J. L. Fuertes, L. Martínez, H. Soza, *Measuring the Social Ability of Software Agents*, Sixth International Conference on Software Engineering Research, Management and Applications (SERA'08), 2008.

[Alo09] F. Alonso, J. L. Fuertes, L. Martínez, H. Soza, *Towards a set of Measures for Evaluating Software Agent Autonomy*, Eighth Mexican International Conference on Artificial Intelligence, 2009.

[Alo10a] F. Alonso, J. L. Fuertes, L. Martínez, H. Soza, *Measuring the Pro-Activity of Software Agents*, Fifth International Conference on Software Engineering Advances, IEEE Computer Society, 2010.

[Alo10b] F. Alonso, J. L. Fuertes, L. Martínez, H. Soza, *Evaluating Software Agent Quality: Measuring Social Ability and Autonomy*, T. Sobh, K. Elleithy (Eds.), Innovations in Computing Sciences and Software Engineering, Springer Science+Business Media, 2010.

[Arl04] F. Arlabosse, M.-P. Gleizes, M. Occello : *Méthodes de Conception*. Dans Systèmes Multi-Agents, volume 29 de *Arago*, pages 137-171. Editions Tech & Doc, 2004.

[Aus62] J. Austin. *How to Do Things with Words ?* Clarendon Press, London, 1962.

[Aza07] S. Azaiez, *Approche dirigée par les modèles pour le développement de systèmes multi-agents*, Thèse de doctorat de l'université de Savoie, 2007.

[Azu04] M. Azuma, *Applying ISO/IEC 9126-1 Quality Model to Quality Requirements Engineering on Critical Software*,Proceeding of the Third International Workshop on Requirements for High Assurance Systems, 2004.

[Ban02] J. Bansiya et C. G.Davis, *A Hierarchical Model for Object-Oriented Design Quality Assessment*, IEEE Transactions on Software Engineering, VOL. 28, No. 1, 2002.

[Bar07] J. B. Barranco, T. Stratulat, J. Ferber, *A Unified Model for Physical and Social Environments*, D. Weyns, H.V.D. Parunak, F. Michel (Eds.): E4MAS 2006, LNAI 4389, pp. 41–50, 2007.

[Bar99] K. S. Barber, C. E. Martin, *Agent Autonomy: Specification, Measurement, and Dynamic Adjustment*, In Proceedings of the Autonomy Control Software Workshop at Autonomous Agents 1999 (Agents'99), 1999.

[Bau12] L. Bautista, A. Abran, A. April, *Design of a Performance Measurement Framework for Cloud Computing*, In Journal of Software Engineering & Applications . Feb2012, Vol. 5 Issue 2, 2012.

[Beh09] B. Behkamal, M. Kahani, M. K. Akbar, *Customizing ISO 9126 Quality Model for Evaluation of B2B Applications*, In *Information and Software Technology*, Elsevier, 2009.

[Bel07] F. Bellifemine, G. Caire, D. Greenwood, *Developing Multi-Agent Systems with JADE,* John Wiley & Sons, 2007.

[Ber04] F. Bergenti, M. P. Gleizes, F. Zambonelli (Eds), *Methodologies and Software Engineering for Agent Systems - The Agent-Oriented Software Engineering Handbook,* Kluwer Academic Publishers, 2004.

[Ber08] M. Berryman, *Review of software platforms for agent based models*, Rapport technique, DTIC Document, 2008.

[Ber09] C. Bernon, M. P. Gleizes, G. Picard, *Méthodes orientées agent et multi-agent*, Dans Technologies des systèmes multi-agents et applications industrielles, Amal El FallahSeghrouchni, Jean-Pierre Briot (Eds.), Lavoisier, p. 44-76, IC2, avril 2009.

[Bey09] G. Beydoun, G. Low, B. Henderson-Sellers, H. Mouratidis, J. J. Gomez-Sanz, J. Pavon, C. Gonzalez-Perez, *FAML: A Generic Metamodel for MAS Development*, IEEE Transaction On Software Engineering, Volume 35, Issue 06, 2009.

[Bit12] P. D. Bitonto, M. Laterza, T. Roselli, V. Rossano, *Evaluation of Multi-Agent Systems: Proposal and Validation of a Metric Plan*, Dans N.T. Nguyen (Ed.): Transactions on CCI VII, LNCS 7270, Springer-Verlag, 2012.

[Boe78] B.W. Boehm, J.R. Brown, H. Kaspar, M. Lipow, G. J. Macleod, M. J. Merrit, *Characteristics of Software Quality*, North-Holland, Amsterdam, 1978.

[Boi13] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, A. Santi, *Multi-agent oriented programming with JaCaMo*. Sci. Comput. Program. 78(6): 747-761, 2013.l_

[Boo92] G. Booch, F. Reese et L. Reese, *Conception Orientée Objets et Applications*, Addison-Wesley France Eds, 1992.

[Bor05] R. H. Bordini, M. Dastani, J. Dix, A. F. Seghrouchni (Eds), *Multi-Agent Programming Languages: Tools and Applications*, Springer, 2005.

[Bor07a] R. H. Bordini, M. Dastani, J. Dix, A. F. Seghrouchni (Eds), Programming Multi-Agent Systems, 4th InternationalWorkshop, ProMAS 2006 Hakodate, Japan, May 9, 2006, LNCS 4411, Springer, 2007.

[Bor07b] R. H. Bordini, J. F. Hübner, M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak Using Jason*, John Wiley & Sons, 2007.

[Bor09] R. H. Bordini, M. Dastani, J. Dix, A. F. Seghrouchni (Eds), *Multi-Agent Programming Languages: Tools and Applications*, Springer, 2009.

[Bou92] T. Bouron, *Structures de Communication et d'Organisation pour la Coopération dans un Univers Multi-Agents - Une contribution à la définition d'un modèle de programmation agent basé sur les concepts d'engagement et d'acte de langage,* Thèse de doctorat de l'université Paris 6, Novembre 1992.

[Bou06] S. Bouktif, *Amélioration de la prédiction de la qualité du logiciel par combinaison et adaptation de modèles*, Thèse de doctorat de l'université de Montréal, 2006.

[Bra87] C. Braganza et L. Gasser, *MACE Multi-Agent Computing Environment, version 3*, Release Note 1.0, Technical Rapport CRI 87-16, University of Southern, California, USA, 1987.

[Bra88] M. E. Bratman, D. J. Israel et M. Pollack, *Plans and resource bounded practical reasoning*, Computational Intelligence, Vol. 4, pp. 349-355, 1988.

[Bre04] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos, *Tropos: An Agent-Oriented Software Development Methodology*, Autonomous Agents and Multi-Agent Sytems, 8, 203–236, 2004.

[Cab03] A. F. Caballero, V. L. Jaquero, F. M. Simarro, P. González, *Adaptive Interaction Multi-agent Systems in E-learning/E-teaching on the Web*, J. M. C. Lovelle, B. M. G. Rodríguez, L. J. Aguilar, J. E. L. Gayo, M. d. P. P. Ruíz (Eds.), Web Engineering, International Conference, ICWE 2003, Springer, Oviedo, Spain, 2003.

[Cab13] J. I. Cabrera, J. H. Orallo, *Interaction settings for measuring (social) intelligence in multi-agent systems,* Computing Research Repository (CoRR), 2013.

[Cai01] G. Caire, W. Coulier, F. Garijo, J. G. Sanz, J. Pavon,F. Leal, P. Chainho, P. E. Kearney, J. Stark, R. Evans, P. Massonet, *Agent Oriented Analysis Using MESSAGE/UML*. LNCS 2222, Springer, 2001.

[Cam11] A. Campbell et A., S. Wu, *Multi-agent role allocation: issues, approaches, and multiple perspectives*, Autonomous Agent and Multi-Agent Systems (2011) 22, 2011.

[Car12] A. Carlin et S. Zilberstein, *Bounded Rationality in Multi agent Systems Using Decentralized Meta-reasoning*, In T. Guy, M. Karny, and D. Wolpert (Eds.), Decision Making with Imperfect Decision Makers, Springer, 2012.

[Cia01] P. Ciancarini et M. Wooldridge (Eds), *Agent-Oriented Software Engineering*, Proceedings of the First International Workshop (AOSE-2000), Lecture Notes in Computer Science, Springer, 2014.

[Coa12] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, C. Greenough, *Exploitation of high performance computing in the flame agent-based simulation framework*, In Proceedings of the 2012 IEEE 14th International

Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems, HPCC '12, HPCC '12, IEEE Computer Society, 2012.

[Col95] D. Coleman, B. Lowther, P. Oman, *The application of software maintainability models in industrial software systems*, J. Syst. Softw. 29(1), 1995.

[Cos05] M. Cossentino, *From Requirements to Code with the PASSI Methodology*, Agent-Oriented Methodologies, B. H. Sellers and P. Giorgini (Eds), Idea Group Inc., 2005.

[Cos14] M. Cossentino, V. Hilaire, A. Molesini, V. Seidita (Eds), *Handbook on Agent-Oriented Design Processes*, Springer, 2014.

[Cro79] P. B. Crosby, *Quality Is Free: the art of making quality certain*, McGraw-Hill, 1979.

[Das10] M. Dastani, K. V. Hindriks, J. J. C. Meyer, *Specification and Verification of Multi-agent Systems*, Springer, 2010.

[Dei07] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, J.-F. Girard, *An Activity-Based Quality Model for Maintainability*, IEEE International Conference on Software Maintenance (ICSM 2007), 2007.

[Dek02] M. Dekhtyar, A. Dikovsky, M. Valiev, *Complexity of Multi-agent Systems Behavior*, 8th European Conference Logics in Artificial Intelligence, JELIA 2002, Springer Berlin Heidelberg, Cosenza, Italy, 2002.

[Del99] S. A. DeLoach, *Multiagent Systems Engineering: A Methodology And Language for Designing Agent Systems*, Presented at Agent-Oriented Information Systems (AOIS) '99, 1999.

[Dem01] Y. Demazeau. *VOYELLES*. Mémoire d'Habilitation à Diriger des Recherches, Institut National Polytechnique de Grenoble INPG, Avril 2001.

[Dev07] B. Devèze, C. Chopinaud, P. Taillibert, *ALBA: A Generic Library for Programming Mobile Agents with Prolog*, R. H. Bordini, M. Dastani, J. Dix, A. F. Seghrouchni (Eds), Programming Multi-Agent Systems, 4th InternationalWorkshop, ProMAS 2006 Hakodate, Japan, May 9, 2006, LNCS 4411, Springer, 2007.

[Din10] R. Dinu, T. Stratulat, J. Ferber, "*A formal approach to MASQ*", AAMAS'10: The Ninth Int. Conf. on Autonomous Agents and MultiAgent Systems, Toronto. 2010.

[Dos03a] J. Dospisil, *Code Complexity Metrics for Mobile Agents Implemented with Aspect/J™*, Multi-Agent Systems and Applications III, LNCS 2691, 2003.

[Dos03b] J. Dospisil, *Software metrics, information and entropy*, S. J. Lloyd and J. Peckham (Eds), Practicing Software Engineering in the 21st Century, IGI publishing, 2003.

[Dro92] R.G. Dromey et A.D. McGettrick, *On specifying software quality*, Software Quality Journal 1, 45-74 , 1992.

[Dro95] R. G. Dromey, *A Model for Software Product Quality*, in IEEE Transactions on Software Engineering (TSE), Volume 21, Number 2, 1995.

[Dum10] R. Dumke, S. Mencke, C. Wille, *Quality Assurance of Agent-Based and Self-Managed Systems*, CRC Press, 2010.

[Dzi07] M. Dziubiński, R. Verbrugge, B. D. Kęplicz, *Complexity Issues in Multiagent Logics*, Fundamenta Informaticae 75, 2007.

[Fac06] N. Faci, Z. Guessoum, O. Marin, *DimaX: a fault-tolerant multi-agent platform*, SELMAS '06 Proceedings of the 2006 International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, ACM New York, 2006.

[Fer95] J. Ferber. *Les systèmes multi-agents: vers une intelligence collective*. InterEditions, Paris, 1995.

[Fer98] J. Ferber et O. Gutknecht, *A meta-model for the analysis and design of organizations in multi-agent systems*, In the 3rd International Conference on Multi Agent Systems *(ICMAS '98)*, Washington, DC, USA, IEEE Computer Society, 1998.

[Fer99] J. Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[Fer04] J. Ferber, O. Gutknecht, F. Michel, *From Agents to Organizations: an Organizational View of Multi-Agent Systems*, In P. Giorgini, J. Müller, J. J. Odell (Eds), Agent-Oriented Software Engineering (AOSE) IV, ,Melbourne, July 2003, LNCS 2935, 2004.

[Fer05] J. Ferber, F. Michel, and J. Baez, *AGRE: Integrating Environments with Organizations*, D. Weyns et al. (Eds.): E4MAS 2004, LNAI 3374, pp. 48–56, 2005.

[FIPA01] FIPA, *FIPA ACL Message Structure Specification*. FIPA, 2001.

[Fra96] S. Franklin et A. Graesser, *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*, Proceeding ECAI '96 Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages, 1996.

[Gal09] S. Galland, N. Gaud, J. Demange, A. Koukam, *Environment Model for Multiagent-Based Simulation of 3D Urban Systems*, 7th European Workshop on Multi-Agent Systems (EUMAS 2009), Ayia Napa, Cyprus, 17th–18th December 2009.

[Gar84] D. A. Garvin, *What does "product quality" really mean?*, MIT Sloan Manag. Rev. **26**(1), 1984.

[Gau08] N. Gaud, S. Galland, V. Hilaire, A. Koukam, *An Organisational Platform for Holonic and Multiagent Systems*, In E. Hindriks, A. Pokahr , S. Sardina (Eds.), Sixth International Workshop on Programming Multi-Agent Systems (PROMAS'08), Seventh International Conference on Autonomous agents and Multiagent Systems (AAMAS'08), 2008.

[Gio05] P. Giorgini et B. H. Sellers, *Agent-Oriented Methodologies : An Introduction*, dans Agent-Oriented Methodologies, (Eds), , IGI-Global, 2005.

[Gla96] N. Glaser, *Contribution to Knowledge Modelling in a Multi-Agent Framework (the CoMOMS Approach)*, Thèse de Doctorat de l'Universtité Henri Poincaré, Nancy I, France, November 1996.

[Gue01] Z. Guessoum, M. Occello, *Environnements de développement*, Dans J. P. Briot, Y. Demazeau (Eds) Principes et architectures des systèmes multi-agents, Éditions Hermès, 2001.

[Gue03] Z. Guessoum, *Modèles et Architectures d'Agents et de Systèmes Multi-Agents Adaptatifs*, Dossier d'Habilitation à Diriger les Recherches, Université Pierre et Marie Curie, 2003.

[Gue12] Z. Guessoum, R. Mandiau, P. Mathieu, O. Boissier, P. Glize, A. Hamri, S. Pesty, G. Picard, J. P. Sansonnet, C. Tessier, E. Tranvouez, *Systèmes Multi-Agents et Simulation*, Chapter in book "Information - Interaction - Intelligence, Le point sur le i(3)", F. Sèdes, P. Marquis, J. M Ogier (Eds), Cépaduès Editions, Vol. ISBN 978.2.36493.009.4, pp 76-120, 2012

[Gup08] V. Gupta, J. K. Chhabra, *Measurement of Dynamic Metrics Using Dynamic Analysis of Programs*, APPLIED COMPUTING CONFERENCE (ACC '08), Istanbul, Turkey, May 27-30, 2008.

[Gut09] C. Gutiérrez et I. G. Magariño, *A Metrics Suite for the Communication of Multi-agent Systems, Journal of Physical Age*nts, VOL. 3, NO. 2, 2009.

[Has03] S. Hassas, *Systèmes Complexes à base de Multi-Agents Situés*, Mémoire pour l'obtention de l'HDR de l'université Claude Bernard, Lyon 1, France, 2003.

[Her10] M. Herrera, M. A. Moraga, I. Caballero, C. Calero, *Quality in Use Model for Web Portals (QiUWeP)*, In 10th International Conference on Web Engineering ICWE 2010 Workshops, 2010.

[Hil08] V. Hilaire, *Du semi-formel au formel : une Approche Organisationnelle pour l'Ingénierie de Systèmes Multi-Agents*, Habilitation à Diriger les Recherches de l'Université de Franche-Comté, 2008.

[Hor04] B. Horling et V. Lesser, *A survey of multi-agent organizational paradigms*, The Knowledge Engineering Review, Volume 19 Issue 4, December 2004, 2004.

[Hus09] S, A. Husein. *Coupling and Cohesion Metrics Suite for Object-Oriented Software*, In International Conference on Computer Technology and Development (ICCTD '09), 2009.

[Hüb10] J. F. Hübner, O. Boissier, R. Kitio, A. Ricci, *Instrumenting multi-agent organisations with organisational artifacts and agents*, Dans Autonomous Agents and Multi-Agent Systems, Volume 20, Issue 3, Springer, 2010.

[IEEE90] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE, 1990.

[IEEE98] *IEEE Standard for a Software Quality Metrics Methodology*, IEEE Std 1061-1998, 1998

[Igl98] C. A. Iglesias, M. Garijo, J. C. González, J. R. Velasco, *Analysis and design of multiagent systems using MAS-CommonKADS*, Intelligent Agents IV Agent Theories, Architectures, and Languages Lecture Notes in Computer Science Volume 1365, pp 313-327, 1998.

[ISO01] ISO/IEC 9126-1 Software Engineering – Product Quality – Part 1: Quality Model. Geneva, Switzerland: International Organization for Standardization, 2001.

[ISO03a] ISO/IEC 9126-2 Software Engineering – Product Quality – Part 2: External Metrics. Geneva, Switzerland: International Organization for Standardization, 2003.

[ISO03b] ISO/IEC 9126-2 Software Engineering – Product Quality – Part 3: Internal Metrics. Geneva, Switzerland: International Organization for Standardization, 2003.

[ISO11] ISO/IEC 25010:2011: Systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – system and software quality models, 2011

[Jac92] I. Jacobson, M. Christerson, P. Jonsson, and Övergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. ACM Press, 1992.

[Jer06] T. Jerraya, Réutilisation des protocoles d'interaction et démarche orientée modèles pour le développement multi-agents, Thèse de doctorat de l'Université de Reims Champagne Ardenne, 2006.

[Jon08] C. Jones, *Applied Software Measurement - Global Analysis of Productivity and Quality*, Third Edition, The McGraw-Hill, 2008.

[Jou08] H. Joumaa, Y. Demazeau, J. M. Vincent, *Evaluation of Multi-Agent Systems: The case of Interaction,* Proceedings of the 3rd International Conference on Information & Communication Technologies: from Theory to Applications (ICTTA'08), Damascus, Syria, IEEE Computer Society, 2008.

[Jul04] V. Julián, J. Soler, M.C. Moncho, V. Botti, *Real-Time Multi-Agent System Development and Implementation,* Frontiers in Artificial Intelligence and Applications Vol. 113, 2004.

[Kab02] H. Kabaili, R. Keller, F. Lustman, *Class Cohesion as predictor of changeability: An Empirical Study*, In F. B. Abreu, B. H. Sellers, M. Piattini,

G. Poels, H. A. Sahraoui (Eds), Quantitative Approaches in Object-Oriented Software Engineering, Object-Oriented Technology, LNCS 2323, 2002.

[Kad09] E. Kaddoum, M.P. Gleizes, J.P. Georgé, P. Glize, G. Picard, *Analyse des critères d'évaluation de systèmes multi-agents adaptatifs*, Journées Francophones sur les Systèmes Multi-Agents (JFSMA'09), 2009.

[Kan02] S. H. Kan, *Metrics and Models in Software Quality Engineering*, Second Edition, Addison Wesely, 2002.

[Kan06] R. K. Kandt, *Software Engineering Quality Practices*, Auerbach Publications, 2006.

[Kic97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. M. Loingtier, J. Irwin, *Aspect-Oriented Programming*, In proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241, 1997.

[Kir06a] S. Kirn, O. Herzog, P. Lockemann (Eds), *Multiagent Engineering: Theory and Applications in Enterprises*, Springer, 2006.

[Kir06b] S. Kirn, *Flexibility of Multiagent Systems*, In S. Kirn, O. Herzog, P. Lockemann (Eds), Multiagent Engineering: Theory and Applications in Enterprises, Springer, 2006.

[Klü08] F. Klügl, *Measuring Complexity of Multi-Agent Simulations – An Attempt Using Metrics*, Languages, Methodologies and Development Tools for Multi-Agent Systems, LNCS 5118, 2008.

[Kra04] S. Kramer, et H. Kaindl, *Coupling and Cohesion Metrics for Knowledge-Based Systems Using Frames and Rules*, In ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 13, Issue 3, 2004.

[Kum09] A. Kumar, P. S. Grover, R. Kumar, *A Quantitative Evaluation of Aspect-Oriented Software Quality Model*, ACM SIGSOFT Software Engineering Notes Volume 34, Number 5, September 2009.

[Kum12] P. Kumar, *Aspect-Oriented Software Quality Model: The AOSQ Model*, Advanced Computing: An International Journal (ACIJ), Vol.3, No.2, 2012.

[Lad03] R. Laddad, *AspectJ in Action*, Manning Publications, 2003.

[Lai06] L. M. Laird et M. C. Brennan, *Software Measurement and Estimation A Practical Approach*, John Wiley & Sons Publication, 2006.

[Lee98] L. C. Lee, H. S. Nwana, D. T. Ndumu, P. D. Wilde, *The stability, scalability and performance of multi-agent systems*, BT Technol Journal, Vol 16, No 3, 1998.

[Leg03] F. Legras, *Organisation dynamique d'équipes d'engins autonomes par écoute flottante*, Thèse de doctorat de l'université de Toulouse, 2003.

[Lew10] P. Lew, L. Olsina, L. Zhang, *Quality, Quality in Use, Actual Usability and User Experience as Key Drivers for Web Application Evaluation*, Dans Proceeding of 10th International Conference, ICWE 2010, Web Engineering - LNCS 6189, 2010.

[Lin01] J. Lin, *Iterative Software Engineering for Multiagent Systems: The MASSIVE Method*, Springer 2001.

[Lin07] H. Lin (Ed), *Architectural Design of Multi-Agent Systems: Technologies and Techniques*, Information science reference, 2007.

[Luc08] M. Luck et L. Padgam (Eds), *Agent-Oriented Software Engineering VIII - 8th International Workshop*, AOSE 2007 Honolulu, HI, USA, May 14, 2007, Lecture Note on Computer Science, Springer, 2008.

[Luy96] M.R. Lyu, (Ed.), *Handbook of Software Reliability Engineering*, IEEE Computer Society Press/McGraw-Hill, Silver Spring/New York, 1996.

[Mag10] I. G. Magariño, M. Cossentino, V. Seidita, *A metrics suite for evaluating agent-oriented architectures*, Proceedings of the 2010 ACM Symposium on Applied Computing, 2010.

[Mah14] S. Mahar, P. K. Bhatia, *Measuring the Intelligence of Software Agent*, International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 6, 2014.

[Mar14a] T. Marir, F. Mokhati, H. B. Seridi, *Do We Need Specific Quality Models for Multi-Agent Systems? - Toward Using the ISO/IEC 25010 Quality Model for MAS*, A. Holzinger, T. Libourel, L. A. Maciaszek, S. J. Mellor (Eds.): ICSOFT-EA 2014 - Proceedings of the 9th International Conference on Software Engineering and Applications, Vienna, Austria, 29-31 August, 2014.

[Mar14b] T. Marir, F. Mokhati, H. B. Seridi, Z. Tamrabet, *Complexity Measurement of Multi-Agent Systems*, J. P. Müller, M. Weyrich, A. L. C. Bazzan (Eds.): Proceedings Multiagent System Technologies - 12th German Conference, MATES 2014, Stuttgart, Germany, September 23-25, 2014, MATES 2014, Springer, 2014.

[Mar14c] T. Marir, *Quality Assessment of Multi-Agent Systems*, S. Lehnhoff (Ed), MATES Doctoral Consortium 2014, Technical Report IfI-14-04, Clausthal University of Technology, 2014. http://www.in.tu-clausthal.de/fileadmin/homes/techreports/ifi1404lehnhoff.pdf

[Mar15] T. Marir, F. Mokhati, H. B. Seridi, Y. Acid, M. Bouzid, *QM4MAS: A Quality Model for Multi-Agent Systems*, à apparaître Int J. Computer Applications In Technology - Special Issue on Current Trends Improvements in software Engineering Practices, 2015.

[McC76] T. J. McCabe, *A Complexity Measure*, IEEE Transactions on Software Engineering, VOL. SE-2, NO.4, December 1976.

[McC77] J. A. McCall, P. K. Richards, G. F. Walters, *Factors in Software Quality*, National Technical Information Service, Springfield. 1977.

[McC87] J. McCarthy, *Ascribing Mental Qualities to Machines*. Technical Report, Stanford University AI Lab, Stanford, CA 94305, 1987.

[Mit13] S. Mittal, P. K. Bhatia, *Software Component Quality Models from ISO 9126 Perspective: A Review*, IJMRS's International Journal of Engineering Sciences, Vol. 02, Issue 02, 2013.

[Moh14] A. H. Mohamed, *Integrated Agent-based and Case-based Reasoning System*, Arab Journal of Nuclear Science and Applications, 47(3), (24-31), 2014.

[Mül97] H. J. Müller, *Towards Agent Systems Engineering*. In: International Journal on Data and Knowledge Engineering, Special Issue on Distributed Expertise 23, pp. 217-245, 1997.

[Nai08]: K. Naik and P. Tripathy, *Software testing and quality assurance – Theory and practice*, Johan Wiley and Sons, 2008.

[Oul91] M. Ould, *Quality Control and Assurance*, in Handbook of Software Engineering, McDermid, J. (Ed.), Butterworth, London, 1991.

[Pav06] J. Pavón, *INGENIAS : Développement Dirigé par Modèles des Systèmes Multi-Agents*, Dossier d'Habilitation à Diriger des Recherches de l'Université Pierre et Marie Curie, Universidad Complutense Madrid, 2006.

[Pic04] G. Picard, *Méthodologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalité émergente*, Thèse de doctorat de l'université Paul Sabatier, 2004.

[Pok05] A. Pokahr, L. Brauhach, W. Lamersdorf, *Jadex: A BDI Reasoning Engine*, Dans R. H. Bordini, M. Dastani, J. Dix, A. F. Seghrouchni (Eds), *Multi-Agent Programming Languages: Tools and Applications*, Springer, 2005.

[Pok14] A. Pokahr, L. Braubach, C. Haubeck, J. Ladiges, *Programming BDI Agents with Pure Java*, Dans J. P. Müller, M. Weyrich, A. L. C. Bazzan (Eds), 12th German Conference on Multiagent System Technologies, LNCS 8732, Springer, 2014.

[Pre01] R. S. Pressman, *Software Engineering: A practitioner's approach*, 5th Edition, McGraw-Hill, 2001.

[Rad11] F. Radulovic, R. G. Castro, *Towards a Quality Model for Semantic Technologies*, Computational Science and Its Applications - ICCSA 2011, LNCS 6786, 2011.

[Rao95] A. Rao, M. P. Georgeff, *BDI Agents: From Theory to Practice*, Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), Menlo Park, California, AAAI Press, pp. 312-319, 1995.

[Reg05] A. Regayeg, A. H. Kacem, M. Jmaiel, *Towards a Formal Methodology for Designing Multi-agent Applications*, Multiagent System Technologies 2005, Lecture Notes in Computer Science Volume 3550, pp 153-164, 2005.

[Rej05] L. Rejeb, *Simulation Multi-Agents de Modèles Economiques - Vers des Systèmes Multi-Agents Adaptatifs*, Thèse de doctorat de l'université de Reims Champagne-Ardennes, France, 2005..

[Ric11] A. Ricci, M. Piunti, M. Viroli, *Environment programming in multi-agent systems: an artifact-based perspective*, Autonomous Agent & Multi-Agent Systems (2011) 23:158–192, Springer, 2011.

[Rou14] A. Rousset, B. Herrmann, C. Lang, *Etude comparative des plateformes parallèles pour systèmes multi-agents*, P. Felber, L. Philippe, E. Riviere, A. Tisserand (Eds). ComPAS 2014 : conférence en parallélisme, architecture et systèmes, 2014.

[Rus03] S. J. Russell et P. Norvig, *Artificial Intelligence. A Modern Approach.* Prentice Hall, 2003.

[San11] A. Santi, M. Guidi, A. Ricci, *JaCa-Android: An Agent-Based Platform for Building Smart Mobile Applications*, Dans M. Dastani, A. F. Seghrouchni, J. Hübner, J. Leite (Eds), Languages, Methodologies, and Development Tools for Multi-Agent Systems, LNCS 6822, Springer, 2011.

[Sav03] M. Savall, "*Une architecture d'agents pour la simulation, Le modèle YAMAM et sa plate-forme Phoenix"* THÈSE Doctorat de l'INSA de Rouen, juin2003.

[Sch94] A. Th. Schreiber, B. J. Wielinga, and J. M. Akkermans W. Van de Velde, *CommonKADS: A comprehensive methodology for KBS development*, Deliverable DM1.2a KADSII/M1/RR/UvA/70/1.1, University of Amsterdam, Netherlands Energy Research Foundation ECN and Free University of Brussels, 1994.

[Sch01] M. Schumacher, *Objective Coordination in Multi-Agent System Engineering Design and Implementation,* Lecture Notes in Computer Science Vol. 2039: Lecture Notes in Artificial Intelligence, Springer, 2001.

[Sch14] E. Schneider, O. Balas, A. T. Özgelen, E. I. Sklar, S. Parsons, *An Empirical Evaluation of Auction-based Task Allocation in Multi-robot Teams (Extended Abstract)*, Dans A. Lomuscio, P. Scerri, A. Bazzan, M. Huhns (Eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), Paris, France, May 5-9, 2014.

[Sea69] J. Searle, *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, 1969.

[Sec03] Y. Secq, *RIO: Rôles, Interactions et Organisations une méthodologie pour les systèmes multi-agents ouverts*, Thèse de doctorat de l'université des Sciences et Technologies de Lille, 2003.

[Sel05] B. H. Sellers et P. Giorgini (Eds), *Agent-Oriented Methodologies*, IGI-Global, 2005.

[Sha14] M. Sharma, Sapna, *A Study of Agent Oriented Metrics*, International Journal of Engineering Science Invention Research & Development, Vol. I Issue II, 2014.

[Sho93] Y. Shoham, *Agent Oriented Programming*, Journal of Artificial Intelligence 60 (1), 1993.

[Siv12] N. Sivakumar et K. Vivekanandan, *Measures for Testing the Reactivity Property of a Software Agent*, International Journal of Advanced Research in Artificial Intelligence, Vol. 1, No. 9, 2012.

[Som07]: I. Sommerville, *Software Engineering*, 8th Edition, China Machine Press, 2007.

[Ste09] L. Sterling et K. Taveter, *The Art of Agent-Oriented Modeling*, The MIT Press, 2009.

[Sto00] P. Stone et M. Veloso, *Multiagent Systems: A Survey from a Machine Learning Perspective*, Autonomous Robots, Volume 08, Issue 03, Kluwer Academic Publishers, 2000.

[Sur03] W. Suryn, et A. Abran, *ISO/IEC SQuaRE: The second generation of standards for software product quality*, In Software Engineering and Applications (SEA' 2003), 2003.

[Sur14] W. Suryn, *Software Quality Engineering - A Practitioner's Approach*, John Wiley & Sons, 2014.

[Tah10] A. Tahir, R. Ahmad, K. M. Kasirun, *Maintainability Dynamic Metrics Data Collection Based on Aspect-Oriented Technology*, Malaysian Journal of Computer Science, Vol. 23(3), 2010.

[Tim06] I. J. Timm, T. Scholz, H. Knublauch, *The Engineering Process*, In Multiagent Engineering: Theory and Applications in Enterprises, S. Kirn, O. Herzog, P. Lockemann, Springer, 2006.

[Tob04] R. Tobias, C. Hofmann, *Evaluation of free java-libraries for social-scientific agent based simulation*, Journal of Artificial Societies and Social Simulation, vol. 7, n1, 2004.

[Tra07] J. Tranier, *Vers une vision intégrale des systèmes multi-agents - Contribution à l'intégration des concepts d'agent, d'environnement, d'organisation et d'institution*, Thèse de doctorat, Université des Sciences et Techniques du Languedoc, Décembre 2007.

[Tra08] Q. N. N. Tran, G. Low, *MOBMAS: A methodology for ontology-based multi-agent systems development*, Information and Software Technology, Elsevier, Volume 50, Issues 7–8, June 2008.

[Twu14] P. Twu, Y. Mostofi, M. Egerstedt, *A Measure of Heterogeneity in Multi-Agent Systems*, American Control Conference (ACC), 2014.

[Wag13] S. Wagner, *Software Product Quality Control*, Springer-Verlag Berlin Heidelberg, 2013.

[Wei99] G. Weiss (Ed), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, 1999.

[Wey04] D. Weyns, H. V. D. Parunak, F. Michel, T. Holvoet, J. Ferber, *Environments for Multiagent Systems State-of-the-Art and Research Challenges*. In D. Weyns, H. V. D. Parunak, F. Michel (Eds), Third International Workshop (E4MAS 2004), volume 3347 of Lecture Notes in Artificial Intelligence, pages 1- 47. Springer, 2006.

[Wey07] D. Weyns, A. Omicini et J. Odell, *Environment as a first class abstraction in multiagent systems*, Autonomous Agent and Multi-Agent Systems, Springer, 2007.

[Wil01] K. Wilber,"*A Theory of Everything: An Integral Vision for Business, Politics, Science and Spirituality"*, Shambhala, October 2001.

[Woo95] M. Wooldridge et N. R. Jennings, *Intelligent agents: Theory and practice*, The Knowledge Engineering Review, 1995.

[Woo00] M. Wooldridge, N. R. Jennings, D. Kinny, *The Gaia Methodology for Agent-Oriented Analysis and Design*, Journal of Autonomous Agents and Multi-Agent Systems. 3(3):285-312. 2000.

[Woo02a] M. Wooldridge, *Intelligent Agents: The Key Concepts*, Multi-Agent Systems and Applications II, Lecture Notes in Computer Science Volume 2322, 2002.

[Woo02b] M. Wooldridge, *An Introduction to Multiagent Systems*, JOHN WILEY & SONS, LTD, 2002.

[Yok12] M. Yokoo, Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-agent Systems, Springer Publishing Company, 2012.

[Zei07] B. Zeiss, D. Vega, I. Schieferdecker, H. Neukirchen, J. Grabowski, *Applying the ISO 9126 Quality Model to Test Specifications - Exemplified for TTCN-3 Test Specifications*, In Software Engineering 2007 (SE 2007). Lecture Notes in Informatics (LNI) 105, 2007.

[Zhu03] H. Zhu, *A formal specification language for agent-oriented software engineering*, Proceedings of AAMAS 2003, pp. 1174–1175, Melbourne, Australia, 2003.

ملخص

هذه الأطروحة تعالج إشكالية ضمان الجودة بالنسبة للأنظمة متعددة الأعوان. و قد ركزنا فيها على تمثيل الجودة التي تمثل حجر الأساس في تعريف جودة الأنظمة متعددة الأعوان. لقد اهتم عملنا هذا باستخدام نماذج الجودة القياسية كنماذج قابلة للتوسيع لتشمل خصائص نماذج البرامج المختلفة. هذه المقاربة تسمح بتجاوز عائق تعدد النماذج الموجودة. و بالتالي، فقد اقترحنا نموذجين للجودة خاصين بالأنظمة المتعددة الأعوان. أول نموذج (يسمى QM4MAS) يرتكز على نموذج الجودة القياسي ايزو 9126. النموذج الثاني يعتبر خطوة أولى لتطوير نموذج جودة مرتكز على النموذج القياسي ايزو 25010 لأنه يرتكز على مفهوم العون الضعيف. كخطوة ثانية في بحثنا، قمنا بدراسة تطبيق النموذج QM4MAS على المنصتين JADE و DIMA من خلال اقتراح مجموعة من المقاييس. و كآخر إسهاماتنا في هذا الموضوع، عرضنا دراسة و قياس التعقيد كميزة مؤثرة بشكل واضح على جودة الأنظمة متعددة الأعوان.

كلمات مفتاحية : ضمان الجودة، الأنظمة متعددة الأعوان، نماذج الجودة، مقاييس الجودة، ايزو 9126، ايزو 25010، تعقيد الأنظمة متعددة الأعوان.

Abstract

This thesis addresses the problem of the quality assurance of multi-agent systems. We focus on the modeling of the quality which represents the basic building block for defining the quality of MAS. Our work focuses on the standard quality models as extensible ones allowing supporting the features of the different software paradigms. This approach allows overcoming the disadvantage of the diversity of existing models. Hence, we proposed two specific models of the quality of MAS. The first model (called QM4MAS) is based on the standard quality model ISO/IEC 9126. The second one is considered as a first step in order to develop a quality model based on the standard quality model ISO/IEC 25010 because it is based on the weak agency notion. Next, we investigated the applicability of the QM4MAS model on JADE and DIMA multi-agent platforms by proposing a set of measures. The last contribution presented in this thesis is to study and measure of the complexity as an attribute that significantly affects the quality of MAS.

Keywords: Quality assurance, Multi-Agent Systems, Quality models, Quality metrics, ISO/IEC 9126, ISO/IEC 25010, Complexity of MAS

Résumé

Cette thèse traite le problème d’assurance qualité des systèmes multi-agents. Nous nous focalisons sur la modélisation de la qualité qui représente la brique de base pour la définition de la qualité des SMA. Notre travail met l’accent sur les modèles standards de qualité en tant que modèles extensibles permettant de tenir compte de spécificités de différents paradigmes logiciels. Une telle approche permet de dépasser l’inconvénient de la diversité des modèles existants. Ainsi, nous avons proposé deux modèles spécifiques pour la qualité des SMA. Le premier modèle (appelé QM4MAS) est basé sur le modèle standard ISO/IEC 9126, tandis que le deuxième modèle est basé sur le modèle standard ISO/IEC 25010. Ce dernier représente, dans sa première version, le premier pas vers le développement d’un modèle tenant compte de la plupart de spécificités des SMA. Dans sa version actuelle, ce modèle ne supporte que la notion de l’agent faible. Ensuite, nous avons étudié l’applicabilité du modèle QM4MAS sur les plateformes multi-agents JADE et DIMA à travers la proposition d’un ensemble de mesures. La dernière contribution présentée dans cette thèse consiste à l’étude et la mesure de la complexité en tant qu’attribut influant fortement sur la qualité des SMA.

Mots clés : Assurance qualité, Systèmes Multi-Agents, Modèles de qualité, Métriques de qualité, ISO/IEC 9126, ISO/IEC 25010, Complexité des SMA.