RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université LARBI BEN M'HIDI - Oum El Bouaghi

Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie

Département de Mathématiques et Informatique

N ° d'ordre :

Série

Une Approche de Maintenance préventive des Systèmes Multi-Agents

THÈSE présentée par

M^me^ TORCHANE née GHRIEB Nawel

Pour l'obtention du diplôme de

DOCTORAT EN SCIENCE en Informatique

(Option : **Computation & Information**)

Soutenue publiquement le : 16 février 2022

Devant le jury composé de :

Président	**BENABOUD Rohallah**	***Professeur***	*Université d'Oum El Bouaghi*
Directeur de thèse	**MOKHATI Farid**	***Professeur***	*Université d'Oum El Bouaghi*
Co-directeur de thèse	**GUERRAM Tahar**	***MCA***	*Université d'Oum El Bouaghi*
Examinateur	**MAAROUK Toufik Messaoud**	***MCA***	*Université de Kenchela*
Examinateur	**HOUASSI Hichem**	***MCA***	*Université de Kenchela*
Examinateur	**MAHDAOUI Rafik**	***MCA***	*Université de Kenchela*

2021 - 2022

Remerciements

*Je tiens tout d'abord à exprimer ma sincère gratitude et reconnaissance à mon encadreur Monsieur **MOKHATI Farid**, Professeur à l'Université d'Oum El-Bouaghi, pour m'avoir fait bénéficier de son expérience et de son savoir. Qu'il trouve ici l'expression de mes plus vifs et sincères remerciements pour ses disponibilités, son aide constante et ses conseils. J'ai apprécié l'atmosphère scientifique qu'il m'a toujours offert et le temps qu'il m'a consacré tout au long de mon travail de thèse.*

*Je voudrais remercier également mon co-encadreur de thèse, Monsieur **GUERRAM Tahar** Maître de Conférence A à l'Université d'Oum El-Bouaghi, pour son apport scientifique.*

Mes sincères remerciements sont adressés à Monsieur BENABOUD Rohallah, Professeur à l'Université d'Oum El-Bouaghi, pour avoir accepté de présider le jury de ma thèse. Mes remerciements vont également aux membres de jury qui m'ont fait l'honneur d'avoir accepté d'évaluer mes travaux de thèse :

DR. MAAROUK Toufik Messaoud

DR. HOUASSI Hichem

DR. MAHDAOUI Rafik,

*J'adresse mes plus vifs remerciements à mon **père**, qu'aucun remerciement ne saurait exprimer l'estime et le respect que j'ai toujours eu pour lui.*

*Mes remerciements vont également à ma **mère**, symbole de bonté par excellence et source de tendresse qui n'a jamais cessé de m'encourager et de prier pour moi.*

*C'est envers mon **mari**, que ma reconnaissance est la plus grande. Son soutien, sa compréhension et son aide constante m'ont permis de me concentrer pleinement sur mon travail de thèse.*

*Mes remerciements vont également à mes trois perles **Rania, Lina et Nada** et à mon fils adoré **Anis** qui ont fait beaucoup de sacrifices pour me permettre de mener à bien mon travail de thèse.*

*Enfin, j'adresse mes plus affectueux remerciements à mes deux **frères** et à ma **sœur** pour leurs encouragements et leurs soutiens.*

Dédicaces

À mes parents.

À mon mari.

À mes puces : Rania, Lina, Nada et Anis.

À mes frères et sœur Tayeb, Toufik et Ahlem.

Ainsi qu'à tous ceux qui m'aiment.

ملخص

تعد صيانة البرامج الوقائية نشاطًا مهمًا للبرامج يتضمن التغييرات والتحديثات لمنع حدوث مشكلات البرامج الخطيرة في المستقبل. في سياق الأنظمة متعددة الوكلاء، تم تجاهل هذا النشاط تمامًا. الخصائص المتأصلة في الأنظمة متعددة العوامل (على سبيل المثال: الاستقلالية، الاستباقية، التفاعلية والقدرة على التكيف، وما إلى ذلك) تجعل من الصعب تحقيق صيانتها. في هذه الرسالة، نقترح طريقتين مختلفتين للصيانة الوقائية المشروطة. النهج الأول المقترح، يسمح بالصيانة الوقائية المشروطة للتطبيقات متعددة الوكلاء ويتكون من قياس مقياسين للجودة (الاستقلالية والتواصل الاجتماعي) للتطبيق قيد التشغيل بطريقة ديناميكية ومستمرة باستخدام كودAspectJ وتحذير مهندس الصيانة في حالة اكتشاف تراجع غير طبيعي في جودة MAS، من أجل تجنب الضرر المحتمل. النهج المقترح مدعوم بأداة برمجية قمنا بتطويرها وأطلقنا عليها اسم PMMAS(الصيانة الوقائية للأنظمة متعددة العوامل). من ناحية أخرى، يتعلق نهج الصيانة المقترح الثاني بالأنظمة التنظيمية متعددة الوكلاء (OCMAS) ويتكون من قياس مقاييس الجودة الأخرى الخاصة بتنظيم هذه الأنظمة، مثل كفاءة الوكلاء في أداء أدوارهم. بمجرد اكتشاف انخفاض في الكفاءة، يسمح نظامنا بالتدخل باستخدام تقنيات إعادة التنظيم من أجل الحفاظ على جودة التطبيق. تم التحقق من قابلية تطبيق نظام MAINOMACSالمقترح للأنظمة القائمة على النموذج التنظيمي OMACS من خلال دراسة حالة .

الكلمات الدالة:

الصيانة الوقائية المشروطة، نموذج الإصلاح السريع، الأنظمة متعددة العوامل ،OCMAS، البرمجة الموجهة للجانب، قياس الجودة، JADE

Résumé

La maintenance préventive des logiciels est une activité logicielle importante qui consiste à inclure des modifications et des mises à jour afin d'éviter de futurs problèmes graves du logiciel. Dans le contexte des systèmes multi-agents, une telle activité est complètement omise. Les spécificités inhérentes aux systèmes multi-agents (ex : autonomie, proactivité, réactivité, adaptabilité, etc.) rendent leur maintenance difficile à réaliser. Nous proposons, dans cette thèse, deux approches différentes de maintenance préventive conditionnelle. La première approche proposée, permet la maintenance préventive conditionnelle des applications multi-agents et consiste à mesurer deux métriques de qualité (autonomie et sociabilité) de l'application en cours d'exécution de manière dynamique et continue en utilisant le code AspectJ et à avertir le mainteneur en cas de détection de régression anormale de la qualité MAS, afin d'éviter d'éventuels dommages. L'approche proposée est soutenue par un outil logiciel que nous avons développé et appelé PMMAS (Preventive Maintenance of Multi-Agent Systems). En revanche, la seconde approche de maintenance proposée concerne les systèmes multi-agents organisationnels (OCMAS) et consiste à mesurer d'autres métriques de qualité spécifiques à l'organisation de ces systèmes telle que l'efficacité des agents à exécuter leurs rôles. Dès qu'une régression de l'efficacité est détectée notre système permet d'intervenir en utilisant les techniques de la réorganisation afin de préserver la qualité de l'application. L'applicabilité du système MAINOMACS proposé pour les systèmes se basant sur le modèle organisationnel OMACS a été validée par une étude de cas.

Mots-clés :

Maintenance Préventive Conditionnelle, Modèle de Réparation Rapide, Systèmes Multi-Agents, OCMAS, Programmation Orientée Aspect, Mesure de la Qualité, JADE.

Abstract

Preventive software maintenance is an important software activity that involves including changes and updates to prevent serious software problems in the future. In the context of multi-agent systems, such activity is completely omitted. The specificities inherent to multi-agent systems (eg: autonomy, proactivity, reactivity, adaptability, etc.) make their maintenance difficult to achieve. In this thesis, we propose two different approaches to conditional preventive maintenance. The first proposed approach allows conditional preventive maintenance of multi-agent applications and consists To measure two quality metrics (autonomy and sociability) of the running application in a dynamic and continuous manner using AspectJ code and to warn the maintainer in case of detection of abnormal regression of the MAS quality, in order to avoid possible damage. The proposed approach is supported by a software tool we have developed and baptized PMMAS (Preventive Maintenance of Multi-Agent Systems). On the other hand, the second proposed maintenance approach concerns multi-agent organizational systems (OCMAS) and consists in measuring other quality metrics specific to the organization of these systems, such as the efficiency of agents in performing their roles. As soon as a decrease in efficiency is detected, our system allows us to intervene using the techniques of reorganization in order to preserve the quality of the application. The applicability of the proposed MAINOMACS system for systems based on the OMACS organizational model was validated by a case study.

Key words :

Conditional preventive maintenance, Quick repair model, Multi-Agent Systems, OCMAS, Aspect-Oriented Programming, Quality Measurement, JADE.

Table des matières

Liste des figures

Liste des tables

INTRODUCTION GÉNÉRALE

1. Contexte général et problématique

La maintenance logicielle est une tâche importante et cruciale dans le cycle de vie du développement logiciel. Bien que la maintenance des systèmes logiciels existants puisse représenter plus d'efforts que toute autre activité de génie logiciel, la maintenance logicielle est encore une phase négligée dans le processus de génie logiciel (Singh & Goel, 2007). La recherche sur la maintenance logicielle, par rapport aux autres phases du processus de génie logiciel, est rare. De plus, il est largement admis que les méthodes et techniques actuelles de maintenance logicielle sont incapables de faire face à la complexité inhérente à la maintenance des systèmes logiciels complexes.

L'efficacité de la maintenance des systèmes logiciels est un enjeu économique majeur pour leur exploitation. Les principales difficultés et sources d'inefficacité résident dans le choix des actions de maintenance qui sont coûteuses pour plusieurs raisons.

Tout d'abord, les actions de maintenance nécessitent souvent un arrêt de fonctionnement du système. Dans ce cas, durant toute la phase de maintenance, le système n'est pas opérationnel. Plus la phase de maintenance est longue, plus elle est coûteuse dû à l'indisponibilité du système. Par conséquent, la phase de maintenance doit idéalement être effectuée, sans tâtonnement. La décision d'une action de maintenance est très complexe et doit reposer sur une surveillance et une analyse intelligente de l'état du système. Un diagnostic est alors nécessaire pour déterminer le plus précisément possible les parties du système qui doivent être réparés. Moins le diagnostic est ambigu, plus les opérations de maintenance sont efficaces.

La seconde raison pour laquelle une maintenance peut être coûteuse concerne les cas d'urgence dans lesquels la sécurité ou l'accomplissement de la fonction du système sont mis en jeu. En effet, lorsqu'un composant du système tombe soudainement en panne et que le système ne peut plus réaliser sa fonction, des actions de maintenance doivent être automatiquement réalisées pour remettre le système en état de fonctionnement. Ces actions imprévues sont naturellement plus coûteuses car les besoins et services pour la maintenance n'ont pas été anticipés et doivent être rapidement disponibles. Pour minimiser l'occurrence de ce genre de situation, une maintenance préventive peut être envisagée. Les pannes des différents composants du système peuvent être anticipées et corrigées avant de générer de trop importants dégâts qui pourraient provoquer un arrêt imprévu du système.

Dans le contexte des systèmes multi-agents, les problèmes de maintenance sont accentués et sont plus compliqué à résoudre. Les spécificités inhérentes aux systèmes multi-agents (e.g. autonomie, proactivité, réactivité, adaptabilité, etc.) rendent leur maintenance difficile à réaliser. De plus les environnements d'application imprévisibles rendent les systèmes multi-agents vulnérables à des pannes individuelles qui peuvent réduire considérablement la capacité du système à accomplir son objectif. Les agents eux-mêmes peuvent présenter des propriétés particulières qui n'étaient pas initialement prévues.

Certains agents peuvent alors devenir incapables de remplir les rôles qui leurs sont assignés. Par exemple, des dangers ou même des actions malveillantes pourraient rompre les liens de communication entre les agents. Alternativement, les agents ne peuvent pas s'acquitter de leurs rôles en raison : de capacités insuffisantes des agents pour accomplir leurs rôles, ou une surcharge d'agent entraînant un manque à certains de ses objectifs par exemple.

À mesure que les systèmes multi-agents se développent, la configuration et le réglage de ces systèmes peuvent devenir aussi complexes que les problèmes qu'ils prétendent résoudre. Un système multi-agents robuste devrait donc être maintenu afin de s'adapter aux environnements, se remettre d'une panne et s'améliorer au fil du temps afin d'éviter les situations indésirables. Par conséquent, la question centrale abordée dans cette thèse peut être formulée comme suit : « Comment anticiper les pannes d'un SMA afin d'assurer la continuité de ses activités de manière à ce que l'échec de certains des agents à remplir les objectifs correspondants n'affecte pas drastiquement le fonctionnement du système ? ».

Afin de répondre à ces préoccupations, nous nous intéressons dans ce travail à la maintenance préventive des SMA afin de permettre à ces systèmes d'atteindre leurs objectifs en toute sécurité. La maintenance préventive d'un SMA est une activité logicielle importante qui consiste à inclure des modifications et des mises à jour afin d'éviter de futurs problèmes graves du SMA en exécution. Les SMA qui sont considérés, sont composés d'agents totalement imprévisibles et nécessitent des techniques adaptées pour être surveillés de manière dynamique à l'exécution.

Cette thèse porte sur le développement d'une approche pour la maintenance des systèmes multi-agents. Cette approche fournit des métriques, des directives et des procédures pour effectuer une variété d'activités effectuées pendant la maintenance préventive du logiciel.

Dans ce travail, notre premier objectif porte sur l'évaluation des politiques de maintenance et l'étude des spécificités des systèmes multi-agents afin de bien choisir les métriques de qualité à évaluer et d'optimiser les décisions de maintenance durant la phase d'exploitation du système considéré. Notre second objectif consiste en la mise en place d'une architecture de supervision (monitoring). Cette architecture intègre des capacités de diagnostic dans l'objectif d'aider à la prise de décisions des actions de maintenance (voir même à appliquer des actions de maintenance de façon automatique dans certains cas). Ainsi nous présentons dans ce travail deux systèmes de maintenance à partir desquels il est possible d'assurer un couplage original des techniques du monitoring, d'aide à la décision et de contrôle (réparation).

2. Plan de la thèse

La présente thèse est organisée en quatre chapitres, les deux premiers chapitres sont consacrés à la présentation d'un état de l'art sur les travaux en liaison avec notre

problématique. Ensuite, nos contributions seront présentées dans les deux derniers chapitres :

Systèmes multi-agents : dans ce chapitre des concepts d'ordre général seront présentés autour de l'agent et des caractéristiques inhérentes aux SMA. Nous allons étudier les SMA d'un point de vue organisationnel en traitant les différents modèles organisationnels existants. Ensuite, nous mettons l'accent sur le problème de réorganisation pour les SMA organisationnels (OCMAS) et finalement nous exposerons les différentes phases du processus de réorganisation.

Maintenance de systèmes complexes : présente le cadre de la maintenance des systèmes complexes et montre l'intérêt d'une stratégie de maintenance préventive. Les principaux travaux sur la maintenance préventive sont également présentés. Finalement, une étude comparative sera exposée et examinée à la fin du chapitre. Une analyse de ces travaux permet de mettre en évidence les propriétés et les limites des politiques rencontrées et les objectifs que nous cherchons à atteindre.

Une approche de maintenance préventive pour les systèmes multi-agents : dans ce chapitre nous allons présenter des préliminaires de l'approche proposée. Ainsi, nous allons introduire le langage AspectJ. Ensuite nous présenterons l'approche présentée pour la maintenance préventive des SMA. L'approche proposée est supportée par un outil logiciel nommé PMMAS (pour Préventive Maintenance for Multi-Agents Systems). Finalement, nous illustrons l'outil que nous avons développé à partir d'une étude de cas concrète et nous montrons son apport par rapport aux travaux discutés sur la maintenance préventive. Finalement, une synthèse bibliographique a été mise en exergue pour discuter les points forts et les limites de notre proposition.

Une approche de maintenance préventive des systèmes multi-agents Centrés organisation (OCMAS) : Le dernier chapitre est consacré à la proposition d'une approche de maintenance préventive générique pour les systèmes OCMAS. Ensuite, nous proposons un système de maintenance préventive pour les systèmes SMA basés sur le modèle OMACS. Notre proposition est nommée MAINOMACS pour (MAINtenance of Multi-Agent System based OMACS). Notre proposition est validée sur une étude de cas. Un bilan récapitulatif est dressé à la fin du chapitre pour bien montrer l'apport de MAINOMACS par rapport aux solutions existantes.

Chapitre 1

SYSTÈMES MULTI-AGENTS

Sommaire

1. Introduction

Les systèmes multi-agents appartiennent à un domaine de l'intelligence artificielle appelé intelligence artificielle distribuée. Emergé il y a quelques décennies, ce domaine est encore en pleine expansion (Davis, 1980) (Gasser, 1987), (Erceau & Ferber,1993). Les systèmes multi-agents et l'intelligence artificielle distribuée, en général, permettent de traiter des problèmes de taille et de complexité plus importantes que ceux abordés par les approches classiques de l'intelligence artificielle.

Ce premier chapitre présente une synthèse de l'état de l'art sur les SMA et leur organisation. Il sera donc composé de deux parties. La première partie a seulement pour objectif de donner une vision générale du paradigme multi-agents et surtout d'introduire quelques concepts de base communément employés dans les systèmes multi-agents. Dans la deuxième partie de ce chapitre, nous exposons les travaux réalisés autour de la thématique de l'organisation dans le cadre des SMA. Nous commençons par mettre l'accent sur le concept de l'organisation et la manière d'implémenter les SMA organisationnels en se basant sur les modèles organisationnels. Afin de bien exposer notre problématique concernant la maintenance des SMA organisationnels, nous introduisons les notions fondamentales sur la réorganisation dynamique dans les SMA et nous présentons les différents types de réorganisation et les phases principales du processus de réorganisation.

2. Notion de l'agent

Malgré les efforts de standardisation des aspects opérationnels et techniques, la notion d'agent reste encore polémique concernant sa définition. Plusieurs définitions ont été proposées concernant la notion d'agent.

D'après Wooldridge (Wooldridge, 2002) :

« Un agent est une entité informatique située dans un environnement et capable d'agir dessus d'une manière autonome dans le but d'atteindre les objectifs pour lesquels elle a été conçue. »

Ferber & Perrot ont donné une autre définition au terme agent (Ferber & Perrot, 1995) :

« *Un agent est une entité autonome, réelle ou abstraite, qui est capable d'agir sur elle-même et sur son environnement, qui, dans un univers multi-agents, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec d'autres agents*. »

Michael Wooldridge fait la distinction entre deux notions de l'agent, qui sont : l'agent faible et l'agent fort (Wooldridge & Jennings, 1995) :

L'agent faible : est une entité matérielle ou logicielle ayant les caractéristiques suivantes :

- *L'autonomie* : est la capacité de l'agent à se comporter d'une manière individuelle et à avoir un auto-contrôle sur ses actions et sur son état interne.
- *La réactivité* : désigne la capacité de l'agent à percevoir son environnement et à réagir aux changements qui y surgissent, dans le temps adéquat.
- *La sociabilité* : désigne la capacité de l'agent à interagir avec d'autres agents. La coopération, la négociation, la communication et le partage de buts représentent des aspects de la socialité de l'agent.
- *La proactivité* : désigne la capacité de l'agent à avoir l'initiative d'agir afin de réaliser un but

L'agent fort : en plus des caractéristiques de l'agent faible décrites précédemment, l'agent possède des caractéristiques supplémentaires qui sont inspirées de l'être humain comme l'utilisation des notions mentalistes qui sont spécifiées sous forme de connaissances, buts, intentions et obligation (Shoham, 1993).

Aux caractéristiques précédentes, Wooldridge a ajouté les propriétés suivantes :

- la situation (Wooldridge, 2009) : qui désigne l'environnement où se situe l'agent et interagit. Cet environnement est défini par l'ensemble d'objets passifs qui constituent l'entourage de l'agent (Ferber, 1995),

- La mobilité : est la capacité de l'agent à se déplacer d'une plateforme à une autre dans un réseau (White, 1994).

- La véracité : désigne l'hypothèse qu'un agent ne communique pas des informations falsifiées (Galliers, 1988).

- La bienveillance : désigne l'hypothèse qu'un agent ne possède pas des buts contradictoires et que chaque agent essaye d'accomplir sa tâche (Rosenschein & Genesereth, 1985).

- La rationalité : désigne l'hypothèse qu'un agent se comporte dans le but de réaliser ses buts et non à interdire ses buts d'être achevés (Galliers , 1988).

A partir des définitions citées précédemment, nous pouvons distinguer la propriété d'autonomie qui est une propriété importante et spécifique aux agents. Cette particularité des agents engendre d'autres caractéristiques propres aux SMA qui font la différence de ces systèmes avec les autres systèmes informatiques. La notion de sociabilité qui représente la capacité d'un agent à communiquer avec d'autres agents est une notion importante qui a également été abordée dans toutes les définitions.

Les agents disposent de plusieurs propriétés communes, mais ces derniers peuvent être différents. A titre d'exemple, on distingue, les agents stationnaires et les agents mobiles : un agent stationnaire s'exécute sur une même machine, tandis qu'un agent mobile peut se

déplacer à travers un réseau informatique en transportant son code et ses données afin de les exécuter dans un site distant.

Nous pouvons également distinguer les agents réactifs, les agents cognitifs et les agents hybrides. La différence entre les agents réactifs et les agents cognitifs se situe au niveau de leurs architectures internes et concerne leurs manières à traiter les informations perçues. En effet, le comportement des agents réactifs est régi par des réponses immédiates à des stimuli fournis par l'environnement, alors que le comportement des agents cognitifs se base plutôt sur leur capacité à représenter symboliquement leurs environnements, à mémoriser leurs états passés et à planifier leurs actions en tenant compte de leurs connaissances, de leurs croyances et de leurs intentions.

En effet chacun de ces deux types d'agents convient à certains problèmes et moins bien à d'autres. Les chercheurs ont donc essayé de combiner les deux approches afin d'obtenir une architecture hybride (Jarras & Chaib-Draa, 2002). Les agents hybrides sont des agents ayant des capacités cognitives et réactives. Ils conjuguent en effet la rapidité de réponse des agents réactifs ainsi que les capacités de raisonnement des agents cognitifs.

3. Systèmes multi-agents

Plusieurs définitions ont été proposées concernant les systèmes multi-agents. Parmi ces définitions on retient celle de Ferber (Ferber, 1995), qui définit un SMA par :

- Un environnement E, un espace disposant généralement d'une métrique.

- Un ensemble d'objets situés O, auxquels on peut associer une position dans l'environnement E. Ces objets peuvent être perçus, créés, détruits et modifiés par les agents.

- Un ensemble A d'agents qui sont des objets particuliers ($A \subseteq O$), représentant les entités actives du système.

- Un ensemble de relations R qui unissent des objets (et donc des agents) entre eux.

- Un ensemble d'opérations Op permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler des objets de O.

- Des d'opérateurs représentant l'application des opérations OP et la réaction du monde en réaction à cette modification.

Les SMA reposent sur le principe de faire interagir plusieurs agents afin de réaliser un objectif global. Chacun de ces agents dispose d'une représentation partielle de son environnement, et possède des connaissances et des compétences propres qui lui permettent d'évoluer dans un environnement commun de manière à satisfaire son but local.

Les agents d'un SMA sont regroupés par des relations organisationnelles. Chacun de ces agents possède le contrôle d'une partie de l'environnement sous forme d'une "sphère d'influence". L'ensemble de ces sphères d'influence peuvent interférer ensemble ce qui entraine des situations d'interactions entre les agents des diverses organisations. La figure 1.1 présente la structure typique d'un SMA.

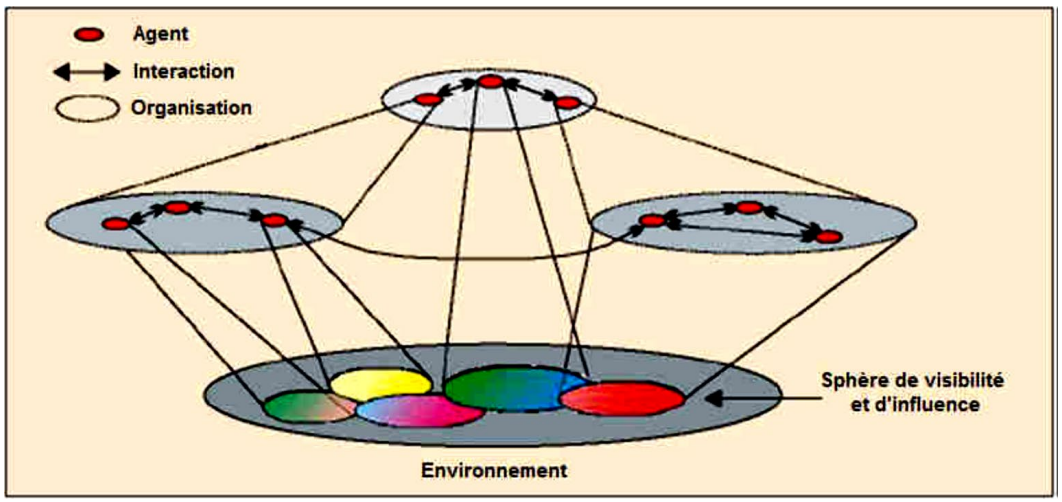


Figure 1.1. Structure typique d'un système multi-agents (Wooldridge, 2002)

4. Organisation dans les SMA

Les SMA ont été considérés comme des sociétés d'agents qui interagissent ensemble et coordonnent leurs comportements afin de réaliser un but global. À cet effet, deux approches ont été distinguées pour la conception des SMA (Ferber, & al., 2004) : les approches centrées agent connu sous le nom ACMAS (Agent Centered Multi Agent Systems) et les approches centrées organisation qui sont connus sous le nom OCMAS (Organization Centered Multi Agent Systems).

Approche centrée agent (ACMAS)

Les approches centrées agents (ACMAS) mettent l'accent sur l'agent individuel et l'étudie selon deux angles distincts : le fonctionnement interne et le comportement externe (Yoo & Briot, 1999). L'étude du fonctionnement interne concerne les caractéristiques de base de l'agent telles que l'autonomie et la réactivité par exemple. Alors que, l'étude du comportement externe concerne la relation de l'agent avec son environnement et avec les autres agents (Mansour, 2007).

Picard et al. (Picard, & al., 2009) ont déclaré que l'approche centrée sur l'agent considère les agents comme le « moteur » de l'organisation du système, et les organisations d'agents existent implicitement en tant que phénomènes émergents observables. L'idée principale de cette approche est que l'organisation est le résultat du comportement émergent

collectif dû à la façon dont les agents agissent selon leurs comportements individuels et à la manière avec laquelle ils interagissent dans un environnement commun partagé et dynamique.

Les principaux problèmes du point de vue ACMAS sont l'imprévisibilité et l'incertitude, car cette approche peut conduire à des comportements émergents indésirables qui peuvent avoir un impact sur les performances du système.

La sécurité est un autre problème des ACMAS qui est dû à la liberté de communication entre les agents dans les ACMAS. Jacques Ferber (Ferber, & al., 2004) montre que cette liberté permet à un type d'agents intrus de pirater le système et d'affecter sa progression d'une manière frauduleuse.

Jennings & Wooldridge (Jennings & Wooldridge, 2000) ont souligné le besoin de recourir à d'autres structures afin de pallier aux problèmes des ACMAS :

"*Another common misconception is that agent-based systems require no real structure. While this may be true in certain cases, most agent systems require considerably more system-level engineering than this. Some way of structuring the society is typically needed to reduce the system's complexity, to increase the system's efficiency, and to more accurately model the problem being tackled*".

En raison des problèmes rencontrés dans les ACMAS, les chercheurs ont pensé à augmenter ces systèmes par des mesures organisationnelles. Ceci a donné naissance à un second point de vue de l'ingénierie SMA qu'on appelle SMA centré sur l'organisation (OCMAS).

Approche centrée organisation (OCMAS)

Dans l'approche OCMAS la structure du système reçoit une plus grande attention à travers l'abstraction explicite de l'organisation des agents. Selon cette approche, le concepteur conçoit l'ensemble des schémas d'organisation et de coordination d'une part, et les comportements locaux des agents d'autre part. Cette approche est considérée comme une approche descendante car l'abstraction de l'organisation impose certaines règles ou normes utilisées par les agents pour coordonner leurs comportements locaux et leurs interactions avec d'autres agents.

Afin de définir l'organisation dans le contexte des SMA, Ferber (Ferber, 1995) a adopté la définition d'Edgar Morin (Morin, 1977) :

« L'organisation peut être définie comme un agencement de relations entre composants ou individus qui produit une unité, ou système, dotée de qualités inconnues au niveau des composants ou individus. »

De même, Wooldridge et al. (Wooldridge, & al., 2000) ont proposé une autre définition :

« We view an organisation as a collection of roles that stand in certain relationships to one another, and that take part in systematic institutionalised patterns of interactions with other roles. »

D'après Ferber (Ferber, 1995) (Ferber, & al., 2004), l'organisation dispose deux niveaux différents à savoir : le niveau structurel et le niveau concret.

- *L'organisation structurelle* : désigne les caractéristiques abstraites de structuration de l'organisation, afin que l'ensemble des membres soit cohérent (Ferber & Gutknecht, 1998).

- *L'organisation concrète* : représente une instance de l'organisation structurelle.

En effet, l'organisation d'un SMA permet de structurer les entités du système et permet de définir des schémas génériques d'interaction. Ainsi, l'organisation présente plusieurs avantages :

- Réduire la complexité de l'espace de recherche qu'un agent doit envisager pour atteindre son but.
- Limiter le nombre de conflits et donc augmenter l'efficacité globale de la société.
- Accroître la modularité et la réutilisation dans la conception des SMA.

En résumé, l'adoption des deux approches ACMAS ou OCMAS dépend principalement de la nature du domaine d'application et du degré de complexité du système. Cependant, l'approche OCMAS est plus adéquat pour l'ingénierie de systèmes multi-agents adaptatifs complexes, qui devraient être, dans un proche avenir, l'approche dominante pour l'ingénierie de domaines d'application à grande échelle et même à très grande échelle, en particulier avec le sujet en évolution de l'Internet of Things (IoT) (Mattern, & Floerkemeier, 2010), qui concerne des appareils capables de communiquer via Internet et de manipuler une énorme quantité de données.

5. Modèles organisationnels

Les modèles organisationnels ont été récemment utilisés dans la théorie des agents pour modéliser la coordination dans les systèmes ouverts et pour assurer l'ordre social dans les applications SMA (Argente, & al., 2007). L'adoption des modèles organisationnels revêt actuellement une grande importance dans la plupart des méthodologies d'ingénierie logicielle orientées agent.

Dans les environnements ouverts, les agents doivent être capables de s'adapter vers les organisations les plus appropriées en fonction des conditions de l'environnement et de leurs changements imprévisibles. En conséquence, les modèles organisationnels doivent garantir la capacité des organisations à se réorganiser dynamiquement en réponse aux changements des environnements dynamiques.

Le but premier d'un modèle organisationnel consiste à améliorer l'analyse et la conception des SMA Organisationnels (OCMAS), il est donc généralement intégré à une méthodologie d'ingénierie logicielle basée sur des agents.

5.1. Définition

Un modèle organisationnel est un Framework conceptuel ayant une syntaxe permettant de décrire une spécification organisationnelle d'un SMA (Coutinho, & al., 2009). Cette spécification permet une formalisation structurelle et fonctionnelle d'une organisation d'agents.

Un modèle organisationnel permet, en général, d'établir une définition (Coutinho, & al., 2009) : Structurelle, Fonctionnelle, Interactionnelle, Normative, Environnementale, Ontologique, Évaluative et Évolutive. Nous nous limitons dans ce qui suit à la définition des quatre premières dimensions fondamentales qui existent presque dans tous les modèles organisationnels. Les dimensions qui restent sont complémentaires et peu utilisées.

- *Structurelle* : représente l'aspect structurel et statique de l'organisation
- *Fonctionnelle* : décrit les objectifs de l'organisation, par la spécification des buts à réaliser par les agents de l'organisation.
- *Interactionnelle* : permet de déterminer les actions et les interactions attendues par les éléments de l'organisation afin de réaliser ses buts.
- *Normative* : représente les contraintes de l'organisation en termes de lois, normes et contraintes d'ordre social. Ces contraintes établissent une relation ternaire entre les trois dimensions structurel, interactionnel et fonctionnel de l'organisation.

5.2. Principaux modèles organisationnels

Plusieurs modèles organisationnels ont été proposés dans la littérature, tels que les modèles : AGR (Ferber, & al., 2004), MOISE+ (Hübner, & al., 2002), OMACS (Deloach, & al., 2008), OperA (Dignum, & al., 2004), OMNI (Dignum, & al., 2005). Ces modèles diffèrent dans leurs dimensions organisationnelles, la sémantique de leurs spécifications et le type du SMA modélisé. Dans ce qui suit nous présentons quelques modèles organisationnels les plus connus dans la littérature.

5.2.1. Modèle AGR (Agent, Group, Rôle)

AGR (Ferber & al., 2004) représente l'un des premiers modèles organisationnels proposés dans la littérature. Il s'agit d'un modèle OCMAS très concis et minimal appelé AGR, pour Agent/Group/Rôle, également connu sous le nom de modèle AALAADIN (Ferber, & Gutknecht, 1998). Les auteurs du modèle AGR ont proposé un ensemble de notations et un cadre méthodologique pour aider le concepteur à construire un SMA en utilisant AGR.

Leur modèle repose sur la création dynamique de groupes d'agents (partitionnement d'agents) et la constitution dynamique de hiérarchies de groupes (Holarchies). Ils ont souligné que leur modèle basé sur AGR peut être intégré à la méthodologie de développement Gaia (Wooldridge & al., 2000) pour compléter les phases d'analyse et de conception du développement MAS. La figure 1.2 présente le méta-modèle AGR.

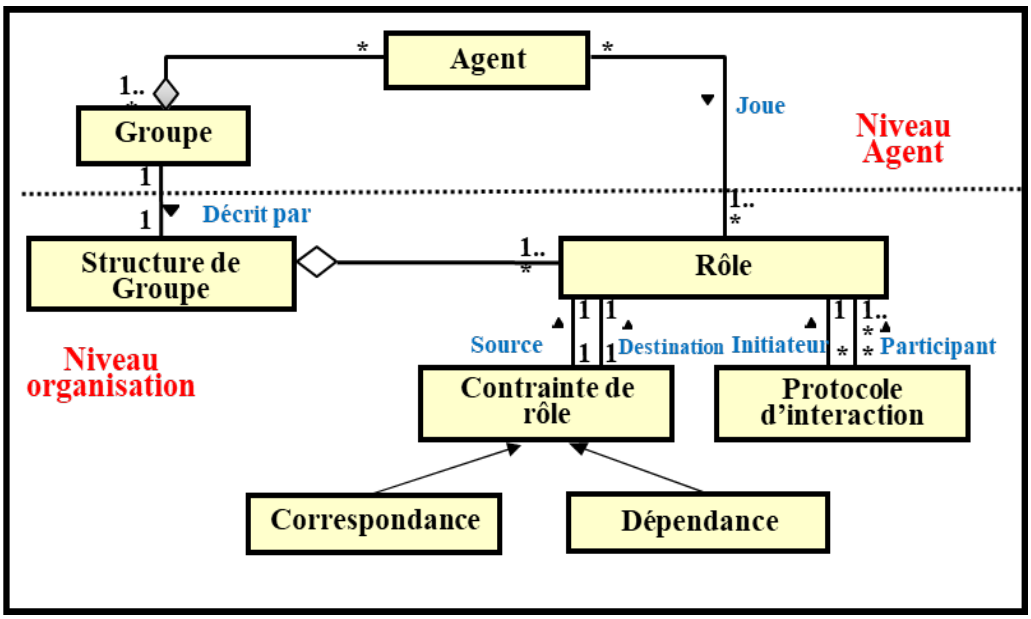


Figure 1.2. Le méta-modèle AGR (Ferber, & al., 2004)

Le modèle AGR est basé sur les trois concepts de base *Agent, Groupe et Rôle*.

- Agent : une entité active et communicante qui joue des rôles au sein des groupes, sans aucune restriction sur son architecture interne.
- Groupe : est défini comme l'unité de base de l'agrégation d'agents. Chaque agent fait partie d'un ou plusieurs de ces groupes.
- Rôle : est une abstraction d'une fonction, d'une position dans l'organisation ou d'un comportement attendu par l'agent qui va le jouer. Un rôle peut être décrit par : un attribut de cardinalité représentant le nombre maximal des agents pouvant jouer ce rôle et deux types de relation inter-rôle :
  (1) Correspondance : qui indique que si un agent joue le rôle A, il va jouer automatiquement le rôle ayant une relation de correspondance avec le rôle A.
  (2) Dépendance : qui définit l'ordre dans l'adoption des rôles

La structure de groupe est un concept abstrait important qui est également présenté dans le méta-modèle AGR illustré à la figure 1. La structure de groupe est une représentation abstraite des rôles requis dans ce groupe et de leurs relations et protocoles d'interaction.

Ferber et al. (Ferber & al., 2005) ont présenté une extension du modèle d'organisation AGR, appelé AGRE (AGR + Environnement), qui inclut des environnements physiques (ou simplement géométriques). Cette extension est basée sur le concept d'un espace qui peut

être vu soit comme un espace physique, soit comme un groupe social. Les principaux avantages des modèles AGR/AGRE sont : la prise en charge d'architectures d'agents hétérogènes, de langages de communication hétérogènes et de relations dynamiques entre les groupes de rôles.

La spécification AGR présente, plusieurs inconvénients. D'une part, elle offre la possibilité de représenter des liens d'interactions (protocole) entre les rôles, mais ne permet pas la spécification de la logique du protocole d'interaction. D'autre part, l'AGR n'offre qu'une vue structurelle minimale pour l'organisation des SMA, et ne supporte ni une spécification fonctionnelle ni une spécification interactionnelle (Coutinho & al., 2009). Cependant, d'autres extensions de AGR ont été proposés par la suite, tel que AGRE (Ferber, & al., 2005).

5.2.2. MOISE (Model of Organization for multI-agent SystEms)

MOISE (Modèle d'organisation pour les systèmes multi-agents) a été proposé par Hannoun et al. (Hannoun & al., 2000) afin de modéliser les aspects organisationnels du SMA. Semblable au modèle AGR, leur modèle est basé sur trois concepts majeurs : les rôles, les liens organisationnels (relations de rôles) et les groupes.

Le modèle MOISE se distingue par la possibilité d'intégrer les deux points de vue, ACMAS et OCMAS. Car d'une part, MOISE donne la possibilité au concepteur de modéliser totalement ou partiellement le comportement social des agents du système en précisant les structures organisationnelles possibles qui serviront pour la vérification et la validation du système. D'autre part, dans un souci de flexibilité, les agents devraient pouvoir raisonner sur leurs comportements sociaux et avoir une influence directe sur la réorganisation dynamique du système.

Le modèle MOISE est structuré selon trois niveaux :

- *Niveau individuel* : décrit pour chaque agent, les tâches dont il est responsable,
- *Niveau agrégat* : décrit l'agrégation des agents dans de grandes structures,
- *Niveau de la société* : concerne la structuration globale et l'interconnexion des agents et des structures.

L'organisation dans MOISE est vue comme un ensemble normatif de règles qui contraint les comportements des agents (Hannoun, & al., 2000). Le modèle organisationnel MOISE a été étendu par Hübner et al. à MOISE+ (Hübner, & al., 2002a) pour créer un modèle centré sur l'organisation. MOISE+ supporte la spécification explicite des trois dimensions structurelle, fonctionnelle et déontique (Hübner, & al., 2002) :

● *Spécification structurelle* : décrit les relations statiques qui existent entre agents, en termes de relations inter-rôles et de relations inter-groupes. Les relations inter-rôles sont représentées par les liens de Communication, Acquaintance, Autorité et de

Compatibilité entre les agents. Les relations inter-groupes sont définies selon un point de vue plus général, entre les rôles de deux groupes différents.

● *Spécification fonctionnelle* : décrit les objectifs globaux de l'organisation à l'aide d'un arbre de décomposition du but global en un ensemble de plans. Chaque plan est une exécution séquentielle, alternative ou parallèle d'un ensemble de buts élémentaires ou d'autres sous plans. Les buts élémentaires sont affectés aux agents sous forme des missions à réaliser.

● *Spécification déontique* : lie les deux spécifications structurelle et fonctionnelle par un ensemble de règles déontologiques (obligation, permissions et prohibition) sur les rôles et les missions qui sont définies dans la dimension fonctionnelle.

MOISEInst est une autre extension de MOISE+ (Hubner, & al., 2004) qui a été réalisée pour ajouter un processus de réorganisation dynamique afin d'adapter les changements d'environnement.

5.2.3. OMACS (Organization Model for Adaptive Computational Systems)

Le modèle OMACS a été proposé par Deloach et al. (Deloach, & al., 2008) pour la spécification de l'organisation dans les SMA adaptatifs. Il permet de définir des connaissances sur la structure organisationnelle du système et les capacités de ses agents. Ces connaissances seront utilisées par l'organisation, afin de s'adapter aux changements de son environnement ou des capacités des agents.

Le modèle OMACS définit une organisation composée d'objectifs, de rôles et d'agents ainsi que d'entités supplémentaires appelées capacités, affectations et politiques. Chaque rôle est défini pour atteindre un ou plusieurs buts de l'organisation. L'affectation d'un agent à un rôle est contrainte par la possession de l'agent aux capacités exigées par le rôle.

Afin de réaliser l'objectif global du système, une organisation OMACS doit trouver les meilleures affectations pour la réalisation de ses buts de manière efficace et optimale. Ces affectations définissent un ensemble de tuples agent-rôle-objectif (a, r, g) pour indiquer qu'un agent a est assigné pour jouer le rôle r afin d'atteindre l'objectif g. Enfin, les politiques précisent les réglementations du système telles que « un agent ne peut jouer qu'un seul rôle à la fois ». Le métamodèle OMACS de base est présenté sur la figure 1.3.

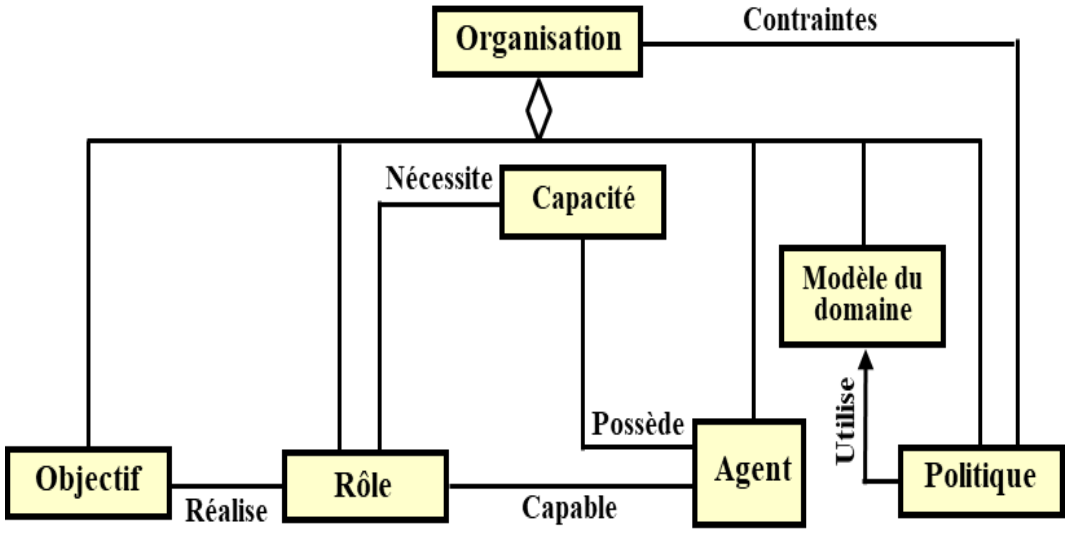


Figure 1.3. Modèle organisationnel pour les systèmes informatiques adaptatifs (Deloach, & al., 2008)

D'autres modèles organisationnels ont été proposés dans la littérature. Pour plus d'informations se refaire à l'évaluation des modèles organisationnels établie par Coutinho et al. (Coutinho, & al., 2009).

6. Adaptation dynamique des SMA

Les mécanismes d'organisation des SMA proposés auparavant abordaient les aspects organisationnels lors de la conception, ce qui nécessite des connaissances initiales à propos des objectifs exacts du système futur et les interactions auxquelles il peut être confronté à l'avenir doivent être connues lors de la conception du système (Bernon, & al., 2005). Néanmoins, l'ouverture, la complexité et l'hétérogénéité des logiciels récents imposent de nouvelles exigences au génie logiciel orienté agent (AOSE) (Jennings, 1999), et nécessitent des techniques avancées qui permettent le développement de SMA efficaces et adaptatifs.

Le concept d'adaptation dynamique fait référence aux modifications de la structure et du comportement d'un SMA pendant son fonctionnement sans l'arrêter (Valetto, & al., 2001). L'adaptation dynamique exige que les systèmes puissent évaluer leur propre état (c'est-à-dire leur succès et d'autres paramètres d'utilité) et prendre les mesures nécessaires pour le préserver ou le récupérer, en effectuant des actions d'intégration et de reconfiguration appropriées. La réorganisation des organisations doit donc décrire à la fois des situations dans lesquelles le comportement opérationnel de l'organisation change, en raison de l'admission ou du départ d'agents, ainsi que des situations dans lesquelles la structure de la société change.

Deux techniques d'adaptation ont été identifiés par Picard à savoir l'Auto-organisation et la Réorganisation (Picard, & al., 2009).

● **Auto-organisation**

« L'auto-organisation est un processus endogène ascendant concernant les systèmes dans lesquels seules des informations et représentations locales sont manipulées par les agents inconscients de l'état de l'organisation dans sa globalité, afin d'adapter le système à la pression environnementale en modifiant indirectement l'organisation, donc en changeant directement la configuration du système (topologie, voisinages, influences, différenciation), ou l'environnement du système, par des interactions et propagations locales, en évitant le biais de modèles prédéfinis ».

Cette définition indique que l'auto-organisation représente le point de vue de l'ACMAS. Dans un système auto-organisé, les agents ignorent le niveau d'organisation, le processus de réorganisation est décentralisé, implicite, endogène, et les agents sont responsables de la réorganisation dynamique du système, qui est souvent initiée par un changement environnemental.

● **Réorganisation**

« La réorganisation est un processus endogène ou exogène, concernant les systèmes dans lesquels l'organisation est explicitement manipulée au travers de spécifications, des contraintes ou autres moyens, afin d'assurer un comportement global adéquat, lorsque l'organisation n'est pas adaptée. Les agents étant conscients de l'organisation, ils sont capables de manipuler des primitives afin de modifier leur environnement social. Ce processus peut-être à la fois initié par une entité externe au système ou par les agents eux-mêmes, en raisonnant directement sur l'organisation (rôles, spécification organisationnelle) et sur les schémas de coopération (dépendances, engagements, pouvoirs) ».

Ce processus peut être à la fois initié par une entité externe ou par les agents eux-mêmes, en raisonnant directement sur l'organisation (rôles, spécification organisationnelle) et les modes de coopération (dépendances, engagements, pouvoirs). Cette définition suppose que les agents sont conscients de l'existence du niveau organisationnel, elle concerne donc le point de vue de l'OCMAS

Dans un système en réorganisation dynamique où les agents connaissent le niveau d'organisation, le processus de réorganisation peut être décentralisé ou non, mais toujours explicite et directement réalisé par des entités (concepteur ou agents) manipulant des primitives organisationnelles. Par conséquent, la sensibilisation est une dimension clé ajoutée par Picard pour identifier les SMA auto-organisés.

7. Types de réorganisation

Au début des travaux de réorganisation, la restructuration n'était possible que dans la phase d'initialisation du système. Plus tard, des approches ont permis d'adapter dynamiquement la structure du système (Hannebauer, 2002).

Les mises en œuvre actuelles de l'adaptation organisationnelle incluent des approches basées sur l'équilibrage de charge ou l'allocation dynamique des tâches. Ce qui est souvent le cas dans l'auto-conception organisationnelle des systèmes émergents qui, par exemple, incluent des primitives de composition et de décomposition permettant une variation dynamique de la structure organisationnelle tandis que la population du système reste la même (So & Durfee, 1993). Une autre approche courante est la participation dynamique, dans laquelle, l'interaction de l'agent avec l'organisation est modélisée par la mise en œuvre de certains rôles, et l'adaptation se produit lorsque l'agent entre et sort de ces rôles (Dignum, 2004), (Glasser & Morignot, 1997), (Tambe, 1997).

Sur la base des considérations ci-dessus, les situations de réorganisation suivantes ont été identifiées :

Changement de comportement : dans ce cas, la structure organisationnelle reste la même mais les agents qui jouent des rôles, décident (collectivement ou individuellement) d'utiliser des protocoles différents pour la même interaction abstraite décrite dans la structure. C'est le cas lorsque :

1. Un nouvel agent rejoint le MAS. Dans ce cas, un nouvel accord doit être conclu en spécifiant par exemple, les attentes et les obligations de la société vis-à-vis de la mise en œuvre des rôles, et éventuellement en intégrant quelques exigences propres de l'agent.

2. Un agent quitte le MAS. Dans ce cas, il est nécessaire de déterminer si le fonctionnement organisationnel est encore totalement ou partiellement possible.

3. Instanciation du modèle d'interaction. Dans ce cas, les agents exécutant un modèle d'interaction s'accordent sur un protocole spécifique qui est conforme à la spécification du modèle. Ces protocoles peuvent être différents, en fonction de l'évaluation actuelle de l'environnement par les agents et / ou de leurs propres objectifs et règles de comportement.

Changement structurel : dans ce cas, une décision est prise concernant la modification d'un ou plusieurs éléments structurels.

1. Auto-conception organisationnelle : c'est-à-dire variation dynamique dans les sociétés émergentes, résultant de changements dans l'interaction entre agents.

2. Adaptation structurelle : Dans ce cas, les sociétés conçues sont adaptées aux changements de l'environnement en ajoutant, supprimant ou modifiant ses éléments structurels (par exemple, rôles, dépendances, normes, ontologies, primitives de communication).

Intuitivement, les changements de comportement ont un caractère plus temporaire et n'influencent pas l'activité future de l'organisation, alors que le changement structurel est censé s'adapter à une modification permanente et, en tant que tel, diriger l'activité des

futures instanciations de l'organisation. Dans nos travaux de recherches sur la maintenance des SMA, nous nous intéressons principalement au changement du comportement.

8. Processus de réorganisation

La réorganisation dans les sociétés d'agents peut être définie comme un processus qui change la société (Hübner, & al., 2004). Cette réorganisation fait référence à des modifications de la structure et du comportement de la société d'agents, telles que l'ajout, la suppression ou la substitution de composants, qui sont effectuées pendant le fonctionnement du système et sans l'arrêter (Dignum, & al., 2004). Ces modifications sont liées à la spécification de l'organisation, c'est-à-dire aux rôles, aux objectifs, aux services, aux normes et à la population d'agents, ainsi qu'aux changements dans les relations entre ces agents.

À cet égard, le cycle de vie d'une réorganisation peut être défini comme le processus d'analyse des problèmes de la société d'agents actuelle, de proposition de solutions d'adaptation, de sélection et de mise en œuvre d'une réorganisation, et d'évaluation de ce processus une fois appliqué. Les auteurs (Weyns, & al., 2010), (Kephart & Chess, 2003) s'accordent à dire que la réorganisation et plus précisément, la réorganisation dans les sociétés d'agents (So & Durfee. 1993), (Hubner, & al., 2004), peut être représentée comme un processus en boucle composé de différentes phases. La définition spécifique de chaque phase peut légèrement changer d'un auteur à l'autre.

Picard et al. (Picard, & al., 2009) ont identifié un processus générique qui sera implémenté différemment en fonction de l'approche suivie. Ce processus est généralement composé de deux phases à savoir le monitoring et la réparation. Selon le type du SMA, la phase de réparation, peut être constituée des phases de conception, de sélection et d'exécution ou uniquement des phases de sélection et d'exécution.

Dans ce qui suit, nous présentons les phases principales du processus de réorganisation à savoir la surveillance, conception, sélection et l'évaluation.

8.1. Surveillance (monitoring)

La phase de surveillance définit *pourquoi* et *quand* une organisation doit être réorganisée. La surveillance est essentielle pour pouvoir détecter les comportements indésirables qui doivent être corrigés (Guessoum, & al., 2004), et qui peuvent être déclenchés par des changements dans l'environnement. D'après Dignum et al. (Dignum & al., 2004), les changements d'environnement sont les déclencheurs de la réorganisation. Ainsi, les approches de réorganisation peuvent être classées selon qu'elles sont *réactives* ou *proactives*.

- *Une réorganisation réactive* se produit lorsque l'organisation répond automatiquement aux événements qui provoquent une réorganisation tels que

l'ajout ou la suppression d'un nouveau rôle, agent, etc. Ces événements amènent l'organisation à faire les ajustements nécessaires afin de continuer à atteindre ses objectifs. Les approches qui suivent ce type de réorganisation se basent sur un processus de réorganisation guidé par les événements.

- *Une réorganisation proactive* nécessite un raisonnement implicite de la situation actuelle afin de décider de la nécessité d'une réorganisation. Une réorganisation *proactive* peut également impliquer des situations dans lesquelles un événement se produit, nécessitant une réorganisation à effectuer après un processus de raisonnement. Les approches qui suivent ce type de réorganisation ont deux perspectives différentes. Certaines approches suivent une réorganisation proactive lorsqu'un événement empêche l'organisation à atteindre ses objectifs. D'autres approches appliquent une réorganisation proactive pour augmenter l'utilité de l'organisation.

La phase de surveillance peut être menée de manière :

- Centralisée si un agent ou une autorité spécifique est chargé de décider de la nécessité d'une réorganisation.
- Décentralisée (distribuée) lorsqu'un ensemble d'agents peuvent décider soit de manière autonome, soit à travers un accord de la nécessité de la réorganisation.

8.2. Conception

La phase de conception définit la problématique de la conduite d'une réorganisation. Une fois qu'un processus de réorganisation est requis, la phase de conception comprend une analyse des éléments organisationnels et une proposition de réorganisation qui modifie des éléments spécifiques de la société d'agents. À l'instar de la phase de surveillance, la conception peut également être réalisée de manière centralisée si un seul agent ou une autorité centrale est chargé de proposer la solution de réorganisation. Une conception distribuée implique la participation de plusieurs agents à la proposition de solution de réorganisation.

Selon le modèle spécifique, les approches de réorganisation actuelles fournissent un support pour changer différents éléments de la société d'agents en fonction des exigences des problèmes qu'ils considèrent.

Certains auteurs (Dignum, & al., 2004) proposent une classification des types de réorganisation en adaptation comportementale et structurelle. Néanmoins, une classification plus détaillée peut être fournie en incluant des types courants de changements qui peuvent être trouvés dans la littérature selon les dimensions suivantes :

- **Système ouvert** permet de modifier la population d'agents, c'est-à-dire que les agents peuvent entrer ou sortir du système.
- **L'émergence** permet de modifier les éléments qui définissent le comportement social ; c'est-à-dire l'ajout ou la suppression des rôles que les agents peuvent jouer, des objectifs sociaux, etc.
- **L'adaptation comportementale** permet des changements liés au comportement des agents qui peuplent la société. Par exemple, il s'agit de modifier les capacités offertes par un agent afin de pouvoir jouer un rôle.
- **L'adaptation fonctionnelle** permet de modifier la manière dont les différents éléments de la société d'agents sont liés les uns aux autres, ce qui affecte la fonctionnalité de la société, tels que des changements dans les services offerts par un rôle ou des changements dans les rôles joués par les agents.
- **L'adaptation structurelle** permet des changements dans les relations entre les éléments de la société d'agents, ce qui affecte la structure sociale, comme les relations entre les agents.
- **L'adaptation des normes** permet des changements dans les régulations de la société d'agents. Cette adaptation peut être liée à des modifications dans la spécification des normes qui régissent la société d'agents ainsi qu'à l'ajout ou la suppression de nouvelles normes.

Selon les éléments qui peuvent être modifiés dans un processus de réorganisation, un large ensemble de solutions différentes peut être fourni. Il serait donc souhaitable qu'une réorganisation puisse prendre en compte le plus de dimensions possibles. Certaines approches de réorganisation se concentrent sur les changements dans une dimension spécifique, tandis que d'autres approches sont plus flexibles en considérant les changements dans plusieurs dimensions.

8.3. Sélection

La phase de sélection pose la problématique du choix de la réorganisation finalement mise en œuvre. Si plusieurs réorganisations ont été proposées en phase de conception, la phase de sélection détermine laquelle de ces propositions est appliquée. Selon qu'une seule réorganisation est conçue ou que plusieurs conceptions sont proposées, plusieurs critères peuvent être utilisés en phase de conception pour guider la conception, ou en phase de sélection pour guider la sélection. Dans certains scénarios, la réorganisation peut être considérée comme le mécanisme qui permet à la société d'atteindre les objectifs sociaux. Dans d'autres scénarios, selon Dignum et Sonenberg (Dignum, & al., 2004), une réorganisation est souhaitable si elle conduit à accroître l'utilité du système. Cette utilité doit prendre en compte à la fois le succès de la réorganisation et le coût de tout changement nécessaire pour réaliser la réorganisation à partir de la situation actuelle (Glaser, & al., 1997), (Alberola, & al., 2011).

Comme indiqué dans (Kotter & Schlesinger, 1979), la plupart des changements organisationnels peuvent rencontrer des problèmes lorsqu'ils prennent plus de temps que prévu. Dans ce cas le coût du temps de gestion peut être augmenté et une résistance peut avoir lieu de la part des personnes impliquées dans le changement. Afin de traiter ce problème, les avantages obtenus par la réorganisation et les coûts associés à ce processus sont considérés comme aspects importants qui doivent être pris en compte afin de définir la pertinence d'un processus de réorganisation. Les principaux problèmes de cette phase consistent donc à la définition des critères de choix parmi les solutions possibles, et la minimisation du coût d'implémentation de la nouvelle organisation.

8.4. Évaluation

La phase d'évaluation définit le problème de l'analyse de la qualité d'une réorganisation après son déploiement. Cette phase fournit un feedback (retour d'information) sur la réorganisation afin d'évaluer si elle s'est déroulée comme prévu ou pas. Cela permet d'évaluer la qualité de la réorganisation qui a été sélectionnée ainsi que la qualité de la société qui a été réalisée, afin d'en tenir compte pour les réorganisations futures (Luck, & al., 2005).

Des travaux liés aux systèmes adaptatifs tels que ceux de Cheng et al. (Cheng, & al., 2009) définissent différents facteurs qui mesurent la criticité de la réorganisation, la prévisibilité, les frais généraux qui lui sont associés, et si le système est résilient face au changement.

Cette phase est souvent négligée dans la plupart des travaux de réorganisation et sera également hors de nos préoccupations dans ce premier travail vers la maintenance des SMA organisationnels que nous présenterons au chapitre 4.

9. Approches pour la réorganisation des SMA

Plusieurs approches ont été proposées dans la littérature pour la réorganisation des SMA telles que : OMACS (Deloach & Matson, 2004), MOISE+(Hubner, & al., 2004), MACODO (Weyns, & al., 2010), TSE (Kota, & al., 2008), (Kota, & al., 2012), AON (Gaston & al., 2005), MAGIQUE (Mathieu, & al., 2002), l'AEI (Bou, & al., 2006), 2-LAMA (Campos, & al., 2011), etc. Dans ce qui suit, nous présentons les principales caractéristiques des approches les plus connues dans la réorganisation des SMA, en mettant l'accent sur les différentes phases du processus de réorganisation

La plupart des approches de réorganisation ne se concentrent que sur des phases spécifiques ou des changements dans des dimensions spécifiques. À titre d'exemple, des travaux tel que (Seelam, 2009) se concentrent sur la sélection de la meilleure réaffectation des rôles, tandis que d'autres travaux tel que (Kamboj and Decker, 2006) se concentrent sur la sélection de la meilleure réorganisation structurelle. D'autres travaux portent notamment sur la phase de surveillance (Guessoum, & al., 2004) ou sur l'apprentissage

d'algorithmes pour optimiser les interactions des agents (Abdallah and Lesser, 2007). D'autres travaux par contre nécessitent une interaction humaine élevée afin de choisir la décision de réorganisation (Carvalho, & al., 2006).

La phase de monitoring a été prise en compte dans presque toutes les approches. Peu de travaux envisagent une réorganisation associée à des événements tel que dans OMACS. Cependant, la plupart des approches se focalisent sur la réorganisation afin de maximiser le processus d'organisation. Seules les approches proposées par Weyns et al. (Weyns, & al., 2010) et par Tinnemeier et al. (Tinnemeier, & al., 2010) proposent une approche de réorganisation qui se concentre sur le domaine, c'est-à-dire une réorganisation causée par des changements dans l'environnement.

La phase de conception est la plus étudiée dans les démarches de réorganisation actuelles. Chaque approche analysée fournit des mécanismes pour proposer une solution de réorganisation. Cependant, les changements des éléments de l'organisation sont couverts séparément par les approches actuelles (Alberola, & al., 2011). Certains des travaux actuels se concentrent sur des problèmes qui abordent la réorganisation dans sa dimension fonctionnelle comme l'approche OMACS, qui modifie l'affectation des agents aux rôles. D'autres approches traitent des problèmes qui nécessitent des changements structurels comme dans TSE, AON ou MAGIQUE. D'autres approches comme AEI ou 2-LAMA sont spécialisées dans l'évolution de la réglementation du système.

La phase de sélection est également assurée par les approches actuelles. Seules quelques approches considèrent une sélection distribuée de la réorganisation (Alberola, & al., 2011). Les travaux de Hubner et al. (Hubner, & al., 2004) et ceux de Horling et al. (Horling, & al., 1999) proposent des approches dans lesquelles plusieurs agents sont impliqués dans la sélection de la réorganisation. Ainsi, les approches actuelles sont principalement axées sur la proposition d'une solution de réorganisation unique, qui est finalement retenue. Le critère qui a principalement été pris en compte dans la phase de sélection est la solution qui maximise l'utilité de l'organisation. Cependant, cette utilité ne prend en compte que les bénéfices apportés par le processus de réorganisation et non les coûts pour y parvenir. Seuls Nair et al. (Nair, & al., 2003) utilisent une politique de prise de récompense pour évaluer l'utilité du processus de réorganisation en prenant en compte les avantages d'un processus de réaffectation des rôles et le coût qui lui est associé. L'approche MOISE+ utilise un coût global associé au nombre de rôles et de missions supprimés pour sélectionner une solution.

Enfin, très peu d'efforts ont été faits pour étudier la phase d'évaluation. Par conséquent, il est supposé que la réorganisation qui est conçue et sélectionnée va fonctionner correctement. Seule l'approche MOISE+ mesure le niveau de réussite de la réorganisation et l'intègre dans les réorganisations futures au moyen d'une technique de Q-learning.

Discussion

À partir de l'étude que nous avons menée, nous avons constaté que les approches de réorganisation présentent plusieurs lacunes notamment en phase de monitoring qui constitue une étape fondamentale pour l'adaptation du système en général et sa maintenance en particulier.

Certaines approches mettent en œuvre des stratégies de surveillance sous forme de règles prédéfinies (OMACS, MACODO), tandis que d'autres approches, se base sur l'évaluation de la performance du système (2-LAMA). Les règles qui régissent l'exigence de réorganisation sont donc prédéfinies à la conception et ne peuvent pas être modifiées à l'exécution du système. Ceci est complètement différent dans la réalité où les règles de réorganisation ne sont pas toutes connues à l'avance et sont même en évolution tout au long de la durée de vie de l'organisation.

Comme indiqué dans (Abdu, & al., 1999), les systèmes complexes et ouverts peuvent également entraîner une modification des exigences de surveillance. Afin de s'adapter aux nouvelles exigences, un accompagnement dynamique offrirait plus de flexibilité aux systèmes dynamiques, en particulier dans les scénarios où il est difficile de spécifier la logique de réorganisation à la conception.

De plus, les informations utiles devant être surveillées sont généralement spécifiées hors ligne, avant l'exécution du système. Cependant, à mesure que le nombre d'agents dans la société et leur complexité augmentent, beaucoup plus d'informations sont échangées entre les agents et la plupart de ces informations pourraient ne pas être utiles à l'exécution. Le système de monitoring devrait donc tenir compte des changements de ces informations pertinentes pour le bon fonctionnement du système.

De ce qui précède, nous constatons qu'une approche adaptative devrait s'appliquer non seulement au comportement et à la structure du système en exécution, mais également à la conception du système de surveillance, en particulier lorsqu'il s'agit de gérer des systèmes complexes sur de longues périodes. Il serait donc intéressant pour la prochaine génération de sociétés d'agents d'avoir un support permettant la spécification dynamique des informations et des règles qui déclenchent la réorganisation.

Au vu des lacunes ressenties dans les différentes approches étudiées, nous proposons dans le cadre de cette thèse d'utiliser un système de monitoring se basant sur la Programmation Orientée Aspects (PAO). Nous pensons que l'utilisation des aspects dans les systèmes de monitoring pourrait apporter plusieurs réponses à ces préoccupations et permettra des améliorations considérables en termes de flexibilité pour l'adaptation des SMA. Un outil de monitoring basé sur la POA pour l'analyse et l'étude des SMA sera bien utilisé dans notre système de maintenance dans le prochain chapitre 3.

10. Synthèse

A partir de l'étude que nous avons menée dans ce chapitre, nous soulignons les points clés suivants :

• les SMA disposent de caractéristiques essentielles, qui font leurs différences par rapport aux autres systèmes tels que l'autonomie et la sociabilité des agents du système.

• OCMAS est une approche prometteuse pour l'ingénierie de systèmes multi-agents adaptatifs complexes.

• la réorganisation est une faculté très importante dans l'adaptation des systèmes multi-agents.

• la réorganisation d'un SMA peut prendre différentes formes :

- Changement de structure, de relations, de localisation des programmes ou de localisation des données ;
- Changement de contenu, de connaissances, ou de compétences ;
- Changement de normes ;

• la réorganisation d'un SMA peut se faire par différents processus :

- Réaction par une réponse prédéterminée à un évènement particulier ;
- Raisonnement ;
- Apprentissage des changements au cours de la vie du système.

• la réorganisation d'un SMA peut être réalisée :

- Au niveau de l'agent en le dotant d'attitudes sociales, de coopération, d'évolution de connaissances et de compétences ;
- Au niveau de l'organisation en incluant un processus d'évolution vers une organisation acceptable.

De ce qui précède, nous pouvons faire sortir deux questions fondamentales auxquelles il faudra trouver une réponse quand on cherche à doter un système multi-agent d'un mécanisme de maintenance (système de maintenance) :

• Quels sont les indices à collecter pour que le système puisse reconnaître des situations de perturbations et s'adapter en conséquence ?

• Quels sont les mécanismes à mettre en place pour que le système s'adapte ?

La réponse à ces questions sera abordée à travers nos contributions dans les deux chapitre 3 et 4, qui concernent la maintenance des systèmes multi-agents.

11. Conclusion

L'importance croissante du domaine des SMA comme étant un domaine de recherche fructueux vient du fait que ceux-ci offrent des solutions extensibles et évolutives à la fois aux problèmes fréquents.

Dans ce chapitre, nous avons mis l'accent sur les concepts les plus pertinents à notre problématique à savoir les caractéristiques des SMA et la réorganisation dynamique de ces systèmes. Nous avons pu à travers cette étude dégager le bilan de constats suivants :

- Les SMA possèdent des caractéristiques fonctionnelles propres qui font leurs différences avec les systèmes informatiques classiques et qui rendent leur analyse plus ardue.

- la réorganisation dynamique est une tâche très importante dans l'adaptation des systèmes multi-agents organisationnels ;

Dans le chapitre suivant, nous allons mettre au clair les concepts clé liés à la qualité et à la maintenance des systèmes en général et aux SMA en particulier afin de bien situer notre problématique.

Chapitre 2

Qualité et Maintenance des logiciels

Sommaire

1. Introduction

La maintenance du logiciel dépend des exigences du client ou de l'utilisateur. Ces exigences se traduisent par des changements mineurs, mais il est plus coûteux à mettre en œuvre qu'à anticiper. Dans de telles circonstances, une étude des performances du logiciel pourrait fournir des informations utiles, surtout lorsque le système est complexe et sophistiqué. Certaines caractéristiques fonctionnelles des applications informatiques doivent être prises en compte dans la maintenance du logiciel. Une meilleure gestion de ces fonctionnalités peut être obtenue en implémentant une bonne qualité du logiciel en évaluant les indicateurs de performances du logiciel (Tchoffa, & El Mhamedi, 2012). Dans ce travail nous nous concentrons sur la façon d'assurer la qualité du logiciel pendant son exploitation afin d'améliorer les procédures de maintenance du logiciel.

Ce chapitre sera divisé en deux grandes parties. La première partie sera consacrée à une étude de la qualité logicielle. La seconde partie du chapitre définit le problème de la maintenance et rappelle toutes les formes de maintenance qui existent et montre l'intérêt qu'il est nécessaire de porter à la maintenance préventive. Nous montrons l'intérêt d'une stratégie de maintenance préventive qui permet de réduire les coûts de panne et d'indisponibilité du système. Pour optimiser la maintenance préventive, il faut surveiller et analyser l'état du système à l'aide de méthodes qui s'appuient sur une connaissance plus ou moins approfondie du système à considérer (données historiques, données mesurées en ligne, modèles).

2. Qualité du logiciel

Cette section sera consacrée aux notions fondamentales liées à l'évaluation de la qualité logicielle.

2.1. Notion de qualité

La notion de qualité est complexe et ambiguë, selon que l'on soit du point de vue du client, du fournisseur ou de l'entreprise. Selon (Kitchenham & Pfleeger, 1996) la qualité :

«is a complex concept. Because it means different things to different people, it is highly context-dependent. Just as there is no one automobile to satisfy everyone's needs, so to there is no universal definition of quality».

Jones (Jones, 2008) montre qu'une définition claire et correcte de la qualité du logiciel est nécessaire et reste une réelle préoccupation pour la communauté et souligne que les définitions discutées jusqu'à présent sont données dans un sens étroit. Cependant, (Kitchenham & Pfleeger, 1996) a expliqué la diversité d'opinion par le fait que « la perspective que nous adoptons sur la qualité influence la façon dont nous la définissons ».

Afin de minimiser l'ambiguïté entourant la définition de la qualité Pfleeger et al. (Pfleeger, & al., 2005) ont divisé cette notion selon cinq points de vue :

- Vue transcendantale : la qualité est quelque chose que l'on peut reconnaître sans pouvoir la définir.

- Vue utilisateur : la qualité est la force apparente du produit à remplir des fonctions.

- Vue fabrication : la qualité est le respect du cahier des charges.

- Vue produit : la qualité est liée aux caractéristiques intrinsèques du produit, vue la plus citée par les experts de la qualité logicielle.

- Vue client : La qualité dépend du prix que le client est prêt à payer.

Par conséquent, la qualité logicielle doit avoir une définition qui permette de la reconnaître et de l'évaluer. Une approche globale de tous les aspects de la qualité est fournie par la norme ISO/IEC 9126-1, qui définit la qualité par : « L'ensemble des caractéristiques d'une entité qui portent sur sa capacité à satisfaire des besoins exprimés et implicites » (ISO/IEC 9126-1, 2001).

Toute exigence de qualité d'un logiciel nommée aussi propriété non-fonctionnelle, doit être formulée dans un document de spécification du système logiciel, en utilisant un modèle de qualité.

2.2. Modèle de qualité

Le standard ISO 9126-1 (ISO/IEC 9126-1, 2001) définit un modèle de qualité comme étant :

« The set of characteristics and the relationships between them, which provide the basis for specifying quality requirements and evaluating quality ».

Un modèle de qualité relie la qualité externe d'un logiciel à sa qualité interne. Dans la pratique, les modèles de qualité combinent les valeurs des métriques d'une manière bien définie, afin de faciliter l'analyse de la qualité.

Plusieurs modèles de qualité ont été proposés dans la littérature, tels que le modèle QMOOD (Quality Model for Object-Oriented Design) (Bansiya & Davis, 2002), le modèle GQM (Goal Question Metrics) (Basili, & al., 1994), le modèle FCM (Facteurs Critères Métriques) (McCall & al., 1977) et le standard ISO 9126 qui a été remplacé par le standard SQuaRE (Software product Quality Requirements and Evaluation) défini par ISO/IEC 25010 en 2011 (ISO/IEC 25010, 2011). Ces modèles comportent trois niveaux. Au premier niveau, on distingue les facteurs de qualité, qui sont décomposés en critères. Ces critères peuvent à leurs tours être décomposés en propriétés mesurables :

- **Les facteurs de qualité** appelés aussi caractéristiques représentent les caractéristiques qualité de haut niveau. Chaque facteur de qualité représente un aspect de qualité qui n'est pas directement mesurable.
- **Les critères de qualité** appelés aussi sous-caractéristiques qui affinent les caractéristiques.
- **Les mesures de qualité** appelées aussi attributs peuvent être évalués quantitativement. Un critère ou une sous-caractéristique va être ainsi apprécié quantitativement à l'aide d'une ou plusieurs mesures. Choisir les mesures de qualité appropriées aux (sous) caractéristiques est essentiel pour avoir un modèle « de qualité ». Les mesures peuvent être soit une métrique calculée à l'aide d'une formule soit un avis d'un expert. Cependant, il est à noter qu'une métrique est un type particulier de mesure.

Le modèle qualité de la norme, ISO/IEC 9126 (ISO/IEC 9126, 2003) classe la qualité logicielle dans un ensemble structuré de caractéristiques et de sous-caractéristiques comme suit. Dans le standard ISO/IEC 9126 les facteurs de qualité sont : la capacité fonctionnelle, la fiabilité, la facilité d'utilisation, l'efficacité, la maintenabilité et la portabilité. Ces six caractéristiques sont affinées à leurs tours en 27 sous-caractéristiques. Chaque sous-caractéristique de qualité est ensuite divisée en attributs. Un attribut est une entité qui peut être vérifiée ou mesurée dans le produit logiciel. Les attributs ne sont pas définis dans la norme, car ils varient entre les différents produits logiciels (Figure. 2.1). La norme fournit aux organisations un cadre pour définir un modèle de qualité pour tout produit logiciel et laisse à chaque organisation le soin de définir précisément son propre modèle.

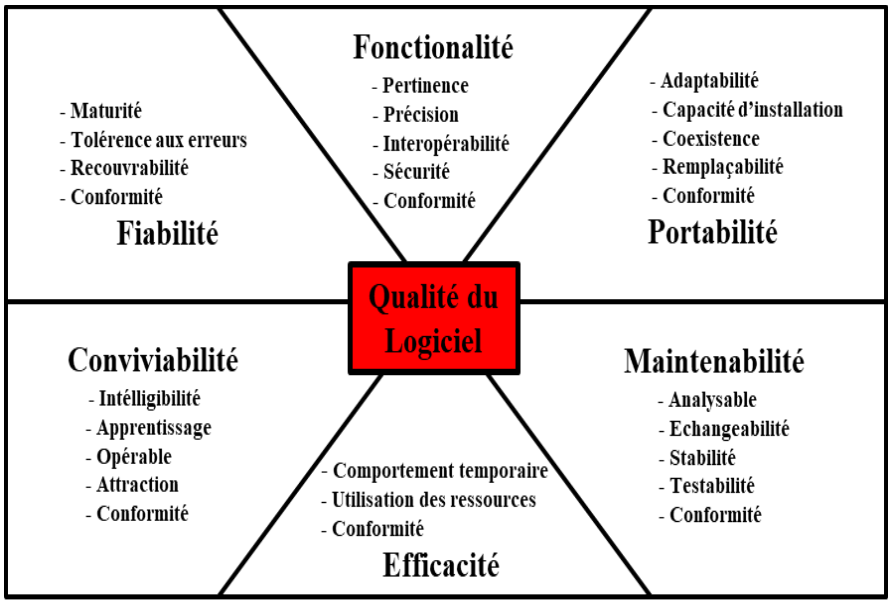


Figure 2.1. Décomposition des caractéristiques de qualité du logiciel en sous-attributs (Tchoffa, & El Mhamedi, 2012)

Les modèles doivent être adaptés aux différents paradigmes logiciels. Cependant, le paradigme agent n'a pas connu des travaux profonds et 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 (Marir, 2015).

Alonso et al. (Alonso, & al., 2008) ont remarqué l'absence d'un modèle global pour l'évaluation de la qualité des logiciels basés sur le paradigme agent. Ils ont donc proposé une série de travaux (Alonso, & al., 2008) (Alonso, & al., 2009) (Alonso, & al., 2010a) afin de développer un modèle dédié à l'évaluation des SMA. Dans un premier temps, Alonso et al. (Alonso, & al., 2008) ont proposé un modèle de qualité selon la définition du standard ISO/IEC 9126-1 (ISO/IEC 9126, 2001) et les caractéristiques de l'agent ont été spécifiées selon la littérature spécialisée. Dans ce modèle, un agent dispose des caractéristiques de (Alonso, & al., 2008) : sociabilité, autonomie, proactivité, réactivité, adaptabilité, intelligence et mobilité. Chaque caractéristique est divisée en un ensemble de sous-caractéristiques qui sont à leurs tours divisées en un ensemble de mesures.

Dans le même cadre, Alonso et al. (Alonso, & al., 2009) (Alonso, & al., 2010a) ont proposé un modèle avec des attributs et des métriques pour l'évaluation de l'autonomie et la proactivité. Ce modèle n'a pas atteint sa version complète à cause de l'absence des études consacrées aux autres caractéristiques de l'agent. Ces auteurs ont proposé par la suite, une autre version de mesures de la sociabilité et de l'autonomie sans prendre en compte les fonctions typiques (Alonso, & al., 2010b).

Afin d'évaluer les systèmes multi-agents, Bitonto et al. (Bitonto, & al., 2012) ont proposé un modèle basé sur le paradigme GQM (Goal-Question-Metric). Ce paradigme spécifie un modèle à trois niveaux : les buts, les questions et les métriques. Dans le premier niveau, les buts représentent les attributs spécifiant la qualité d'un logiciel. Au second niveau, des questions ont été utilisées pour spécifier chaque but. Le troisième niveau permet l'évaluation quantitative de ces buts. Ce modèle présente plusieurs avantages. Tout d'abord, il s'agit d'un modèle générique qui est indépendant de l'implémentation et du contexte de l'utilisation du SMA. De 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 posé. Comme il peut être utilisé par les évaluateurs pour choisir le système le plus adéquat afin de résoudre un problème donné.

Kaddoum et al. (Kaddoum, & al., 2009) se sont intéressés à l'évaluation des systèmes multi-agents adaptatifs. Cette évaluation concerne trois axes : le fonctionnement du système en cours d'exécution, les caractéristiques intrinsèques du système et la méthodologie du développement. Dans cette évaluation, les deux premiers axes concernent la qualité du produit logiciel, tandis que le troisième axe représente la qualité du processus de développement. Chacun des deux premiers axes auxquels nous nous intéressons dans ce travail, est divisé en plusieurs critères. Le fonctionnement du système peut être évalué par les deux critères de performance et d'homéostasie et robustesse.

Tandis que, les caractéristiques intrinsèques du système sont définies par les deux critères de complexité algorithmique et de décentralisation et résolution locale. Chacun des critères peut être mesuré par la suite à l'aide d'une ou de plusieurs métriques. Ce travail a certes essayé de donner une vue globale de la qualité logicielle, en intégrant la qualité du produit et la qualité du processus, mais ce point de vue reste toujours trop simpliste. En effet, ce modèle ignore la nature multi-facette de la qualité en admettant que la qualité se résume uniquement à quatre critères. En plus, le modèle proposé ne donne aucune indication sur les caractéristiques des SMA.

A partir du survol des différents modèles de qualité des SMA, nous avons constaté 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. De plus, la proposition des modèles de qualité pour les SMA ne représente qu'une partie mineure dans le domaine de l'évaluation de la qualité du logiciel. La plupart des travaux effectués dans ce domaine proposent donc des métriques pour l'évaluation des différents critères des SMA.

2.3. Métrique de qualité

Une métrique de qualité est destinée à mesurer quantitativement un aspect de la qualité d'une unité logicielle. Elle peut servir à sélectionner le composant adéquat en comparant entre deux produits différents, estimer les coûts, prédire les défauts futurs, etc.

Selon Schneidewind (Schneidewind, 1992), une métrique est une unité de mesure qui peut être utilisée comme un substitut de facteur de qualité. De même, Michael K.Daskalantonakis (Daskalantonakis, 1992) a défini une métrique comme étant une mesure quantitative pour déterminer un attribut de qualité: « A software metricis defined as a method of quantitatively determining the extent to which a software process, product, or project possess a certain attribute ».

Formellement, une métrique de qualité est une fonction dont le domaine de définition est tout ou partie d'un logiciel (qui peut être un modèle quelconque de spécification ou de conception, un listing de code source, un programme en cours d'exécution, etc.) et qui fournit en retour une ou plusieurs valeurs appartenant à un ensemble image souvent totalement ordonné (Hamza, 2014). Le domaine image de cette fonction est le plus souvent (Basili & Perricone, 1984) : un intervalle continu, un espace nominal (ensemble d'éléments appelés « catégories ») ou espace ordinal (des catégories ayant une relation d'ordre).

En général, nous distinguons deux types de métriques : les métriques statiques et les métriques dynamiques. Les métriques statiques sont calculées à partir du code source du logiciel (Chhabra, & Gupta, 2010). En revanche, nous devons exécuter le logiciel afin de mesurer les métriques dynamiques (Chhabra, & Gupta, 2010).

Les métriques statiques sont principalement appliquées en raison de leur relative simplicité par rapport aux dynamiques (Chhabra, & Gupta, 2010) (Dufour, & al., 2003) (Tahir, & al., 2010). Cependant, les métriques dynamiques fournissent des valeurs plus précises (Chhabra, & Gupta, 2010). Plusieurs paradigmes de programmation ont vu le jour depuis l'apparition du génie logiciel. Évidemment, les spécificités de chaque paradigme de programmation doivent être prises en compte, lors de la proposition des métriques logicielles. Par exemple, des métriques spécifiques sont proposées pour les logiciels orientés objet (Chidamber, & Kemerer, 1994), la programmation orientée aspect (AOP) (Zakaria, & Hosny, 2003) et les systèmes multi-agents (MAS) (Sivakumar, & Vivekanandan, 2012) (Mahar, & Bhatia, 2014).

Dans les SMA, on peut mesurer certains attributs spécifiques comme l'autonomie, la réactivité et la flexibilité des agents. Par exemple, dans certains contextes, nous devons développer des agents autonomes de haut niveau (tels que des robots pour explorer l'espace). Dans des contextes différents, les agents doivent être développés avec une autonomie limitée (par exemple, agent de contrôle de missiles très destructeurs). Ainsi, la mesure des attributs spécifiques aux SMA est très intéressante, car elle peut être utilisée à plusieurs fins, tel que le contrôle de la qualité des systèmes développés, le choix des agents les plus adéquats pour un projet donné ou l'évaluation des systèmes afin de les maintenir. Malgré l'importance des métriques statiques pour mesurer certains attributs du SMA, les métriques dynamiques sont plus appropriées pour les systèmes multi-agents en raison de leur nature évolutive imprévisible (Marir et al., 2016). Par exemple, un agent se caractérise par la flexibilité de ses comportements (Wooldridge, 2009), où il peut modifier son comportement en fonction de sa situation actuelle. Ainsi, le code d'un système multi-agents ne peut donner suffisamment d'informations sur son exécution future (lorsqu'un agent atteint son objectif par exemple) et ne peut être utilisé seul pour mesurer certains attributs des systèmes multi-agents. Donc, il est indispensable dans ce cas de mesurer les attributs souhaités lors de l'exécution du système. Mais, malgré l'importance de la méthode de mesure, elle a été ignorée pour les métriques les plus spécifiques proposées pour les systèmes multi-agents.

En raison de la dynamicité des SMA, nous nous intéressons dans le cadre de cette thèse, uniquement aux métriques dynamiques des Systèmes Multi-Agents (SMA) afin de mesurer et d'évaluer la qualité du logiciel en cours d'exécution.

2.4. Mesure de la qualité

Le développement de logiciels de qualité, avec les coûts estimés et dans les délais prévus fait partie des objectifs essentiels des projets logiciels. Cependant, l'évaluation de ces objectifs n'est pas toujours évidente, en raison de la nature des produits logiciels et des projets qui complique la tâche de mesure et qui ne donne pas des valeurs correctes. Par conséquent, la mesure est devenue un domaine actif du génie logiciel (Laird & Brennan, 2006).

Evaluer quantitativement la qualité d'un logiciel revient à mesurer les différents aspects qui composent le logiciel. Il s'agit d'une activité importante qui donne aux développeurs, aux architectes et aux ingénieurs de qualité la possibilité d'appréhender la qualité de leurs produits et en conséquence, d'éclairer leurs choix. La definition de Kan (Kan, 2002) indique clairement cette importance « *Measurement plays a critical role in effective and efficient software development, as well as provides the scientific basis for software engineering that makes it a true engineering discipline. »*.

Fenton (Fenton & Pfleeger, 1998) propose de définir la mesure comme suit: « *Formally, we define measurement as a mapping from the empirical world to the formal, relational world. Consequently, a measure is the number or symbol assigned to an entity by this mapping in order to characterize an attribute».* Dans cette définition, Fenton considère qu'une mesure est utilisée pour caractériser un attribut d'un logiciel quantitativement. Cependant, il est indispensable, avant de mesurer, de spécifier la partie du produit à mesurer (code source, conception, …), d'identifier les caractéristiques importantes à mesurer (maintenabilité, portabilité, fiabilité, ...) et le destinataire de la mesure (pour qui mesurer ?).

Oman et Pfleeger (Oman & Pfleeger, 1997) ont distingué six objectifs pour une mesure : mesurer pour la compréhension, mesurer pour l'expérimentation, mesurer pour le contrôle de projet, mesurer pour l'amélioration du processus, mesurer pour l'amélioration du produit et mesurer pour la prédiction.

2.5. Mesure de la qualité des SMA

La mesure est une tâche vitale qui permet de contrôler la qualité des produits logiciels et leur évolution. Evidemment, une métrique ne peut être définie sans la prise en compte des spécificités des paradigmes de programmation. Le paradigme multi-agent, en tant que l'un des paradigmes de programmation relativement récents, ne fait pas exception. Plusieurs métriques spécifiques aux systèmes multi-agents sont proposées depuis l'émergence de ce paradigme.

Lass et al. (Las & al., 2009) ont présenté une étude des métriques pour les systèmes multi-agents. Ils ont décomposé les mesures en deux catégories : les mesures de performance et les mesures d'efficacité. De plus, ils ont cité les différents niveaux auxquels nous pouvons appliquer les métriques (l'agent, le framework, la plateforme et l'hébergeur). Ces auteurs ont également proposé un cadre permettant l'application des différentes métriques. Le cadre proposé est composé de trois étapes : la sélection, la collecte et l'application. La première étape permet de déterminer les métriques à appliquer en fonction des objectifs d'évaluation grâce à l'approche Goal, Question, Metric (GQM) (Basili & al., 1994). La seconde étape est utilisée pour collecter les données permettant l'application des métriques. Enfin, les métriques sont appliquées afin de déterminer si le système répond ou pas à l'objectif de l'évaluation. Les auteurs (Las & al., 2009) ont présenté plusieurs points intéressants dans leur travail. D'une part la décomposition du

SMA en système stratifié permet de maîtriser le processus de mesure et d'évaluation. D'autre part, les auteurs ont proposé un cadre qui permet d'appliquer les différentes métriques. Cependant, le cadre proposé ignorait la méthode de mesure. Les auteurs ont simplifié cet aspect en proposant l'instrumentation du code ou l'utilisation d'un chronomètre pour enregistrer les informations de chronométrage. Les méthodes de mesure méritent plus d'attention car elles peuvent influencer les résultats obtenus. De plus, cela peut devenir une tâche complexe et fastidieuse.

Sivakumar et Vivekanandan (Sivakumar, & Vivekanandan, 2012) ont proposé des métriques pour évaluer la réactivité des agents. Pour cela, ils décomposent cette caractéristique en trois sous-caractéristiques : l'interaction, la perception et la communication. Chacune de ces sous-caractéristiques est mesurée par un ensemble de métriques. Par exemple, la sous-caractéristique de perception est mesurée par l'utilisation et la mise à jour des connaissances. Bien que les attributs proposés semblent dynamiques tel que la mise à jour des connaissances, Sivakumar et Vivekanandan ont proposé des métriques statiques pour les mesurer. Par exemple, la mise à jour des connaissances est définie par « le nombre d'instructions qui mettra à jour les variables dans l'agent » (Sivakumar, & Vivekanandan, 2012). Ces auteurs (Sivakumar, & Vivekanandan, 2012) ont également développé un outil qui évalue les métriques proposées sur la plateforme Jade par une analyse syntaxique.

Mahar et Bhatia (Mahar, & Bhatia, 2014) ont ciblé la mesure de l'intelligence des agents. Les auteurs considèrent que les attributs qui déterminent l'intelligence de l'agent sont : l'adaptabilité, l'orientation vers un but et l'apprentissage. Ensuite, ils ont proposé des métriques pour chacun de ces attributs. Le nombre de rôles et la réalisation des objectifs de l'agent sont utilisés pour mesurer l'attribut d'orientation des objectifs. Loin des discussions sur les attributs d'intelligence proposés qui sont discutables en raison du manque de plusieurs attributs bien connus, il est cependant difficile de mesurer certaines métriques en utilisant uniquement le code source du logiciel (Marir & al., 2016). Les auteurs (Mahar, & Bhatia, 2014) ont proposé des métriques dynamiques mais ils n'ont pas présenté explicitement la méthode de mesure. Il semble que les fonctions qui calculent ces métriques soient incorporées dans le code du système analysé. Cette technique souffre de quelques inconvénients majeurs. Premièrement, le code incorporé peut influencer négativement les performances du système analysé, en raison de la consommation des ressources de calcul dans le calcul des fonctions de mesure. De plus, l'incorporation des fonctions de mesure manuellement dans le code du système est une tâche difficile. Enfin, bien que cette méthode assure la mesure des métriques dynamiques, l'application de cette technique nécessite l'existence du code de l'application qui est inaccessible dans la plupart des cas.

Afin de remédier aux inconvénients de cette approche basée sur l'incorporation manuelle de la fonction de mesure, certains outils ont été développés pour mesurer automatiquement des métriques spécifiques des systèmes multi-agents. Généralement,

ces outils sont appelés profilage. Un outil de profilage utilise la trace d'exécution de l'application afin de mesurer les métriques souhaitées. Bien que les outils de profilage soient proposés pour la première fois il y a plusieurs décennies, l'émergence des systèmes multi-agents a conduit au développement de plusieurs outils de profilage spécifiques pour ces systèmes (Doan & al., 2010a) (comme AgentSpotter). Évidemment, les outils de profilage assurent la caractéristique de réutilisabilité car ils permettent d'exécuter plusieurs applications à l'aide du même outil. Cependant, chaque outil de profilage est conçu pour mesurer certaines métriques spécifiques (généralement la performance). En fait, la caractéristique d'extensibilité (la possibilité d'étendre l'outil existant pour prendre en charge plus de métriques) est généralement une caractéristique omise.

Il est cependant important, de noter que l'outil MEANDER (Dimou & al., 2009) a tenté d'être extensible. Ainsi, les développeurs de cet outil donnent la possibilité d'introduire les métriques et l'outil les intègre dans le système multi-agents. Cependant, il est nécessaire d'avoir le code du système en cours d'évaluation pour pouvoir utiliser cet outil.

Enfin, Maarir et al. (Marir & al., 2016) proposent une approche pour mesurer les métriques dynamiques des systèmes multi-agents en utilisant la programmation orientée aspect. L'utilisation de la programmation orientée aspect pour mesurer les métriques dynamiques des systèmes multi-agents offre plusieurs avantages. Tout d'abord, les métriques sont développées indépendamment du système analysé en tant qu'aspects, ce qui permet leur réutilisation. Deuxièmement, il est facile d'étendre les métriques proposées pour mesurer plus d'attributs car les métriques sont indépendantes entre elles d'une part et indépendantes de l'application en cours d'analyse d'autre part. De plus, il est simple d'utiliser l'approche proposée car elle est entièrement automatique. Enfin, l'approche proposée peut être appliquée même si le code de l'application en cours d'analyse n'est pas disponible.

2.6. Synthèse

L'étude bibliographique que nous avons menée sur la qualité des SMA, révèle que :

- les normes internationales tels que ISO et IEEE, qui définissent des modèles de qualités se sont révélées trop générales pour faire face aux caractéristiques spécifiques des agents. Certaines caractéristiques de ces modèles sont adéquates pour l'évaluation des agents, tandis que d'autres ne sont pas bien adaptées à cette tâche. Par conséquent, plusieurs modèles de qualité spécifiques aux agents ont été définis. La plupart d'entre eux sont basés sur le modèle de qualité ISO/IEC 9126 (ISO/IEC 9126, 2001), avec quelques modifications afin de les rendre adéquats au domaine des agents.

- malgré, qu'il existe plusieurs propositions de métriques ad hoc dans la littérature (Sivakumar, & Vivekanandan, 2012) (Mahar, & Bhatia, 2014), aucune n'a fait l'objet d'une étude sérieuse quant à leur complétude, leur cohésion et surtout à leur aptitude à prédire la qualité externe des artefacts développés.

- l'absence de prise en charge de ces métriques par les outils d'analyse de code du marché rend impossible tout usage industriel. A ce jour, les développeurs des SMA sont donc condamnés : au mieux à utiliser, lorsqu'ils le peuvent, des métriques non spécifiques au monde agent mais sans réel certitude sur l'intérêt qu'ils ont à collecter et à interpréter telle ou telle mesure, au pire à n'utiliser aucune métrique. Dans les deux cas, la prédiction a priori de la qualité de leur développement est un pari hautement hasardeux avec pour conséquence une augmentation des coûts consécutives à la découverte tardive de défauts. En effet, en l'état actuel le contrôle (la surveillance) des applications agent est difficile à effectuer.

- 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 (ISO/IEC 9126, 2001).

Partant de ces constatations, nous croyons que la nature flexible des SMA nécessite l'application des métriques dynamiques. Comme l'application de ces métriques, n'est pas évidente, il est donc nécessaire de spécifier explicitement la méthode de mesure appliquée. La première question de recherche à laquelle nous avons tenté de répondre dans cette thèse a donc été :

Quels sont les métriques de qualité pouvant être utilisées afin de surveiller l'exécution des SMA et quelles sont les méthodes adéquates pouvant être utilisées pour la mesure de ces métriques ?

En raison des nombreux avantages de la Programmation Orientée-Aspect (AOP) , nous avons décidé dans cette thèse de l'adopter afin de mesurer certaines métriques dynamiques des Systèmes Multi-Agents (SMA). Ces métriques seront utilisées afin de surveiller l'exécution du SMA en vue de sa maintenance.

3. Notion de maintenance

La notion formalisée de maintenance est née dans l'industrie vers la fin des années 1970. La norme AFNOR NFX 13-306 a donné la définition suivante au terme de la maintenance (AFNOR, 2001) :

« *La maintenance est l'ensemble des actions qui permettent de maintenir ou de rétablir un bien dans un état spécifié ou en mesure d'assurer un service déterminé ».*

Selon cette définition, la maintenance permet de rétablir un système dans un état préalablement spécifié afin qu'il puisse fournir les fonctions pour lesquelles il a été conçu. L'analyse des différentes formes de maintenance repose sur trois concepts.

- **Les événements** qui sont à l'origine de l'action de maintenance : référence à un échéancier, résultat de diagnostic, information de capteur, mesure d'usure, apparition d'une défaillance, etc.

- **Les méthodes de maintenance** qui leur sont associées : maintenance préventive systématique, maintenance préventive conditionnelle, maintenance corrective, etc.

- **Les opérations de maintenance** : inspection, contrôle, dépannage, réparation, etc. Ces opérations de maintenance s'effectuent sur les composants du système complexe qu'ils soient matériels ou logiciels. Les activités de maintenance consistent essentiellement à réparer un composant lorsque celui-ci est défaillant. Un composant défaillant lorsqu'il n'est plus capable de réaliser les fonctions pour lesquelles il a été conçu.

Plusieurs définitions concernant la maintenance des logiciels ont été également proposées dans la littérature :

La définition la plus officielle est fournie par l'IEEE : « *la maintenance logicielle est le processus de modification des bogues potentiels, l'amélioration des performances logicielles et des attributs associés et les activités de modification du système logiciel afin de les adapter à un environnement de plus en plus changeant, après la livraison des produits logiciels aux utilisateurs* » (IEEE, 1994).

Selon Parikh, « *La maintenance est l'ensemble des activités effectuées sur un logiciel après sa livraison ou sa mise en opération* » (Parikh, 1986). Selon cette définition la maintenance logicielle peut être définie par toutes les modifications et mises à jour effectuées après la livraison du produit logiciel.

Dans les systèmes, ayant un nombre important de modules ou composants en interaction, la modification et ensuite la validation d'un logiciel peut s'avérer complexe : l'analyse, le test et le débogage peuvent être nécessaires pour chaque module individuel, puis globalement pour les interactions entre ces modules. Cette complexité s'accroît lorsque les chargés de la maintenance n'ont pas accès aux spécifications de la conception du système mais uniquement au code source. La tâche de maintenance du logiciel devient ainsi plus complexe et plus coûteuse dans ce cas. À cette complexité s'ajoute le fait que la maintenance des logiciels n'est souvent pas prise en compte dans les phases antérieures à leur mise en exploitation.

Les recherches sur les activités de maintenance logicielle ont abouti à de nombreux résultats. Ces résultats s'accordent et indiquent clairement que la réduction des coûts de la maintenance peut être atteinte d'une manière efficace par le contrôle du processus de conception et par des tests effectués plus tôt dans le cycle de vie du logiciel.

4. Types de maintenance

Plusieurs auteurs ont étudié la maintenance dans le but d'identifier les raisons à l'origine des besoins de modifications, ainsi que leurs fréquences. À la suite de ces études, plusieurs classifications des activités de maintenance ont été définies. Ces classifications aident à mieux comprendre la grande importance de la maintenance et ses implications sur le coût et la qualité des systèmes utilisés.

Quatre types de maintenance peuvent être appliquée sur le logiciel (Aggarwal & Singh, 2005) :

Maintenance corrective : consiste en la modification réactive exécutée après la mise en production d'un logiciel pour corriger les problèmes découverts par sa clientèle.

Maintenance adaptive : implique la modification d'un logiciel exécutée après sa mise en production et vise à maintenir le logiciel utilisable dans un environnement technique changé ou en cours d'évolution.

Maintenance perfective : est la modification d'un logiciel après sa mise en production pour améliorer son exécution ou sa maintenance.

Maintenance préventive : implique la modification d'un logiciel après sa mise en production pour détecter et corriger les défauts latents avant qu'ils deviennent des défaillances

Les travaux de maintenance adaptative et perfective sont considérés comme des améliorations du logiciel existant .

5. Maintenance préventive

La maintenance préventive est définie comme la modification d'un produit logiciel après la livraison pour détecter et corriger les défauts latents dans le logiciel avant qu'ils ne deviennent des défauts opérationnels (IEEE, 1994). Ce type de maintenance ne traite pas le fonctionnement du logiciel et ses fonctions opérationnelles, par exemple la sauvegarde, la récupération, l'administration du système, qui sont normalement exécutées par ceux qui utilisent le logiciel. Il est important de comprendre que la maintenance préventive n'est pas un participant au fonctionnement efficace du logiciel mais plutôt une mesure préventive qui travaille sur l'obsolescence du logiciel.

5.1 Objectifs de la maintenance préventive

La maintenance préventive est la maintenance planifiée du système logiciel conçue pour améliorer sa fiabilité et sa maintenabilité et réduire toute activité de maintenance imprévue. En d'autres termes, une maintenance préventive est effectuée pour modifier le logiciel afin de :

- Améliorer sa maintenabilité

- Réduire les défaillances du système

Améliorer la maintenabilité

La maintenabilité du logiciel est le degré, auquel il peut être compris, corrigé, adapté et / ou amélioré (IEEE, 1994). La maintenance préventive inclut dans ce cas :

- la conception d'un changement à un logiciel implémenté afin de rendre le système facile à maintenir.

- Mise à niveau continue d'un système pour lui permettre de faire face aux changements actuels et futurs : en rendant le logiciel moins complexe (Modularité), et plus facile à interpréter (Clarté)

- Documenter le code source ainsi que les processus support appropriés.

Réduire les défaillances

La maintenance préventive vise également à reporter ou à réduire les défaillances d'un système. Certains chercheurs considèrent la maintenance préventive des systèmes logiciels comme approche proactive de la tolérance aux pannes des logiciels opérationnels et visent à contrer l'effet du vieillissement (Garg and al., 1998a).

En dehors de la réduction du coût du projet, la maintenance préventive présente plusieurs avantages, qui consistent à :

- L'amélioration de la maintenabilité et de la convivialité des applications logicielles.
- Permettre des tests ciblés sur les composants bogués et sujets aux pannes du logiciel.
- La réduction globale du temps de réponse des activités.
- L'amélioration des performances et de la qualité des logiciels.

5.2. Types de la maintenance préventive

Une synthèse des politiques de maintenance préventive est donnée par (Rausand & Hoyland, 2004). Selon cette synthèse trois approches sont adoptées :

- Une approche basée sur des intervalles de temps définis appelée aussi maintenance préventive planifiées (systématique ou périodique)
- Une approche basée sur la mesure nommée aussi maintenance préventive non planifiées ou maintenance préventive conditionnelle.
- Une approche effectuée sur la base de l'estimation du temps de fonctionnement correct qui subsiste avant la défaillance, où on parle d'une maintenance préventive prévisionnelle.

5.2.1. Maintenance préventive planifiée (systématique/périodique)

Lorsque la maintenance préventive est effectuée dans des intervalles de temps définis, on parle de maintenance systématique ou périodique (Figure 2.2). Les opérations de maintenance sont effectuées selon un échéancier ou un calendrier déterminé a priori (Devarun, and Sandip, 2009). L'optimisation d'une maintenance préventive systématique

consiste à déterminer au mieux la périodicité des opérations de maintenance sur la base du temps, du nombre de cycles de fonctionnement...etc.

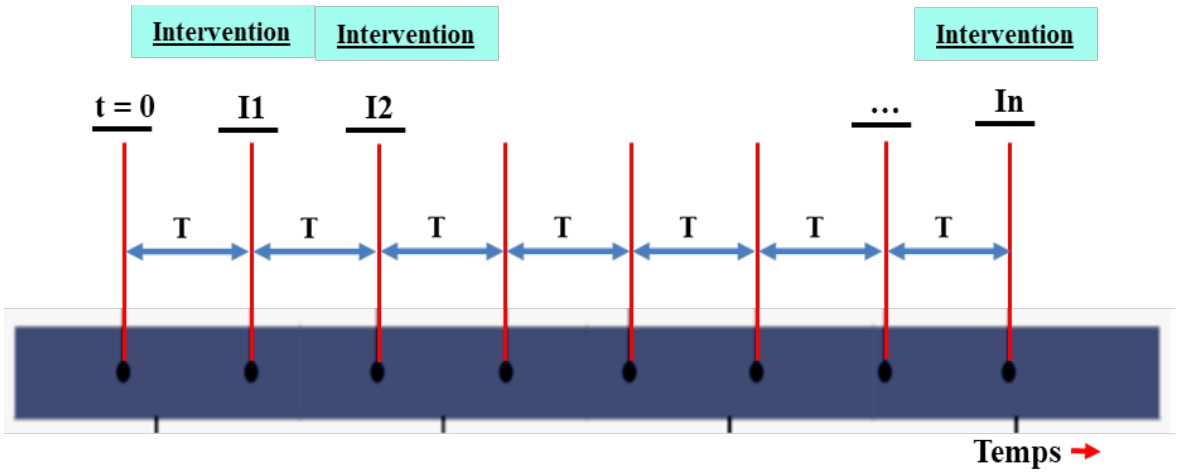


Figure 2.2. Intervention préventive systématique (Benaicha, 2015).

5.2.2. Maintenance préventive prévisionnelle

La maintenance prévisionnelle, également appelée maintenance proactive, est également réalisée à la suite d'une analyse de l'évolution surveillée des paramètres précurseurs de panne qui permettent de qualifier l'état de fonctionnement du système. La maintenance proactive est une forme de maintenance prédictive qui consiste à déterminer les causes à l'origine des défaillances et des usures précoces des équipements du système.

La maintenance prévisionnelle (Figure 2.3) permet d'anticiper et de prévoir au mieux le moment où l'opération de maintenance devra être réalisée. Cette forme de maintenance permet de réduire le nombre de défaillances imprévues, et donc l'indisponibilité du système. Elle permet de planifier les opérations de maintenance de manière à utiliser les équipements au maximum de leurs possibilités. En surveillant les équipements, il est possible de corriger des erreurs de conduite ou des anomalies qui peuvent générer des défaillances plus graves par la suite et d'améliorer la sécurité en évitant des accidents critiques. Par contre, cette forme de maintenance nécessite de mettre en place des techniques de surveillance et de mesure qui peuvent être très coûteuses.

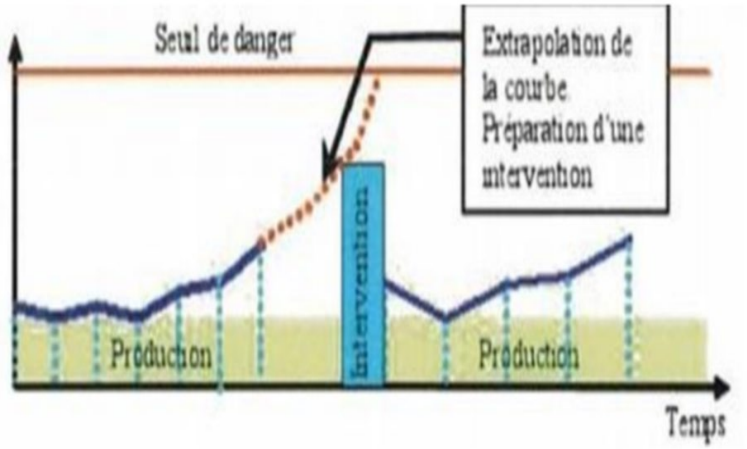


Figure 2.3. Schématisation de la maintenance prévisionnelle (Benaicha, 2015).

5.2.3. Maintenance préventive non planifiée (conditionnelle)

D'après la définition Afnor (AFNOR, 2002), il s'agit d'une forme de maintenance préventive basée sur une surveillance du fonctionnement du système. La maintenance conditionnelle (Figure 2.4) permet d'assurer le suivi continu du matériel ou logiciel en service, et la décision d'intervention est prise lorsqu'il y a une évidence expérimentale de défaut imminent ou d'un seuil de dégradation prédéterminé. Cela concerne certains types de défaut, de pannes qui surgissent progressivement ou par dérive. L'étude des dérives dans le cadre des interventions de maintenance préventive permet de déceler les seuils d'alerte, tant dans les technologies relevant de la mécanique que celles de l'informatique. Lors de la conception, on définit des tolérances pour certains paramètres. La variation progressive d'un paramètre n'implique pas la défaillance d'un organe. Mais lorsqu'un paramètre sort de la tolérance, le fonctionnement peut être complètement perturbé (Jea, 2010).

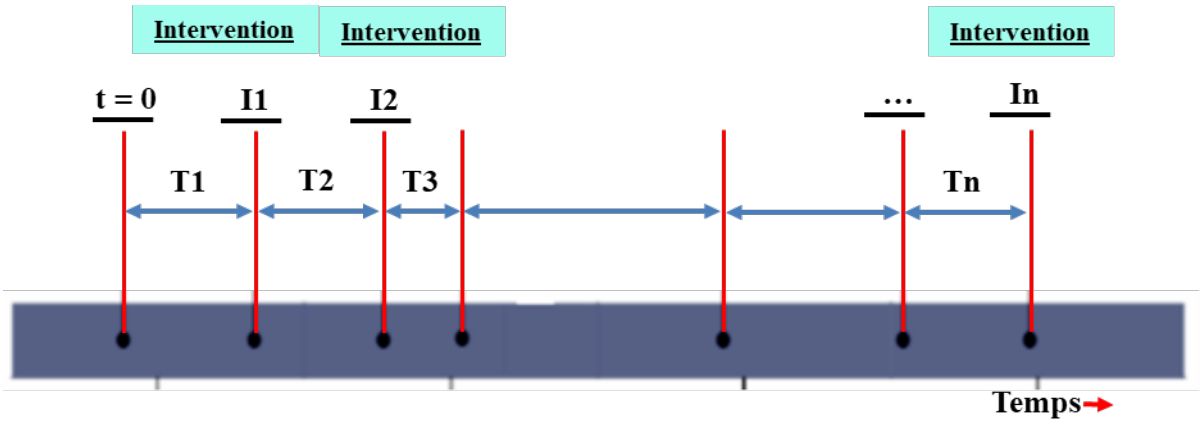


Figure 2.4. Intervention préventive conditionnelle (Benaicha, 2015)

L'application de ce type de maintenance préventive sur les logiciels est fondée sur des mesures permettant la détection de logiciels vieillissants et la prévision des défaillances liées au vieillissement, afin que des méthodes proactives puissent être appliquées pour éviter les pannes imprévues. L'idée de base de cette approche est de surveiller et de collecter des données sur les attributs responsables de la détermination de la santé du logiciel d'exécution. Les données sont ensuite utilisées pour obtenir des indices sur d'éventuelles défaillances imminentes (Garg & al.,1998b) (Vaidyanathan & Trivedi, 1999). Ce second type d'approche a été largement utilisé dans la maintenance matérielle mais rarement utilisé pour la maintenance des logiciels.

Avantages de la maintenance préventive conditionnelle

Plusieurs avantages de la Maintenance Préventive Conditionnelle (MPC) ont été signalés dans des travaux antérieurs et dans l'industrie (Shin, &, Jun, 2015) :

- La MPC permet un avertissement préalable d'une défaillance imminente et une précision accrue dans la prévision des défaillances. Ainsi, cette approche permet de réduire efficacement les défaillances du système par rapport à d'autres approches. De plus, elle facilite les procédures de diagnostic car il est relativement facile d'associer la défaillance à des composants spécifiques via les paramètres surveillés

- La MPC permet d'améliorer la sécurité des systèmes critiques (où la sécurité est primordiale) par la détection des problèmes à l'avance avant qu'ils deviennent graves. Ceci permet la satisfaction des clients grâce à l'assurance d'une qualité élevée et permet également aux gestionnaires de la maintenance d'éviter les coûts du risque dû à l'insatisfaction de la qualité des produits.

- La MPC permet une maintenance mieux planifiée et permet de réduire ou d'éliminer les inspections inutiles par la réduction des intervalles de la maintenance périodique en toute sécurité (Chen & al., 2012).

- L'utilisation de la MPC dans l'industrie permet de réduire les coûts en évitant les actions de maintenances inutiles et en permettant une planification plus efficace de la maintenance (Schwabacher, 2005). De plus, la MPC permet d'optimiser la production en permettant au système de continuer à fonctionner tant qu'il fonctionne dans les limites de performances prédéfinies (Prajapati & al., 2012).

Discussion

La stratégie de maintenance a des répercussions directes sur l'exploitation du système, la production et les charges financières. A chaque instant de l'exploitation du système, le décideur (gestionnaire) de maintenance doit faire un choix face aux interventions possibles sur le système afin de déterminer l'action à effectuer. Ce choix doit permettre de satisfaire aux mieux les objectifs fixés a priori et permettre ainsi une exploitation optimale du système. Cependant, ces objectifs peuvent être multiples et ne conduisent pas

toujours à une unique façon de procéder : une volonté de sécuriser le système exige une fréquence de maintenance préventive élevée alors que d'un point de vue économique, il peut être intéressant de ne pas trop intervenir pour ne pas ralentir le système. Il est donc nécessaire de trouver un compromis et un équilibre entre maintenance préventive et maintenance corrective. Une stratégie de maintenance qui semble prometteuse est la maintenance préventive conditionnelle à laquelle nous nous intéressons dans le cadre de ce travail.

6. Approches de maintenance préventive des logiciels

La maintenance préventive des systèmes logiciels est une approche proactive de la tolérance aux pannes des logiciels opérationnels, elle vise à contrecarrer l'effet de vieillissement (Garg et al., 1998a). Classiquement, le vieillissement des logiciels a deux principaux symptômes : un taux d'échec augmenté, et un taux de service réduit. Dans (Mira, 2000), deux types de vieillissement ont été identifiés : *vieillissement des logiciels* et *vieillissement des processus logiciels*. Le vieillissement des produits logiciels est une dégradation du code logiciel et de la qualité de la documentation par une maintenance continue. Le vieillissement de l'exécution des processus logiciels se manifeste par une dégradation des performances ou des défaillances transitoires dans les systèmes logiciels en fonctionnement continu.

6.1. Vieillissement des produits logiciels

L'amélioration de la maintenabilité est une méthode suggérée pour lutter contre le vieillissement des logiciels. Afin de fournir une évaluation précoce de la maintenabilité du système, des modèles ont été proposés pour prédire la difficulté de changement dans le processus de maintenance (Briand, 1993). Diverses technologies de « *rétro-ingénierie et de réingénierie* » ont été suggérées pour améliorer la maintenabilité (Bennet, 1998) (Lanubile, &, Visaggio, 1995) (Von Mayrhauser, &, Wang, 1999). Selon Pearse (Pearse, 1995), le processus de réingénierie devrait se dérouler en même temps que le processus général de maintenance. Il faut surveiller périodiquement la « santé » du système et prévenir les « maladies » du système en vérifiant le niveau de maintenabilité du système. La fréquence d'application du processus de réingénierie ne devrait pas être liée au taux de modification mais au niveau de maintenabilité (Lanubile, &, Visaggio, 1995).

L'utilisation de directives de « *style de programmation* » a été suggérée comme méthode pour améliorer la maintenance préventive. Un bon style de programmation peut réduire l'impact du changement et réduire ainsi les coûts de maintenance (Lieberherr, &, Holland, 1989).

L'écriture de *« logiciels réutilisables »* (reusable software) a également été proposée pour promouvoir la maintenance préventive (Lieberherr, & Holland, 1989). La réutilisation d'un logiciel éprouvé augmente l'efficacité pendant la phase de maintenance corrective.

La maintenance préventive concerne également les activités visant à améliorer la maintenabilité du système en mettant à jour la documentation, en ajoutant des commentaires et en améliorant la structure modulaire du système (Kajko-Mattsson, 1999). Le travail de la maintenance préventive (réingénierie essentiellement incrémentielle) peut être supporté pour améliorer le système et le rendre plus facile à changer. Il a également été considéré comme un élément clé de la gestion des risques de l'entreprise. Les risques sont évalués et identifiés afin que les mesures d'atténuation nécessaires puissent être prises pour éviter toute perte inutile à l'avenir.

6.2. Vieillissement de l'exécution des processus logiciels

Les caractéristiques de performance d'un système logiciel sont dégradées au cours du temps par un fonctionnement continu. Les effets se manifestent par une réduction des performances du service et / ou des pannes (panne ou blocage du système). D'autres problèmes tels que l'incohérence des données, les fuites de mémoire, les verrouillages de fichiers inédits, la corruption de données, la fragmentation de l'espace de stockage et une accumulation d'erreurs d'arrondi peuvent également se produire.

Le *« rajeunissement du logiciel »* a été préconisé comme mesure de maintenance préventive (Avritzer & Weyuker, 1997) (Garg et al., 1998a). Le logiciel est périodiquement arrêté et redémarré afin de rafraîchir son état interne. Cela empêche, ou au moins reporte l'occurrence des échecs. Bien que le rajeunissement logiciel implique des frais généraux, il prévient des défaillances plus sévères et donc plus coûteuses.

D'autres techniques de maintenance préventive ont été proposées dans la littérature pour identifier les défauts avant le déploiement du logiciel. Ces techniques incluent, sans s'y limiter, la prédiction des erreurs (Ambros & al., 2010), les tests de régression (Pasala & al., 2008), et la conformité de l'architecture (Dennis & al., 2005). Ces techniques utilisent les informations passées, effectuent des analyses sur les artefacts logiciels disponibles tels que les historiques de versions de code, les historiques de bogues, la documentation, etc. et fournissent des solutions qui identifient les défaillances potentielles du logiciel. Cela permet au développeur de prendre des mesures préventives appropriées pour éviter les défaillances pendant que l'utilisateur final utilise le logiciel.

Synthèse

Un examen des différentes techniques suivi dans la maintenance préventive a révélé que :

- La recherche contemporaine sur la maintenance préventive est plus répandue dans le contexte matériel (Kajko-Mattson, 2001) alors que dans la communauté logicielle, elle est encore dans une phase naissante (Card & Glass, 1990) (Singh & Goel, 2007) (Cheluvaraju et al, 2012). Cependant, des mesures de qualité qui indiquent divers facteurs du logiciel comme la maintenabilité et la fiabilité sont à l'étude. Alors que les approches sur la maintenabilité (Card & Glass, 1990) (Dennis & al., 2005) (Sarkar & al., 2007) se

concentrent sur l'amélioration des aspects de conception du logiciel, les techniques de fiabilité (DAmbros & al., 2010) (Pasala & al., 2008) se concentrent sur l'estimation de la probabilité de défaillances potentielles.

- Comme indiqué par Cheluvaraju et al. (Cheluvaraju et al, 2012), plusieurs approches proposées pour améliorer la maintenabilité du logiciel sont appliquées lors du développement du logiciel et ne permettent d'avoir aucune information sur le logiciel après sans déploiement (Ambros & al., 2010) (Pasala & al., 2008) (Dennis & al., 2005).

- Les métriques qui fonctionnent mieux sur un système logiciel ne peuvent fonctionner correctement sur d'autres systèmes. Les métriques de code utilisés pour la prédiction des erreurs par exemple telles que le couplage entre objets, et la suite de métriques Chidember et Kemerer (CK) (Chidamber and Kemerer, 1994) ne sont applicables qu'aux systèmes de nature statique tel que les systèmes orientés objet par exemple. D'autres métriques telles que les métriques de complexité, et les métriques de changement sont appliquées à un niveau trop bas pour fournir un aperçu de niveau supérieur aux gestionnaires. Les métriques d'entropie sont trop lourdes et nécessitent une analyse de programme considérable avec peu d'amélioration des résultats. Ces métriques n'ont pas été conçues pour fournir des informations de plus haut niveau aux managers lors de la prise de décision

- Le rajeunissement du logiciel qui est utilisé par la plupart des approches comme technique de réparation du vieillissement des processus est couteuse et ne convient pas aux SMA qui sont complexes et imprévisibles.

A partir de l'étude que nous avons menée, nous concluons que les approches de maintenance préventive proposées dans la littérature ne conviennent pas aux SMA, car d'une part les approchent qui se concentrent sur l'amélioration de la maintenabilité du code du logiciel, sont appliquées dans la majorité des cas lors du développement du logiciel et d'autres part ces techniques se basent sur des métriques du code logiciel qui sont statiques, de niveau trop bas et sont mesurées avant le déploiement du logiciel.

La maintenance préventive reste donc un problème non résolu et aucune des approches qui sont proposées jusqu'à ce jour n'est utile dans tous les systèmes notamment dans les systèmes dynamiques.

7. Etude comparative de travaux sur la maintenance préventives

La maintenance est la phase la plus longue et souvent la plus coûteuse du cycle de vie du logiciel. Cependant peu de travaux se sont intéressés à la maintenance des logiciels et en particulier à la maintenance préventive des logiciels (Vaidyanathan & al., 2002) (Garget & al., 1998a) (Rahme and Haiping, 2017) (Huang & al., 1995). Dans cette partie nous exposons quelques travaux de maintenance qui sont proches à notre problématique.

En utilisant un processus de régénération de Markov avec un processus de récompense semi-Markov subordonné, Vaidyanathan et al. (Vaidyanathan & al., 2002) ont proposé un modèle analytique d'un système logiciel utilisant la maintenance préventive basée sur l'inspection. En traitant le phénomène du vieillissement des logiciels, les auteurs ont voulu montrer que la maintenance préventive basée sur l'inspection est dans de nombreux cas avantageuse par rapport à la maintenance préventive non basée sur l'inspection.

Garg et al. (Garget & al., 1998a) ont présenté un modèle analytique d'un système logiciel servant les transactions. Les auteurs ont considéré trois mesures, la disponibilité du logiciel pour fournir le service, la probabilité de perte d'une transaction et le temps de réponse d'une transaction. Ce modèle a été proposé afin de contrer le phénomène de vieillissement des logiciels et d'éviter les surcoûts qui peuvent en découler. Par conséquent, il était nécessaire de suivre une approche basée sur l'analyse pour déterminer les moments optimaux pour effectuer la maintenance préventive.

Rahmeet et Haiping (Rahme & Haiping, 2017) ont abordé les défauts liés au vieillissement des logiciels qui peuvent entraîner une dégradation des performances ou une augmentation des taux de défaillance des composants du système. Sur la base de leurs travaux précédents, les deux auteurs ont étudié comment dériver des programmes de maintenance préventive pour les systèmes logiciels basés sur le cloud et qui sont soumis à des taux de défaillance non constants. Ils ont adopté la distribution de Weibull (Abernethy, 1996) pour modéliser un taux de défaillance croissant pour les composants logiciels présentant des problèmes de vieillissement logiciel. Enfin, ils ont utilisé une étude de cas pour montrer que leur approche analytique peut soutenir efficacement le développement de calendriers de rajeunissement des logiciels pour la maintenance préventive des systèmes logiciels basés sur le cloud.

Huang et al. (Huang & al., 1995) proposent une approche préventive et proactive pour gérer les pannes logicielles transitoires. Ils présentent un modèle d'analyse du rajeunissement du logiciel dans des applications fonctionnant en continu et exprime les temps d'arrêt et les coûts dus aux temps d'arrêt pendant le rajeunissement en termes de paramètres dans ce modèle. Des conditions de seuil pour que le rajeunissement soit bénéfique sont également dérivées. Les auteurs ont également mis en place un module réutilisable pour effectuer le rajeunissement du logiciel. Ce module peut être intégré dans n'importe quelle application existante sur une plate-forme UNIX.

Vaidyanathan, K. et Trivedi, K.S. (Vaidyanathan, K. and Trivedi, 1999) proposent un modèle basé sur la mesure pour estimer le taux d'épuisement des ressources du système d'exploitation à la fois en fonction du temps et de l'état de la charge de travail du système. Le modèle de récompense semi-Markov est construit sur la base des données de charge de travail et d'utilisation des ressources collectées à partir du système d'exploitation UNIX. Ils identifient d'abord différents états de charge de travail à l'aide d'une analyse de

cluster statistique et construisent un modèle d'espace d'état. Correspondant à chaque ressource, une fonction de récompense est alors définie pour le modèle en fonction du taux d'épuisement des ressources dans les différents états. Le modèle est ensuite résolu pour obtenir les tendances et les taux d'épuisement estimés et le temps d'épuisement des ressources. Avec l'aide de cette mesure, des techniques proactives de gestion des pannes telles que « le rajeunissement du logiciel » peuvent être utilisées pour éviter les pannes inattendues.

Garg et al. (Garg & al., 1998b) proposent une méthodologie de détection et d'estimation du vieillissement dans le système d'exploitation UNIX. Tout d'abord, ils présentent la conception et la mise en œuvre d'un outil de surveillance distribué basé sur SNMP (Simple Network Management Protocol) (Case & al., 1990), utilisé pour collecter des données sur l'utilisation des ressources du système d'exploitation et l'activité du système à intervalles réguliers, à partir de postes de travail UNIX en réseau. Des techniques de détection de tendances statistiques sont appliquées à ces données pour détecter/valider l'existence du vieillissement. Pour quantifier l'effet du vieillissement des ressources du système d'exploitation, ils proposent une métrique « Temps estimé jusqu'à épuisement » qui est calculée à l'aide de techniques d'estimation de pente. Bien que l'outil de collecte de données distribué soit spécifique à UNIX, les techniques statistiques peuvent également être utilisées pour la détection et l'estimation du vieillissement dans d'autres logiciels.

Critères / Approche	Type maintenance		Système maintenu	Approches		Actions de maintenance		Métriques à surveiller
	Systématique	Conditionnéelle		Modèle analytique	Mesure Données	Surveillance	Réparation	
Vaidyanathan & al, 2002	√		Systèmes opérationnels	√		√	Rajeunissement	Cout, Temps
Garg & al. 1998a	√		Systèmes Transactionnel	√		√	Non	Disponibilité, probabilité de perte, temps réponse
Rahme & Xu., 2017	√		Systèmes basés sur le cloud	√		√	Rajeunissement	Fiabilité
Huang & al. 1995	√		Applications UNIX	√		√	Rajeunissement	Cout, Temps
Vaidyanathan & Trivedi,1999		√	Applications UNIX		√	√	Non	Charge de travail, charge d'utilisation des ressources
Garg & al., 1998b		√	Applications UNIX		√	√	Non	Charge d'utilisation des ressources

Table 2.1. Recueil des approches de maintenance préventive

Lors de cette étude bibliographique sur les approches de maintenance préventives des systèmes opérationnels, nous avons rencontré de nombreuses techniques illustrant la diversité des hypothèses qui peuvent être posées sur le système, son mode de défaillance, de dégradation, les exigences et les objectifs de la maintenance à mettre en place ou encore les opérations de maintenance disponibles, etc. La table 2.1 résume les différences dans le type de maintenance, le système maintenu, la méthode suivie (Modèle analytique/Mesure de données), les actions de maintenance (Surveillance/Réparation) dans des travaux antérieurs sur la maintenance préventive et montre également les différences dans les mesures évaluées.

Système maintenu : Toutes les approches étudiées concernent la maintenance préventive des logiciels (Table 2.1). Cependant, aucun de ces travaux ne traite la maintenance préventive des SMA.

Type maintenance (systématique/conditionnelle) : La majorité de ces travaux est dédiée à la maintenance préventive mais peu d'entre eux s'intéressent à la maintenance préventive conditionnelle (Vaidyanathan, & Trivedi,1999) (Garg & al., 1998b). Or, l'approche conditionnelle semble prometteuse en particulier pour des systèmes (complexes et dynamiques) où la défaillance a un impact important sur le coût d'exploitation ou sur la sécurité du système. En effet, l'approche conditionnelle est plus efficace que l'approche systématique car elle permet d'intégrer des informations sur l'état courant du système (Gertsbakh, 2000) dans le processus de décision.

Approche (Modèle analytique/Mesure des données)

Deux approches ont été adoptées pour la maintenance préventive des systèmes opérationnels, la modélisation analytique et l'approche basée sur la mesure. L'objectif de la modélisation analytique est de déterminer les moments optimaux pour effectuer le rajeunissement. Une telle analyse analytique a été effectuée pour différents types de systèmes logiciels présentant diverses caractéristiques de défaillance/vieillissement (Garg & al., 1998a) (Huang & al., 1995). L'approche basée sur la mesure traite de la détection de l'existence d'un vieillissement logiciel et de la prévision des défaillances liées au vieillissement, de sorte que des méthodes proactives puissent être appliquées pour éviter les pannes imprévues. L'idée de base est de surveiller et de collecter périodiquement des données sur les attributs responsables de la détermination de la santé du logiciel d'exécution. Les données sont ensuite utilisées pour obtenir des indices sur d'éventuelles défaillances imminentes (Garg & al., 1998b) (Vaidyanathan & Trivedi, 1999).

Actions de maintenance (Surveillance/Réparation) : Les approches de maintenance étudiées proposent des techniques qui se limitent uniquement à la surveillance du système pour la détection du vieillissement du logiciel. Cependant, aucune de ces approches ne regroupe les deux actions de surveillance et de réparation au même temps (nécessaire tout deux pour la maintenance du système). A l'exception du rajeunissement

du logiciel utilisé par certaines approches pour la maintenance du système aucune action de réparation n'est proposée par ces dernières pour la prévention du système des défaillances latents.

Métriques à surveiller : La majorité des approches de maintenance visent à optimiser le temps et le coût moyen de la maintenance (Vaidyanathan & al, 2002) (Garget & al. 1998a) (Huang, & al. 1995). Nous trouvons également des travaux qui s'intéressent à la mesure de la charge de travail ou l'utilisation des ressources (Vaidyanathan, & Trivedi, 1999) (Garg, & al.1998b). Les mesures de qualité qui indiquent divers facteurs du logiciel comme la fiabilité sont également à l'étude (Rahme & Xu, 2017). Cependant, toutes les métriques utilisées par ces approches en phase de surveillance sont générales et ne permettent pas de considérer les spécificités des logiciels récents tels que l'autonomie et la sociabilité dans les SMA par exemple.

Bilan

La comparaison des résultats obtenus fait ressortir les points suivants :

– Peu de travaux ont porté sur la maintenance préventive des logiciels de type conditionnelle. Cependant, un nombre important d'approches, récentes même, se sont penchées sur la maintenance préventive conditionnelle du matériel (Ye & Xie, 2015) (Guo & al., 2016) (Moyahabo, & Opeyeolu, 2021).

- Les approches de maintenance préventive des logiciels sont anciennes (Garg & al. 1998a) (Vaidyanathan & Trivedi,1999). En effet, il est très difficile de trouver des approches récentes sur la maintenance préventive conditionnelle des logiciels (Rahme & Xu. 2017). Nous pensons que cela est dû à l'ouverture et la dynamique des logiciels récents de la nouvelle génération qui rendent leur étude et en particulier leur maintenance une tache très compliquée. De plus, aucun des travaux qui portent sur la maintenance préventive des logiciels ne traite la maintenance préventive des SMA.

- L'évolution de systèmes soumis à une politique de maintenance est souvent modélisée par des modèles analytiques qui permettent de déterminer les moments optimaux pour effectuer le rajeunissement du logiciel. Cette approche a la capacité d'intégrer la compréhension physique du produit cible, et repose sur l'utilisation du modèle analytique pour représenter le comportement du système, et les phénomènes de dégradation. Mais elle a la limitation au point qu'elle ne peut être appliquée qu'à des types spécifiques de produits.

- La plupart des approches de maintenance préventive consistent à modéliser la défaillance du système par le biais d'un processus stochastique. Le système est considéré défaillant si son niveau de détérioration dépasse un seuil fixé (Van Noortwijk, & Cooke, 1997). Mais, en pratique, à cause de la complexité des systèmes et de l'influence des différentes variables et de l'environnement sur les modes de fonctionnement des

systèmes, il est difficile, dans la majorité des cas, de réduire les modes de défaillance à un unique processus de dégradation, ce qui est le cas des systèmes multi-agent de nature complexe et dynamique que nous étudions.

– Le critère de maintenance à optimiser, le plus fréquemment rencontré dans la littérature est le temps et le coût moyen de la maintenance. Par contre, si le niveau de dégradation peut être modélisé, une structure de maintenance se basant sur des mesures de performances du système permettra l'optimisation du critère de performance choisi.

– Lorsque le taux de défaillance du système est croissant, la mise en place d'une politique de maintenance à inspections non périodiques (conditionnelle) permet d'améliorer les performances de la politique de maintenance comparée à une stratégie d'inspections périodiques.

– Le rajeunissement du logiciel qui est utilisé par la plupart des approches comme technique de réparation du logiciel en cours d'exécution, est couteuse et ne convient pas aux SMA qui sont complexes et imprévisibles.

– Nous avons pu constater que les modèles utilisés comme outils d'aide à la décision pour la maintenance sont essentiellement basés sur l'hypothèse peu réaliste que le système évolue dans un environnement statique ou du moins évolue dans un environnement qui n'impacte pas sa dégradation.

A la fin de cette étude, nous constatons que la majorité des approches proposées ignorent les métriques de qualité qui permettent de donner des informations très utiles sur l'état du système et qui aident énormément à prévenir le système en exécution d'éventuelles dégradations de ses performances. De plus, ces approches se trouvent impuissant devant la nouvelle génération de logiciels de nature dynamique et imprévisible qui ont besoin de méthodes bien adaptées, d'où la nécessité de :

- Faire appel à *des techniques de maintenance se basant sur des mesures*_afin de surveiller l'état du système pendant son exécution
- Recourir à de nouvelles métriques qui prennent en compte l'aspect dynamique des systèmes complexes de la nouvelle génération.

L'objectif de cette thèse consiste donc, à fournir une aide à la décision aux gestionnaires de la maintenance des SMA à travers un ensemble de métriques dynamiques qui permettent d'avoir les informations nécessaires sur la qualité du système en exécution, afin de le prévenir des erreurs pouvant mettre fin à sa vie. Par conséquent, il est nécessaire de proposer des techniques de maintenance adaptées aux SMA et d'avoir un ensemble de métriques légères qui non seulement sont en corrélation avec les erreurs, mais qui sont également utiles pour fournir une aide à la décision aux gestionnaires de la maintenance.

8. Conclusion

Un problème de maintenance doit avant tout, choisir une stratégie, bien étudier les caractéristiques du système considéré et le type de données de dégradation disponibles. Le présent chapitre a été divisé en deux grandes parties. La première partie a été consacrée à une étude de la qualité logicielle. Dans la seconde partie du chapitre nous avons présenté des notions de base liés à la maintenance. Dans cette partie, nous avons d'abord introduit des définitions d'ordre général et des concepts de base concernant la maintenance et les différents types de maintenance. Ensuite, nous avons mis en évidence la maintenance préventive comme étant une technique pour la prévention des systèmes d'éventuelles défaillances et nous avons présenté également les différentes stratégies de maintenance préventive existants dans la littérature.

Une étude comparative des différents travaux sur la maintenance préventive a également été montrée et discutée selon plusieurs critères techniques. Cette étude nous aide énormément à surpasser les lacunes observées pour nos propositions de maintenance dans les chapitres suivants.

A partir de l'étude que nous avons mené le long de ce chapitre, notre premier objectif (chapitre 3) consiste en la proposition d'une stratégie de maintenance préventive conditionnelle pour les SMA. Cette stratégie permet la surveillance de la qualité du SMA en exécution à travers des métriques qui prennent en compte l'aspect dynamique des SMA et l'environnement dans lequel évolue ces systèmes. La stratégie de maintenance proposée devra être élaborées de façon à rester à la fois suffisamment simple à implanter pour être utilisable dans un contexte opérationnel et suffisamment complexe pour rendre compte de la réalité des pratiques de maintenance et des spécificités innovantes des systèmes multi-agents (autonomie, adaptation, réactivité, etc.).

Le second objectif (chapitre 4) consiste à étendre l'approche proposée dans le contexte des SMA organisationnels. On proposera alors une approche de maintenance préventive se basant sur d'autres métriques spécifiques à ces systèmes organisationnels. L'approche que nous proposerons permet non seulement de surveiller le système en exécution, mais permet également d'appliquer des actions de maintenance (réparation) afin de contrôler le système et le prévenir du danger.

Chapitre 3

Une approche de maintenance préventive pour les Systèmes Multi-Agents

Sommaire

1. Introduction

La maintenance logicielle est une tâche importante et cruciale dans le cycle de vie du développement logiciel. Bien qu'elle représente plus d'efforts que toute autre activité de génie logiciel, la maintenance logicielle est encore une phase négligée dans le processus de génie logiciel (Singh & Goel, 2007). Il existe quatre catégories de maintenance : corrective, adaptative, perfective et préventive. Cette dernière est considérée comme une maintenance effectuée dans le but de prévenir les problèmes avant qu'ils ne surviennent (ISO, 2010).

La maintenance préventive du logiciel peut être systématique, prévisionnelle ou conditionnelle. En outre, plusieurs modèles de maintenance logicielle ont été proposés, qui incluent *Quick-Fix Model, Iterative Enhancement Model* et *Reuse-Oriented Model*. Dans le cadre des travaux actuels, nous nous intéressons à la maintenance préventive conditionnelle basée sur le Modèle Quick-Fix pour deux raisons essentielles. Premièrement, la maintenance préventive conditionnelle dépend de l'expérience et implique des informations collectées en temps réel. Deuxièmement, le modèle de solution rapide est utilisé pour identifier le problème, puis le résoudre le plus rapidement possible. Son avantage, par rapport aux deux autres modèles, est qu'il effectue son travail rapidement et à faible coût.

Dans la littérature, peu de travaux de recherche (Garg & al., 1998a) (Singh & Goel, 2007) (Cheluvaraju & al., 2012) (Sun & Wang, 2012), ont été réalisés sur la maintenance préventive. Cependant, à notre connaissance, aucun travail traitant de la maintenance préventive des systèmes multi-agents. Dans ce chapitre, nous proposons une approche originale de la maintenance préventive conditionnelle des systèmes multi-agents. L'approche proposée est basée sur des mesures de qualité MAS, elle utilise la programmation orientée aspect, et se compose de trois étapes majeures: (i) mesurer deux métriques de qualité (autonomie et sociabilité) de l'application en cours d'exécution de manière dynamique et continue en utilisant le code AspectJ et les comparer avec des seuils minimaux préalablement définis par le concepteur, (ii) avertir le mainteneur en cas de détection de régression anormale de la qualité MAS, et (iii) intervention du mainteneur pour préserver la qualité de l'application et éviter ainsi des dommages potentiels. De plus, l'approche proposée est soutenue par PMMAS (Maintenance Préventive des Systèmes Multi-Agents), un outil que nous avons développé.

Nous entamons ce chapitre par une introduction des préliminaires de l'approche proposée. Ensuite nous présentons l'approche et nous discutons les différents aspects techniques de l'outil développé. Finalement, nous illustrons l'outil que nous avons développé à partir d'une étude de cas concrète et nous montrons son apport par rapport aux travaux discutés dans le chapitre précédent.

2. Préliminaires

Nous introduisons, dans cette section, quelques concepts et outils de base liés à l'approche proposée, à savoir, JADE (Bellifemine, and al., 2007), AspectJ (Kiczales, & al, 2001) et le diagramme de cause-effet (Ron, 2008).

2.1. Plateforme JADE

JADE (Java Agent DEvelopment Framework) (Bellifemine, and al., 2007) est un Framework logiciel entièrement implémenté en langage Java. Il simplifie la mise en œuvre des SMA grâce à un middle-ware conforme aux spécifications FIPA et à un ensemble d'outils graphiques prenant en charge les phases de débogage et de déploiement.

La plate-forme d'agent peut être distribuée sur des machines (qui n'ont pas besoin de partager le même système d'exploitation) et la configuration peut être contrôlée via une interface graphique distante. JADE est actuellement la plate-forme la plus utilisée à des fins de recherche. Il comporte trois modules principaux :

- Le DF (facilitateur d'annuaire) fournit des services de pages jaunes à d'autres agents. Les agents peuvent enregistrer leurs services auprès du DF ou interroger le DF pour savoir quels services sont proposés par d'autres agents, y compris la découverte d'agents et de leurs services offerts dans des réseaux ad hoc.

- L'AMS (Agent Management System) contrôle l'accès et l'utilisation de la plate-forme d'agent et fournit des services tels que la maintenance d'un répertoire de noms d'agents. Il fournit des services de page blanche à d'autres agents. Chaque agent doit être enregistré auprès d'un AMS.

- L'ACC (Agent Communication Channel) gère la communication entre les agents.

2.2. AspectJ

AspectJ (Kiczales, & al, 2001) est une extension transparente orientée aspect de Java. Il permet de modulariser davantage et plus proprement toutes les préoccupations dans un système complexe par rapport au paradigme orienté objet AspectJ (Kiczales, & al, 2001). AspectJ ajoute à Java plusieurs nouvelles constructions, y compris des : points de jointure, pointcuts, advice, des déclarations intertypes et des aspects. Un aspect est une unité modulaire de mise en œuvre transversale dans AspectJ. Chaque aspect encapsule des fonctionnalités qui peuvent recouper plusieurs classes dans un programme. Les points de jointure sont des points bien définis dans l'exécution du programme, tels que l'appel de méthode (un point où une méthode est appelée), l'exécution de méthode (un point où une méthode est appelée) et les points de jointure de réception de méthode (un point où un a reçu un appel, mais cette méthode n'est pas encore exécutée). Les pointcuts sont un moyen de faire référence aux collections de comportement supplémentaire aux points de jonction. Il se compose d'instructions qui s'exécutent avant (before), après (after) ou

autour (around) d'un point de jointure. Le advice autour (around) s'exécute à la place du point de jointure indiqué, ce qui permet à l'aspect de remplacer une méthode. Un aspect peut également utiliser une déclaration intertype pour ajouter une déclaration d'implémentation de méthode, de champ ou d'interface publique ou privée dans une classe (Xie & al., 2006). AspectJ a deux types de tissage : un tissage statique où il fournit tous les codes correspondant aux points de jointure déclarés dans un Pointcut sans exécution, et un tissage dynamique où seuls les points de jointure exécutés sont interceptés à l'exécution. Le plugin AspectJ est une extension orientée aspect du langage de programmation Java. AJDT (AspectJ Developpement Tools) est le nom de ce plugin AspectJ.

2.3. Modèle Quick-Fix

Le modèle Quick-Fix (Penny and Takang, 2003) (Figure 3.1) est une approche ad hoc utilisée pour la maintenance du système logiciel. Le but de ce modèle est d'identifier le problème et de le résoudre le plus rapidement possible. L'utilisation de ce modèle permet également d'éviter le processus fastidieux du cycle de vie de la maintenance logicielle.

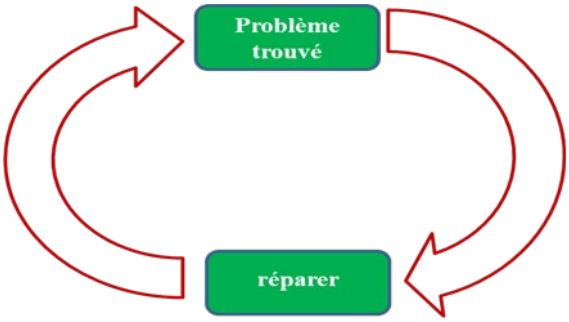


Figure 3.1. Modèle Quick-Fix

2.4. Diagramme Cause-Effet

Un diagramme Cause-Effet (Ron, 2008) (Figure 3.2) est un outil graphique utilisé pour une analyse de cause à effet, où on essaye d'identifier les causes possibles d'un certain problème ou événement.

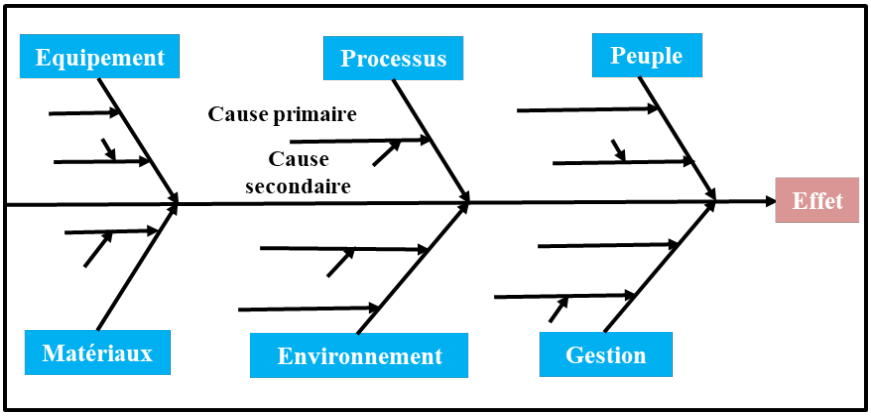


Figure.3.2. Exemple d'un diagramme Cause-Effet (Wittwer, & Fishbone, 2009)

Le but d'une analyse de cause à effet est d'identifier les causes, les facteurs ou les sources de variation qui mènent à un événement, un résultat ou un défaut spécifique dans un produit ou un procédé (Wittwer, & Fishbone, 2009).

3. Approche proposée

L'approche que nous proposons (Figure 3.3), dans ce chapitre, traite la maintenance préventive conditionnelle pour les applications multi-agents. Cette approche permet de mesurer en continu certaines métriques de qualité à l'aide du code de contrôle écrit en AspectJ et de le comparer avec des seuils minimaux préalablement définis par le concepteur. Lorsque les valeurs mesurées sont inférieures à celles spécifiées par le concepteur, une intervention préventive doit être effectuée pour remettre le système dans son état de fonctionnement souhaité.

Il est à souligner que les métriques de qualité concernent différents attributs des systèmes multi-agents tels que : l'autonomie, la sociabilité, la réactivité, la proactivité, la rationalité, l'adaptabilité. Dans ce chapitre, nous nous intéressons aux deux attributs qui influencent le plus la qualité des logiciels multi-agents, à savoir l'autonomie et la sociabilité.

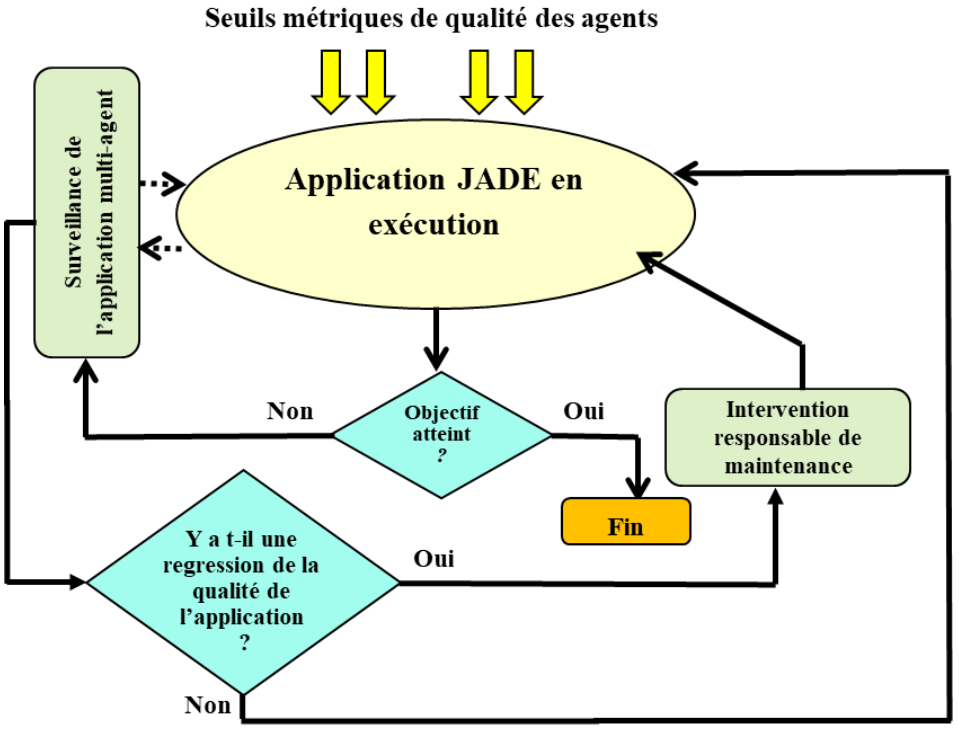


Figure 3.3. Méthodologie de l'approche proposée (Ghrieb et al., 2020)

3.1. Métriques de qualité mesurées

Pour la validation de notre approche, nous choisissons deux métriques essentielles des systèmes multi-agents, l'autonomie et la sociabilité.

Autonomie : afin de mesurer l'autonomie des agents dans les applications JADE, nous avons utilisé la formule métrique suivante proposée par Marir et al. (Marir & al., 2016):

AUTONOMIE de l'agent = 1 - (RS / EB) (1)

Où RS est le nombre de demandes de services. EB est le nombre de comportements exécutés.

Sociabilité : C'est le degré de capacité d'un agent à interagir avec les autres pour atteindre ses objectifs. La sociabilité peut être mesurée par plusieurs métriques. Dans notre travail, nous considérons que toute action de communication fait partie de la sociabilité. La formule utilisée pour mesurer la sociabilité est la suivante :

SOCIABILITÉ = 1 - (MN / EB) (2)

Où MN est le nombre de messages envoyés. EB est le nombre de comportements exécutés.

Selon Toufik Marir et al. (Marir & al., 2016), dans les deux formules précédentes on suppose que l'agent ne fait pas plus d'une requête pour chaque comportement exécuté pour assurer les exigences de la normalisation des valeurs dans l'intervalle [0,1].

4. Outil développé

Nous avons développé, au fil de ce travail, un environnement (composé de plusieurs outils) supportant la maintenance préventive conditionnelle des applications JADE. Baptisé PMMAS (Preventive Maintenance of Multi-Agent Systems), cet environnement est développé à l'aide de la Rich Client Platform (RCP) sous Eclipse.

Le principal avantage de cette plateforme est son architecture extensible qui permet l'intégration d'autres Plugins. Dans notre cas, nous avons intégré le JDT (Java Development Tools) et l'AJDT (AspectJ Development Tools). PMMAS offre toutes les fonctionnalités qui permettent à l'utilisateur de coder son programme (en utilisant JDT Plugin) et le rend sous contrôle continu (en utilisant AJDT Plugin). PMMAS est indépendant d'Eclipse IDE et peut être déployé en tant qu'application autonome ou en tant que plugin.

5. Étude de cas : Simulation d'un système de production automobile

Pour valider notre approche, nous avons utilisé un exemple concret de système multi-agents. L'exemple représente une simple simulation d'un système de production automobile. Dans cet exemple, il existe plusieurs unités de production des différents composants de la voiture avec une unité principale dont le rôle est l'assemblage de ces composants. Chaque unité de production dispose d'un stock de taille limitée. L'unité principale dispose de plusieurs stocks : un stock pour chaque composant et un stock pour le produit final (voitures).

L'unité principale essaie toujours d'assurer le bon fonctionnement du processus de production en remplissant les stocks des différents composants pour éviter les ruptures de stock, et en commercialisant le produit final en stock pour éviter la saturation. L'unité principale cherche également à obtenir les différents composants avec de meilleures offres en utilisant des mécanismes de négociation. Dans notre cas, nous utilisons le protocole Contract-Net pour gérer la négociation entre agents.

La figure 3.4 illustre un protocole d'interactions entre agents. Il décrit, à l'aide d'un diagramme de séquence AUML, le protocole Contract Net. Lorsqu'il est appelé, l'agent Initiator (CarProducer) envoie un appel à proposition à un agent Participant (ComponentProd). Avant une date limite donnée, l'agent participant peut soumettre à l'agent initiateur une proposition (proposer), refuser de soumettre une proposition (refuser), ou indiquer qu'il n'a pas compris (pas compris). La proposition formulée par l'agent participant peut être acceptée ou rejetée par l'agent initiateur. Lorsqu'il reçoit une acceptation de proposition (accepter-proposition), l'agent participant informe l'agent initiateur de l'exécution de la proposition. Cependant, s'il ne peut pas remplir son engagement, il en informe l'initiateur par un message d'annulation (échec).

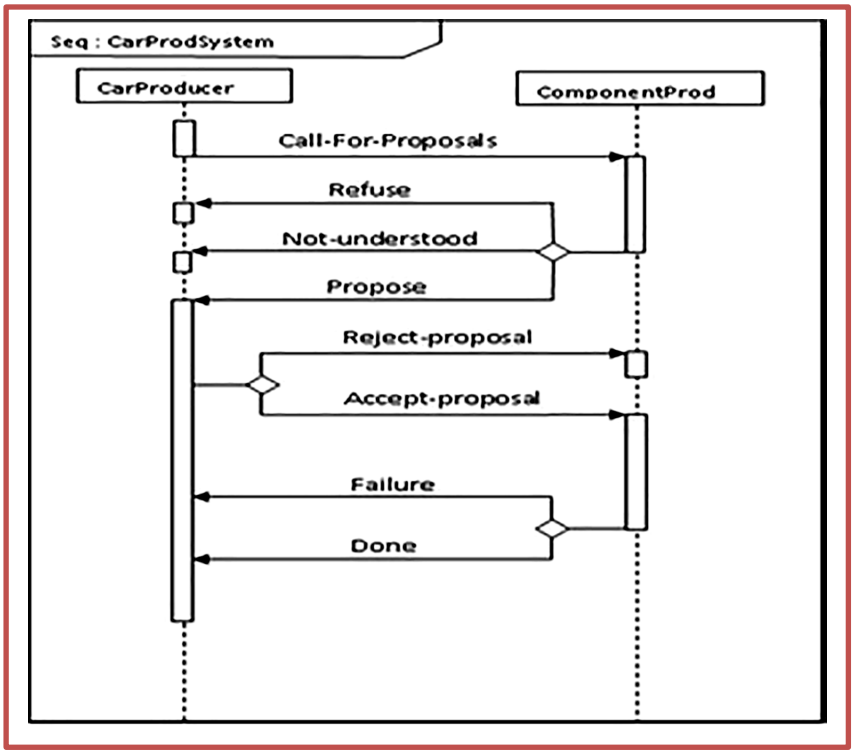


Figure 3.4. Le protocole Contract-Net.

L'interface utilisateur graphique de notre étude de cas présentée sur la figure 3.5 est subdivisée en deux sections, sur le côté droit se trouvent les différents rôles des agents, ainsi que deux boutons pour définir et lancer la simulation. Le côté gauche contient les instances de différents agents, ainsi que la quantité de stock de différents composants pour chaque unité de production.

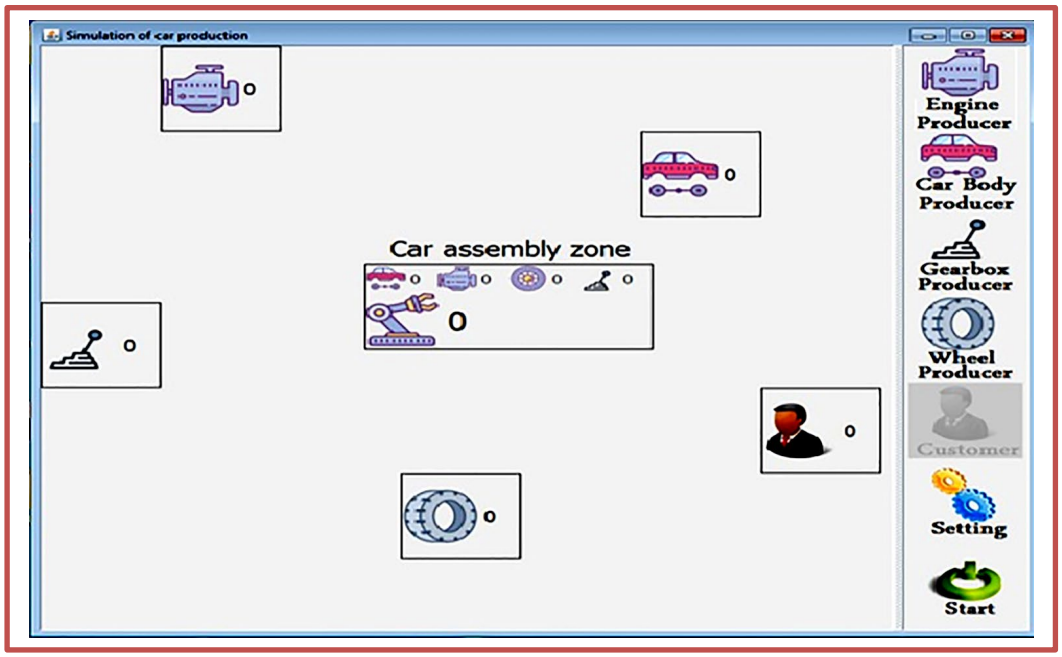


Figure 3.5. L'interface principale de PMMAS.

Une fois la simulation lancée, la fenêtre de réglage du seuil minimum apparaît (Figure 3.6).

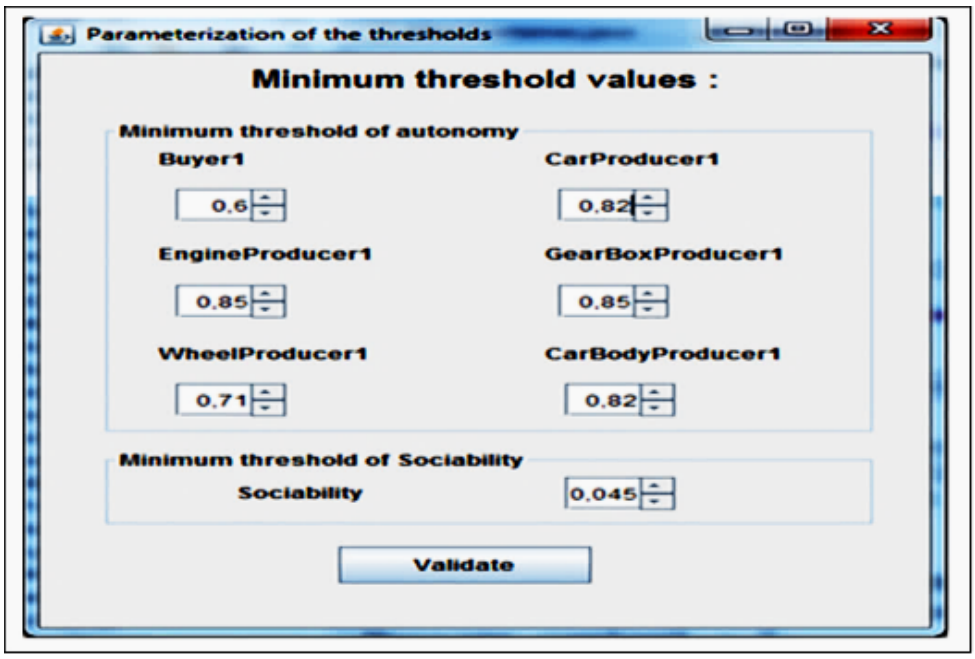


Figure 3.6. Fenêtre de configuration du seuil minimum.

Après une simple analyse statique de notre cas d'étude, nous avons constaté que le système est dans un état sain tant que les métriques mesurées sont supérieures aux seuils présentés dans la figure 3.6. De même, nous devons définir la taille des stocks des différents agents ainsi que la taille de la vague d'expédition (Figure 3.7).

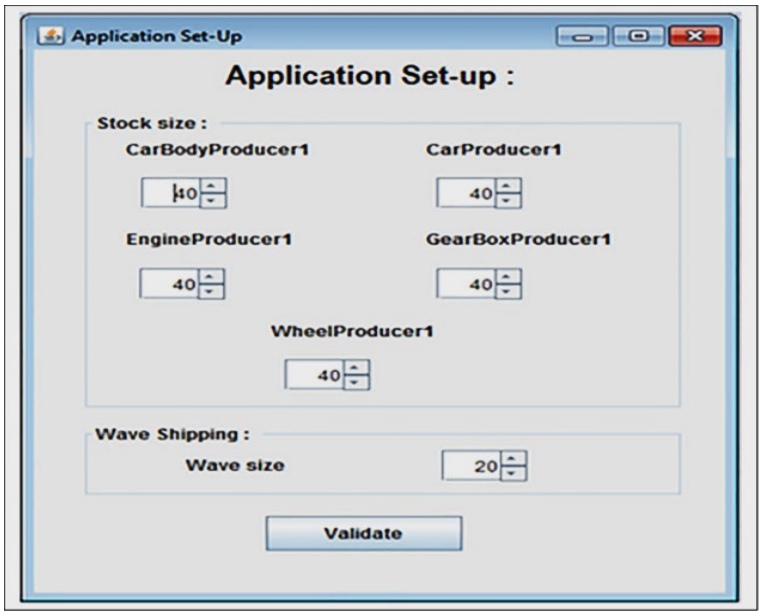


Figure 3.7. Taille de stocks des agents et taille d'envois de commande.

5.1. Scénario1 : Empêcher la régression de l'autonomie des agents

Pour démontrer l'efficacité de notre outil et donc de notre approche, nous avons utilisé un «test d'injection de faute » qui consiste à injecter la cause d'une erreur et à attendre l'apparition et la détection de cette erreur. Les causes d'erreur dans notre cas d'étude sont modélisées sous forme de diagrammes de cause-effet. Les diagrammes Cause-Effet (Figures 3.8 et 3.15) présentent quelques causes d'erreur de notre étude de cas relatives respectivement à l'autonomie et à la sociabilité des agents.

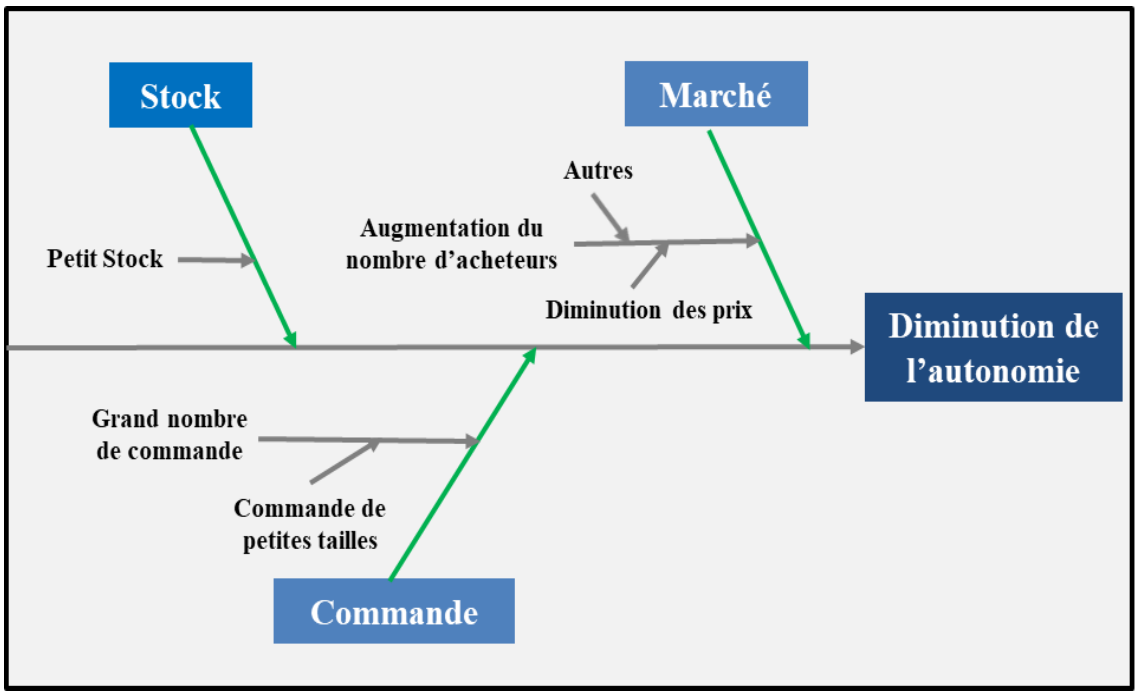


Figure 3.8. Diagramme Cause-Effet de diminution de l'autonomie.

Comme nous l'avons cité ci-dessus, nous utilisons le paradigme Aspect pour mesurer l'autonomie de l'agent. La portion de code présentée dans la figure 3.9. consiste à calculer le nombre de demandes de services (RS dans l'équation (1)) envoyées par l'agent CardProducer.

```
83⊖    @Pointcut("call(* metier..ProductionVoiture.send(..))")
84     public void CarProducerCall(){}
85
86⊖    @Before("CarProducerCall()")
87     public void BCarProducerCall(JoinPoint thisjoinPoint){ CarProducerCallN++; }
```

Figure 3.9. Code AspectJ pour le calcul de RS

La portion de code présentée dans la figure 3.10 consiste, d'une part, à calculer le nombre de comportements exécutés par l'agent CardProducer (EB dans l'équation (1) ainsi que la valeur de son autonomie, et consiste d'autre part, à l'affichage d'un message indiquant la régression de l'autonomie de CardProducer lorsqu'elle devient inférieure au seuil (0.82 voir Figure 3.6).

```
88
89⊖    @Pointcut("execution(* *.ProductionVoiture.*.*(..))")
90     public void CarProducerExecution(){}
91
92⊖    @Before("CarProducerExecution()")
93     public void BCarProducerExecution(JoinPoint thisjoinPoint) throws InterruptedException{
94
95         CarProducerExecutionN++;
96         CmptCarProducer++;
97
98         if(CmptCarProducer%60==0){
99         double a=CarProducerCallN,b=CarProducerExecutionN;
100        double Autonomie=1-((double)a/(double)b);
101        CarProducerExecutionN=0; CarProducerCallN=0;
102        CmptCarProducer=0;
103
104              IndiceCP++;
105              Template.vec5.add(new DataSetElement(Autonomie,"CarProducer1",""+IndiceCP));
106              Template.vec5.add(new DataSetElement(MinThreshold2,"Threshold",""+IndiceCP));
107              if(Autonomie<=MinThreshold2){
108                  Toolkit.getDefaultToolkit().beep();
109                  Dialogue.MessageInformatif("Car Producer Autonomy ("+Autonomie+") is less"
110                       + " than the Minimum Threshold");
111              }
112        }
113    }
```

Figure 3.10. Code AspectJ pour calculer l'EB et l'autonomie du CarProducer.

La figure 3.11 montre la courbe de l'autonomie mesurée en continu de l'agent CarProd en situation normale et son seuil.

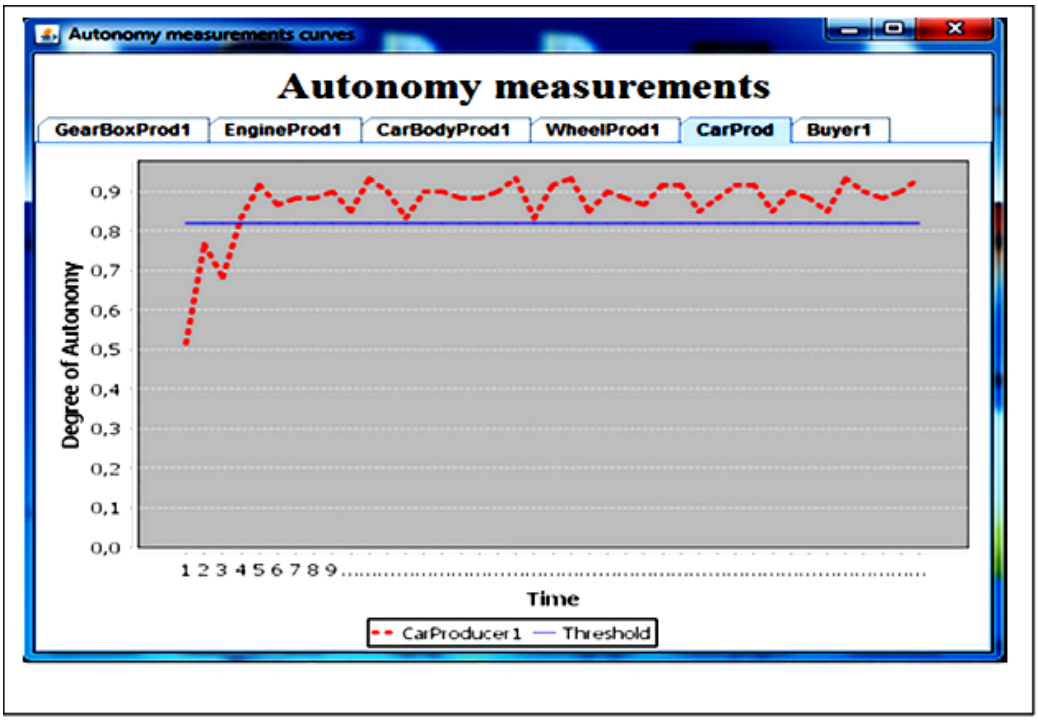


Figure 3.11. Les mesures d'autonomie de l'agent CarProd sans erreur.

Sur la figure 3.12, nous avons changé la taille de la vague d'expédition (de 20 à 2). Cette modification est l'une des causes de la diminution de l'autonomie comme présenté sur la Figure 3.13.

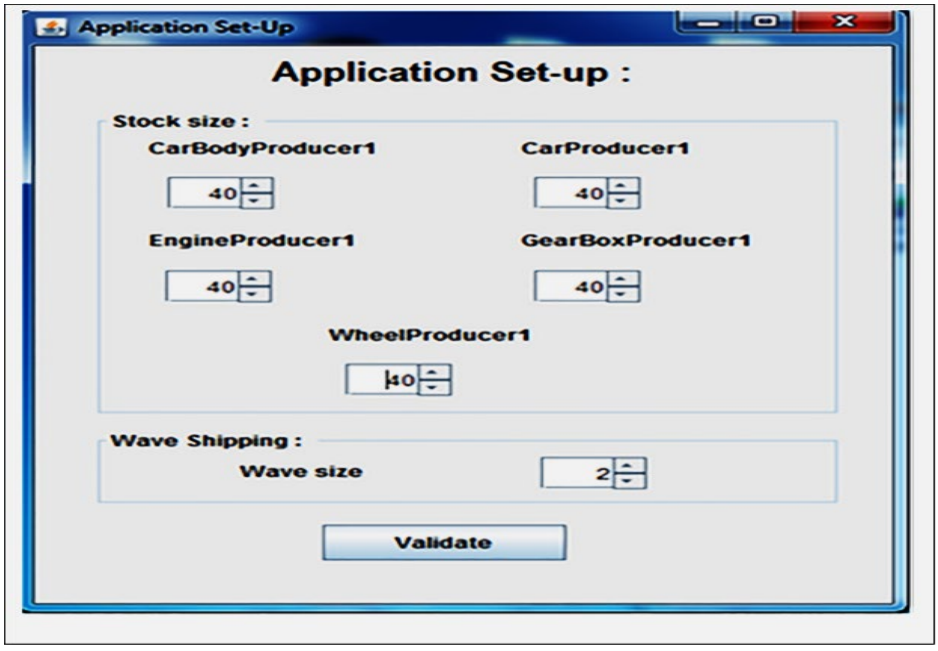


Figure 3.12. Changement de la taille des vagues d'expédition.

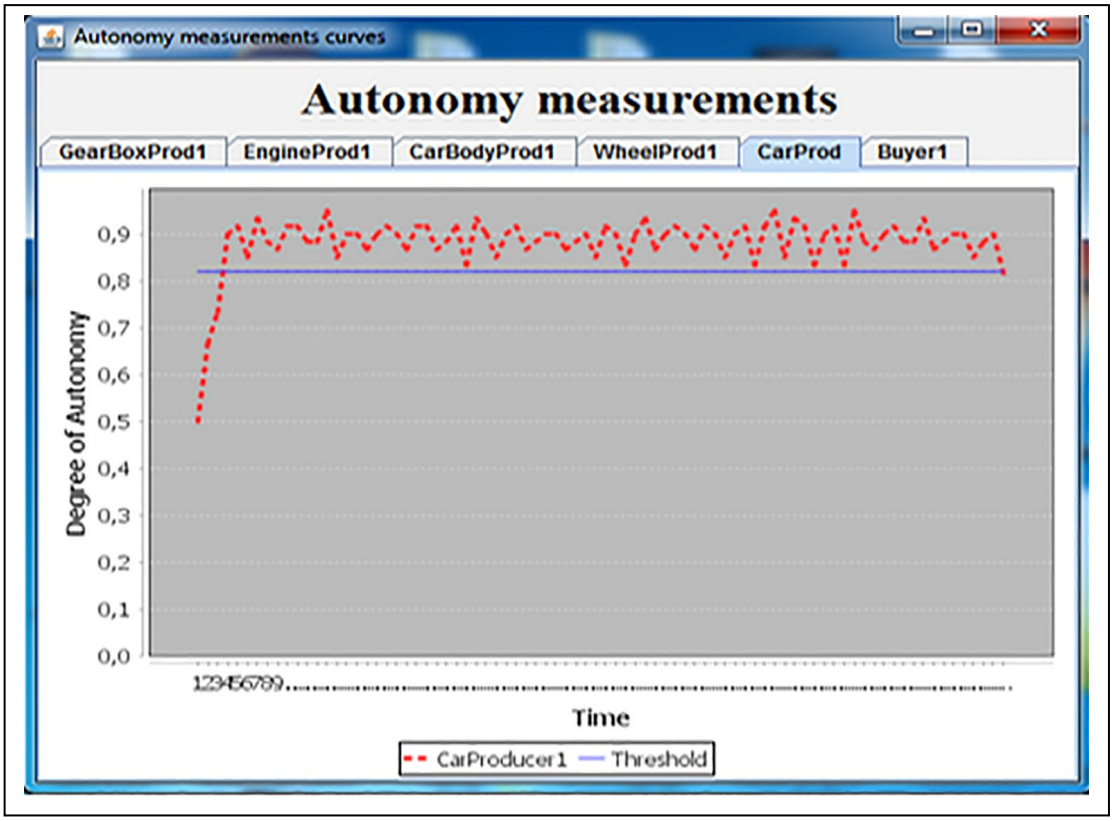


Figure 3.13. Mesure de l'autonomie de l'agent CarProd après injection de l'erreur.

Après avoir injecté la cause de l'erreur, l'autonomie de l'agent CarProd passera sous les seuils (0,82) définis précédemment par le concepteur (Figure 3.13) et une alerte préventive sera lancée (Figure 3.14).

La figure 3.14 présente l'alerte générée lorsque l'autonomie devient juste inférieure au seuil (0,82) précédemment défini par le concepteur (régression de l'autonomie CarProd à 0,8166).

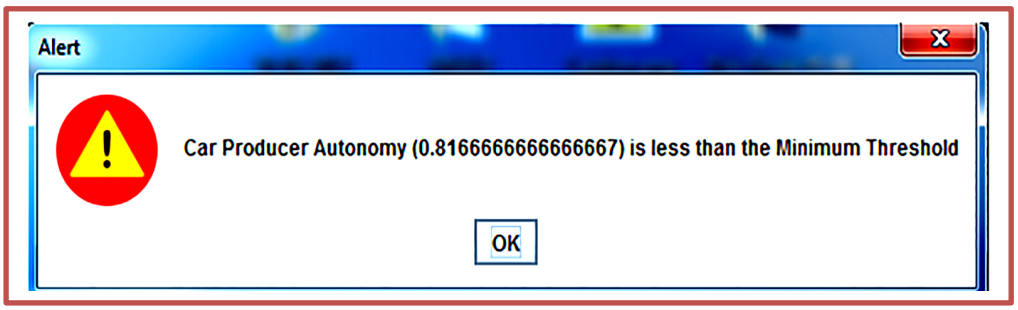


Figure 3.14. Avertissement de dégradation de l'autonomie.

Pour ramener le système à son état normal, nous devons modifier la configuration de l'application de notre étude de cas comme nous l'avons montré dans la Figure 3.12. Dans ce cas, nous avons changé la taille de la vague d'expédition à 50 comme présenté dans la Figure 3.15.

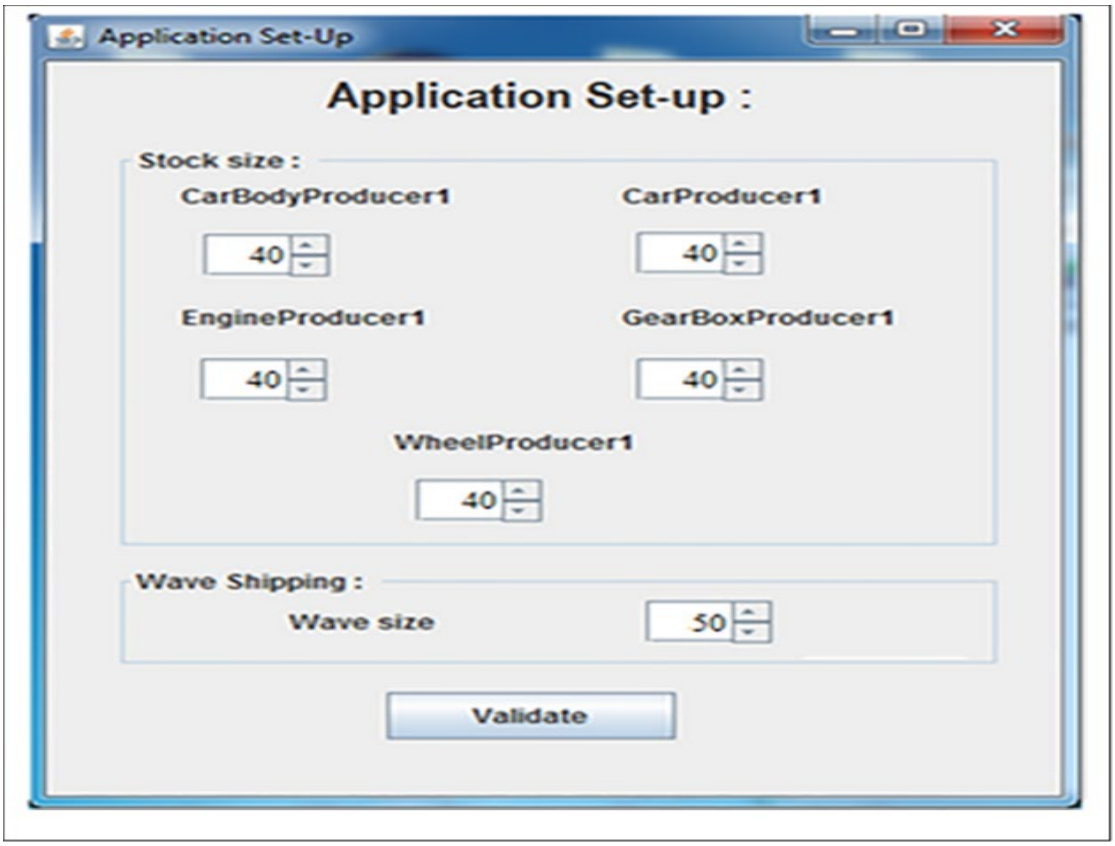


Figure 3.15. Augmentation de la taille de l'expédition des vagues d'expédition.

La figure 3.16 représente la courbe de l'autonomie mesurée de l'agent CarProd avant et après la modification (intervention).

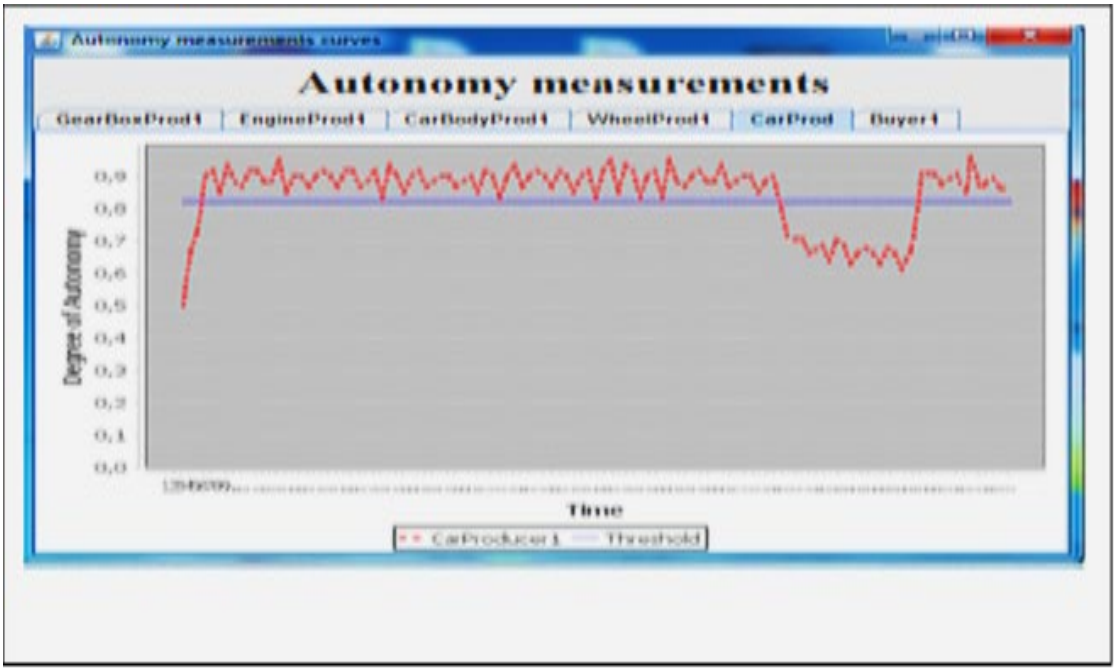


Figure 3.16. Mesures de l'autonomie de l'agent CarProd avec correction.

5.2. Scénario 2 : Prévenir la régression de la sociabilité

La figure 3.17 présente quelques causes de la diminution de la sociabilité entre les agents à l'aide d'un diagramme cause-effet.

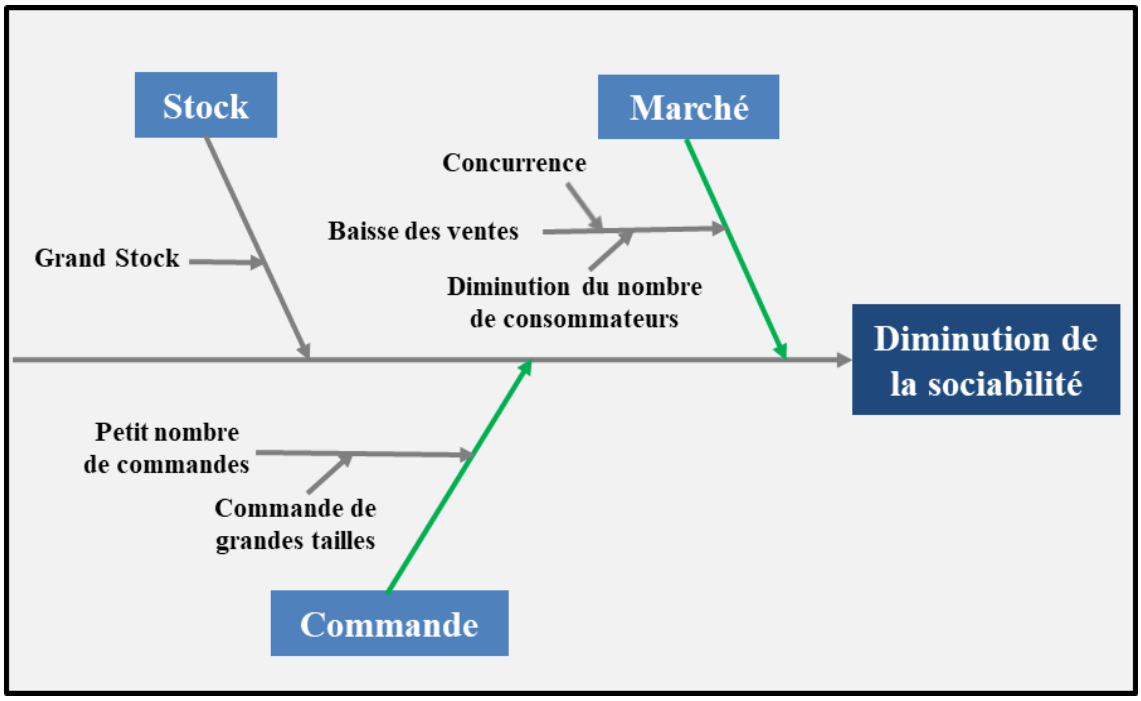


Figure 3.17. Diagramme Cause-Effet de diminution de la sociabilité

De la même manière effectuée avec l'autonomie de l'agent, la figure 3.18 présente la partie du code AspectJ qui consiste à calculer le nombre de messages (MN dans l'équation (2)) échangés entre les agents du système de production automobile.

La partie de code AspectJ présentée dans la figure 3.19 consiste à calculer le nombre de comportements effectués par tous les agents du système de production automobile. De plus, il calcule la valeur de la sociabilité et l'affiche dans l'interface graphique.

```
376  @Pointcut("call(* metier..WheelProducer.send(..)) || call(* metier..GearBoxProducer.send(..)) "
377          + "|| call(* metier..EngineProducer.send(..))|| call(* metier..CarBodyProducer.send(..))"
378          + "|| call(* metier..Buyer.send(..)) || call(* metier..ProductionVoiture.send(..))")
379  public void SociabilityCall(){}
380
381  @Before("SociabilityCall()")
382  public void Sociability(JoinPoint thisjoinPoint){  SociabilityCallN++;  }
```

Figure 3.18. Code AspectJ pour le calcul de MN.

```
387    @Pointcut("execution(* *.WheelProducer.*.*(..)) || execution(* *.GearBoxProducer.*.*(..))||"
388            + " execution(* *.EngineProducer.*.*(..))|| execution(* *.CarBodyProducer.*.*(..)) || "
389            + "execution(* *.Buyer.*.*(..)) || call(* metier..ProductionVoiture.send(..))")
390    public void SociabilityExecution(){}
391
392    @Before("SociabilityExecution()")
393    public void SociabilityExecution(JoinPoint thisJoinPoint) throws InterruptedException{
394        SociabilityExecN++;
395        CmptSociability++;
396        if(CmptSociability%60==0){
397         double a=SociabilityCallN,b=SociabilityExecN;
398         double Sociability=((double)a/(double)b);
399         SociabilityExecN=0; SociabilityCallN=0;
400         CmptSociability=0;
401
402           IndiceSoc++;
403           TemplateSoc.vec.add(new DataSetElement(Sociability,"Sociability",""+IndiceSoc));
404           TemplateSoc.vec.add(new DataSetElement(0.045,"Threshold",""+IndiceSoc));
405
406           if(Sociability<MinSociability){
407             Toolkit.getDefaultToolkit().beep();
408             Dialogue.MessageInforatif("Agents sociability ("+Sociability+") is less than "
409                   + "the Minimum Threshold");
410             }
411       }
412    }
```

Figure 3.19. Code AspectJ pour calculer l'EB et la sociabilité.

L'une des causes de la diminution de la sociabilité dans notre cas d'étude est l'absence de transactions d'achat qui conduit à la saturation des stocks et par conséquent à l'arrêt du processus de production. Sur la figure 3.20, nous avons arrêté l'agent qui simulait le rôle des acheteurs.

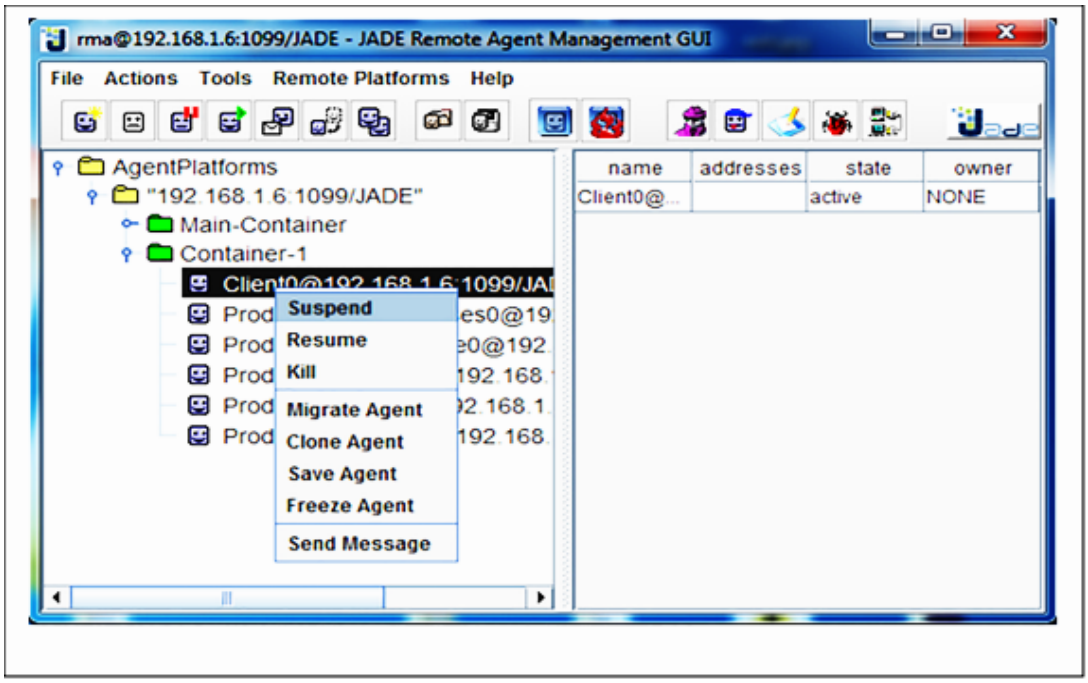


Figure 3.20. Suspension de l'agent acheteur.

La figure 3.21 montre la courbe de sociabilité des agents avant et après l'injection de la cause d'erreur.

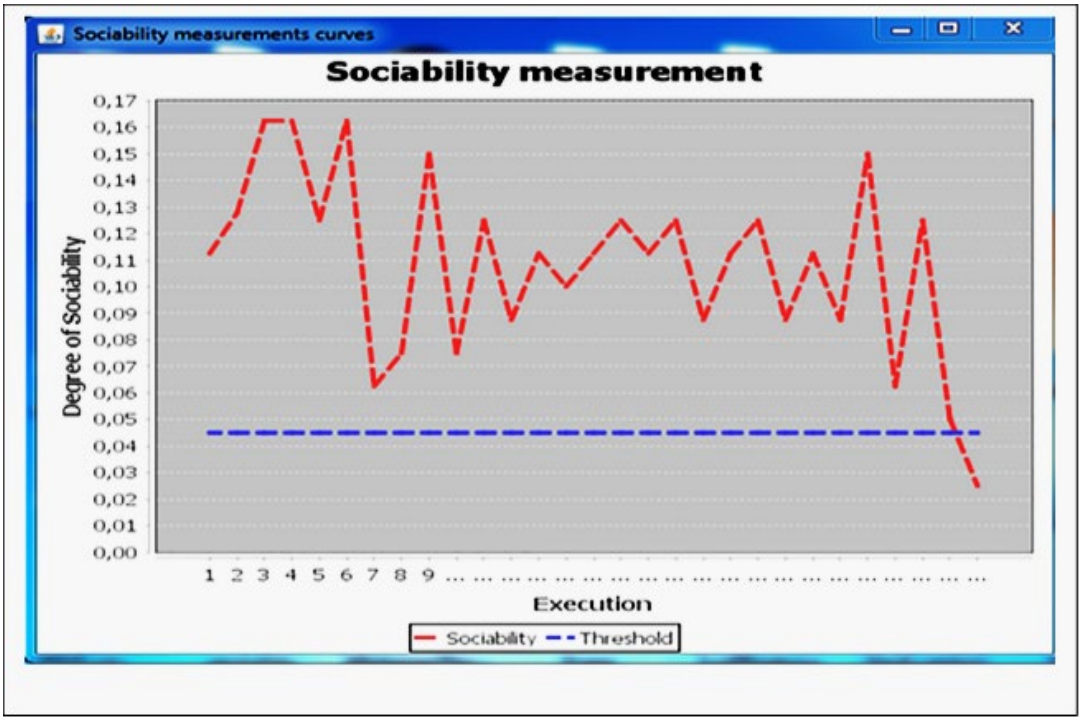


Figure 3.21. Mesures de la sociabilité après injection de l'erreur.

La figure 3.22 présente l'alerte générée lorsque la sociabilité est inférieure au seuil (0,045) précédemment défini par le concepteur (régression de la sociabilité à 0,025).

Pour ramener le système à son état normal, nous pouvons créer un nouveau client-agent, motiver les clients existants à augmenter leurs achats ou reprendre notre client-agent suspendu. Dans notre cas, nous avons créé un nouvel agent client, tel qu'il est présenté sur les figures 23-25. Le résultat de la correction est présenté sur la figure 26.

La figure 3.26 représente la courbe de la sociabilité des agents mesurés avant et après la modification (intervention).

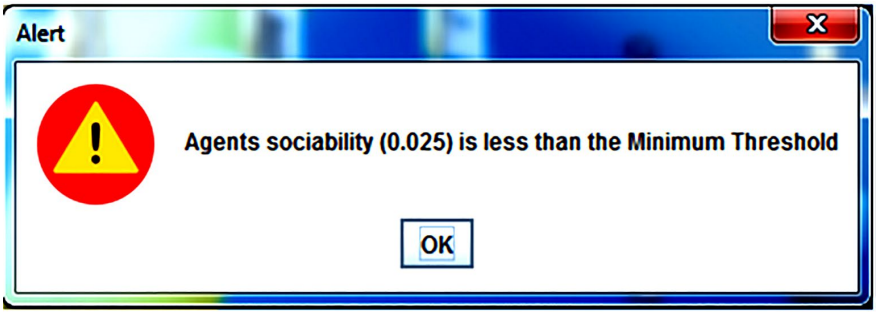


Figure 3.22. Avertissement de dégradation de la sociabilité.

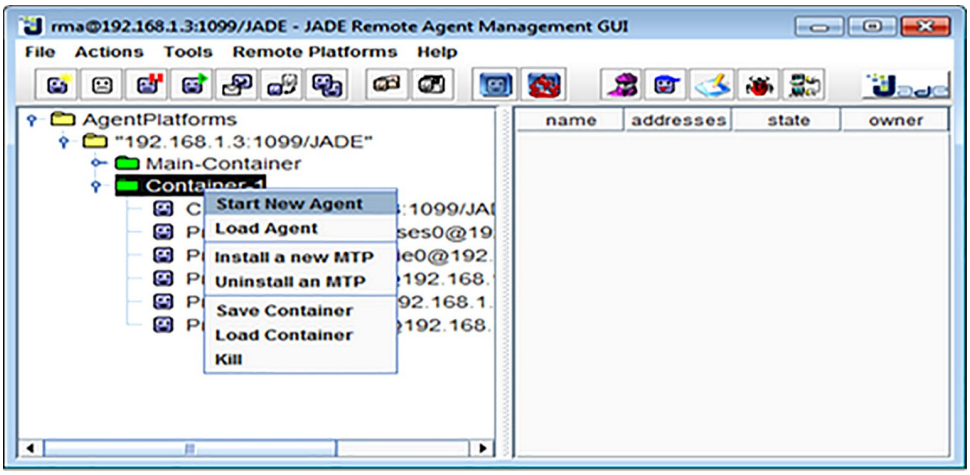


Figure 3.23. Création d'un nouvel agent.

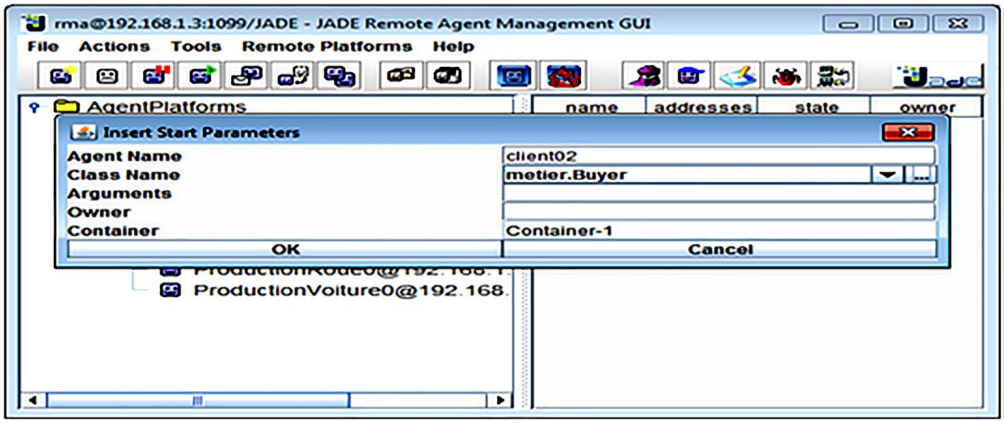


Figure 3.24. Lancement du nouvel agent.

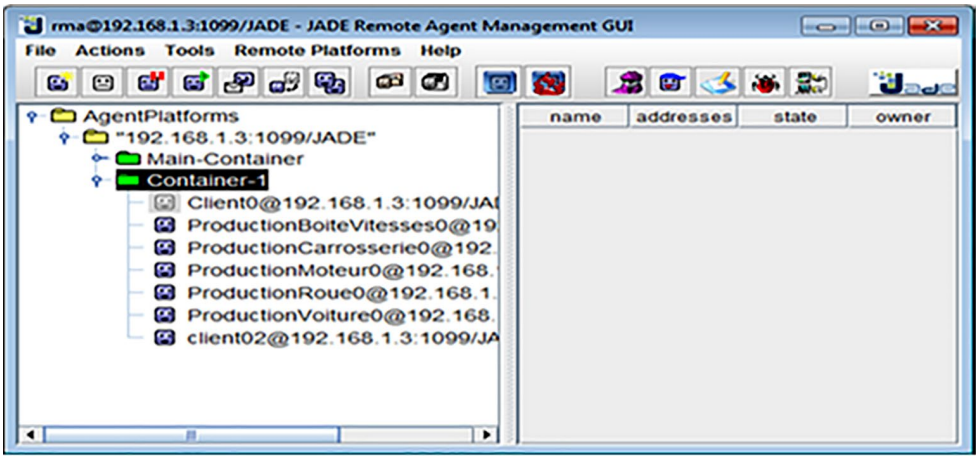


Figure 3.25. Le nouveau client apparaît dans le conteneur.

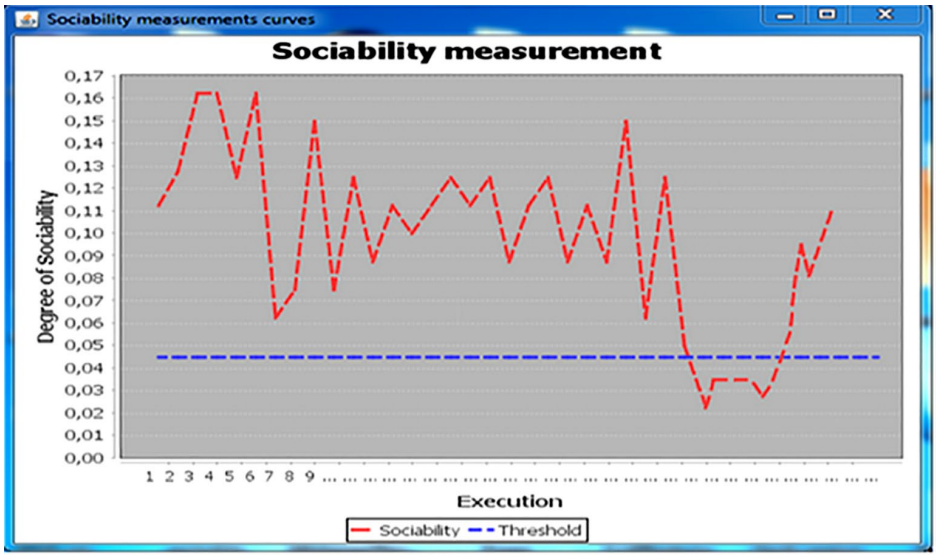


Figure 3.26. Mesures de la sociabilité avant et après la correction.

6. Discussion

Comme indiqué dans l'état de l'art au chapitre 2, quelques recherches ont été effectuées sur la maintenance préventive des logiciels. Ces recherches ont apporté des solutions intéressantes à différents problèmes et dans des contextes différents. Cependant, aucun d'entre elles ne traite la maintenance préventive des codes dans les systèmes multi-agents.

L'approche que nous avons proposée, dans ce chapitre, est la première étape vers la proposition d'une approche générique de maintenance préventive pour les applications multi-agents en prenant en compte tous les attributs de qualité des systèmes multi-agents définis dans le modèle qualité proposé dans (Marir & al., 2016).

Dans ce chapitre, nous n'avons pris en compte que deux attributs du système multi-agents : l'autonomie et la sociabilité car ils sont considérés comme les attributs les plus influents sur la qualité des multi-agents. Les résultats obtenus semblent satisfaisants et fiables. Cependant, la limite de notre approche est sa nature semi-automatique. Cette lacune est la principale cause de la perte de temps du système sous contrôle. Dans sa version actuelle, l'approche n'est pas bien adaptée aux applications multi-agents temps réel.

7. Conclusion

La maintenance préventive des systèmes multi-agents est un nouveau domaine de recherche qui n'a pas encore été exploré, et très rares sont les travaux qui ont portés sur la maintenance préventive des logiciels. Nous avons présenté, dans ce chapitre, une approche originale de la maintenance préventive conditionnelle SMA basée sur des mesures de qualité de ces systèmes et utilisant la Programmation Orientée Aspect.

Notre approche s'appuie sur un outil visuel appelé PMMAS (Preventive Maintenance of Multi-Agent Systems), qui a été validé sur une application JADE implémentant un système de production automobile.

Dans le chapitre suivant, nous utiliserons la réorganisation des systèmes multi-agents comme technique d'intervention afin d'éviter de futurs problèmes graves de l'application en maintenance. L'utilisation de la réorganisation pour la maintenance préventive du MAS est un moyen intéressant qui servira à améliorer l'efficacité du processus de maintenance.

Chapitre 4

Une approche de maintenance préventive pour les Systèmes Multi-Agents centrés organisation (OCMAS)

Sommaire

1. Introduction

L'un des principaux défis de l'ingénierie logicielle est de maintenir les systèmes logiciels, afin qu'ils continuent à être efficaces dans des situations changeantes. Notre recherche aborde cette question dans le contexte des systèmes OCMAS (Organization Centered Multi Agent Systems).

Dans ce chapitre, nous nous sommes concentrés sur la dotation du système OCMAS d'un système de maintenance qui effectue une adaptation autonome de son organisation afin de faire face à des évènements imprévus. Pour cela, nous proposons une nouvelle approche de maintenance préventive qui se base sur l'évaluation de la qualité pour la maintenance des systèmes OCMAS. La qualité du système OCMAS est surveillée afin de détecter toute régression anormale de la qualité du système ou celles des agents le composant. Cette dégradation de la qualité est en général une indication de problèmes pouvant affecter la structure de l'organisation ou ses fonctionnalités. Dès qu'une dégradation anormale est détectée, une intervention préventive devra être appliquée dans ce cas afin d'améliorer la qualité du système et lui permettre de reprendre son comportement normal.

Nous débutons ce chapitre par la présentation d'une approche de maintenance préventive générique pour les systèmes OCMAS. Ensuite, nous proposons quelques métriques pour évaluer la qualité des systèmes OCMAS en exécution et nous proposons des solutions pour rétablir la qualité de ces systèmes aux valeurs normales en cas de problèmes. Afin de valider l'approche et les métriques proposés, nous avons choisi de les appliquer dans le contexte des systèmes multi-agents se basant sur le modèle OMACS (Oraganizational Model for Adaptative Computational Systems) (Deloach & al., 2008).

Dans la suite du chapitre, nous appliquons notre approche de maintenance sur les systèmes basés OMACS. Pour cela, nous illustrons une formalisation du modèle organisationnel OMACS dans la section 4 puis nous présentons l'architecture globale du système proposée dans la section 5. Dans les sections 6 et 7 nous détaillons le processus et les algorithmes suivis pour la maintenance des systèmes OMACS. L'approche proposée est validée dans la section suivante à travers une étude de cas. Enfin, la section 9 clarifie l'apport de notre système de maintenance MAINOMACS.

2. Approche proposée

L'approche de maintenance préventive conditionnelle que nous proposons est basée sur des mesures de qualité des systèmes multi-agents organisationnels OCMAS et utilise une Programmation Orientée Aspect (POA). Notre approche est utilisée pour mesurer en continu les métriques de qualité de l'application et des agents la composant pour les comparer avec des seuils préalablement définis par le concepteur.

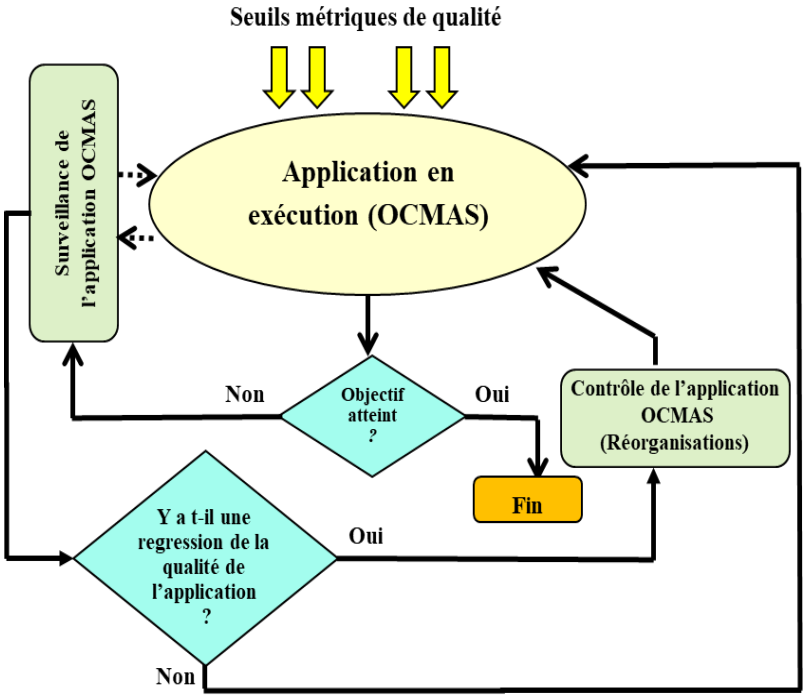


Figure 4.1. Méthodologie de l'approche de maintenance préventive des OCMAS (Ghrieb et al., 2021)

Lorsque les valeurs mesurées sont inférieures/supérieures aux seuils spécifiés, une intervention préventive doit être effectuée afin de rétablir la qualité du système et celle des agents aux valeurs normales préalablement définies. Cette intervention est réalisée par des techniques de réorganisation des SMA (Figure 4.1).

3. Métriques et solutions proposées

Plusieurs travaux se sont penchés sur l'exploration des comportements des systèmes multi-agents au niveau de la conception : théorie des graphes, métriques à la conception, etc. Ces travaux ont certes apporté une aide précieuse aux concepteurs de ces systèmes. Néanmoins la nature imprévisible de ces systèmes nécessite de se pencher sur leurs exécutions afin d'apporter de l'aide aux développeurs des SMA, qui se trouvent impuissant devant la nature des erreurs et la quantité énorme d'informations de ces systèmes à l'exécution.

L'objectif de cette partie est de proposer un ensemble de métriques permettant de détecter que des perturbations (anomalies) ont surgis dans le système OCMAS en exécution et qu'une intervention préventive est possible dans ce cas.

Sur la base de ces constatations et à partir de l'analyse des modèles de qualités des SMA (présentés dans le premier chapitre), nous avons proposé un ensemble de métriques afin d'apporter de l'aide aux développeurs lors de la maintenance des systèmes OCMAS. Ces métriques peuvent être utilisées pour mesurer, surveiller et analyser les performances

des systèmes OCMAS à l'exécution. Afin de faciliter cette intervention, des solutions utilisant des techniques de la réorganisation sont également proposées.

3.1. Métrique d'efficacité

Cette métrique permet de détecter les mauvaises assignations des agents aux rôles, telle que l'assignation des agents non compétents dans des rôles spécifiques qui ne leurs conviennent pas par exemple.

Suite à chaque assignation établie d'un agent *Ai* au rôle *Ri*, cette métrique calcule le taux de non efficacité de l'agent *Ai* au rôle *Rj* à un instant *t* appelé ***Fij (Ai/Rj)*** et permet de représenter le taux d'affectations non fructueuses (objectifs non atteints) de l'agent *Ai* au rôle *Rj* (Cette métrique sera détaillée davantage dans la partie 6 suivante).

Nous définissons *le taux local de non efficacité* ***Fij (Ai/Rj)*** par :

$$Fij\ (Ai/Rj) = (nAss\ (Ai/Rj)\ /tAss\ (Ai/Rj)) \quad \textbf{(1)}$$

Où

nAss (Ai/Rj) *:* Nombre des assignations non fructueuses de *Ai* au rôle *Rj*
tAss (Ai/Rj)*:* Nombre total des assignations de *Ai* au rôle *Rj*

Solution

Si *le taux de non efficacité* ***Fij (Ai/Rj)*** de l'agent *Ai* au rôle *Rj* à un instant *t* est supérieur au ***seuil de non efficacité locale*** défini *Se*, il est possible de réassigner le rôle de cet agent défaillant à un autre agent plus efficace dans ce rôle afin de remédier aux problèmes de l'organisation courante et d'éviter une éventuelle dégradation des performances du système.

3.2. Métrique Obtention ressource

Dans cette partie, nous entendons par demande de service toutes les demandes (requêtes) de fonctionnalités ou de ressources offertes par des entités jouant un rôle spécifique auprès d'autres entités.

Le facteur que nous proposons dans cette partie prend en compte les pannes d'obtention de services dans un intervalle de temps [t1, t2] et est activé si le taux d'échec d'appel à un service S est supérieur à *SeuilEchecService*. Pour détecter qu'une action préventive est nécessaire dans l'organisation, le nombre de demandes d'un service et le nombre d'échecs des demandes de ce service sont pris en compte.

Définissons *R (t1, t2, S)* le nombre de requêtes reçues par le service S dans la période de temps [t1, t2], et *ER (t1, t2, S)* le nombre d'échecs des requêtes au service S pendant la même période de temps. Le service demandé est considéré en échec lorsque l'objectif exécuté en réponse à cette demande de service est en état d'échec.

Le taux d'échec d'appel à un service S pendant une période de temps [t1, t2] est calculé comme suit :

$$\text{TauxEchecService}(S, t1, t2) = \begin{cases} ER(t1, t2, S) / R(t1, t2, S) & / R(t1, t2, S) <> 0 \\ 0 & / R(t1, t2, S) = 0 \end{cases}$$

Solution

Si le taux d'échec au service S est supérieur au seuil défini *SeuilEchecService*, il est possible d'intervenir et de résoudre le problème d'obtention de ce service (ou ressource) afin d'éviter une baisse des performances du système.

La solution proposée pour résoudre ce problème est de parvenir à un accord avec un agent externe *Ai* qui doit être en mesure d'apporter le service *S* (la ressource) à l'organisation. Si l'accord est atteint, l'agent *Ai* sera ajouté à l'organisation puis il sera assigné au rôle *Rj* afin d'offrir le service *S* à l'organisation.

3.3. Métrique Surcharge

Un groupe d'agents peut toujours choisir le même agent, cet agent devient alors surchargé avec les demandes continues tandis que les autres ne reçoivent aucune demande. Lorsque l'agent surchargé est occupé à prêter attention à une grande quantité de requêtes, son temps de réponse retarde et, par conséquent, le temps de réponse global du système retarde également et la qualité des services du système sera réduite.

Afin d'aider les développeurs du système OCMAS à garder une communication équilibrée au sein du système nous proposons une métrique qui permet de mesurer la communication dans les systèmes OCMAS et de détecter l'éminence et l'origine des communications déséquilibrées.

Cette métrique calcule le taux de Surcharge de l'agent dans un Rôle à un instant *t* appelé

SR (Ai/Rj) et représente la quantité de messages reçus par un agent *Ai* au rôle *Rj* par rapport à la moyenne de messages reçus par les agents jouant le même rôle *Rj*.

Nous définissons le *Taux de Surcharge* ***SR (Ai/Rj)*** par :

$$\textbf{\textit{SR (Ai/Rj)}} = \begin{cases} 0 & \text{Si } Ai \text{ n'a reçu aucun message dans le rôle } Rj \\ C(Sj) & \text{Si tous les messages sont reçus par le même agent } Ai \text{ dans le rôle } Rj \\ \dfrac{R(\text{Ai}, \text{Rj})}{\sum_{k=1}^{C(Sj)} R(Ak, \text{Rj}) / C(Sj)} & \text{Sinon} \end{cases}$$

Tels que :

· *A1...An* sont les agents.

· C (X) est la cardinalité de l'ensemble X.

· R (*Ai, Rj*) est le nombre de messages reçus de l'agent *Ai* dans le rôle R_j.

· *Sj* est le sous-ensemble d'agents jouant le même rôle *Rj*.

Ce taux ne doit pas dépasser un seuil prédéfini dit ***seuil de surcharge*** de l'Agent, noté ***SS.***

Dans le cas contraire, l'agent *Ai* se trouve dans une situation de perturbation et est considéré comme agent surchargé.

Solution

Si le Taux de Surcharge ***SR (Ai/Rj)*** d'un agent *Ai* au rôle *Rj* est supérieur au *SeuilSurchargeRôle*, il est possible d'appliquer l'une des solutions suivantes :

- réassigner le rôle de l'agent surchargé à un autre agent qui a moins de charge dans ce rôle.

- interdire toutes les assignations futures à l'agent surchargé, jusqu'à ce que son *Taux de Surcharge* revienne aux normes.

- Modifier la politique d'assignation des agents à ce rôle.

4. Organisation et réorganisation

Les modèles de société d'agents permettent de représenter les éléments qui composent la société ainsi que les interactions entre ces éléments. Afin de supporter les aspects organisationnels dans les systèmes multi-agents organisationnels (OCMAS), plusieurs modèles organisationnels ont été proposés dans la littérature. Bien que ces modèles aient apporté des éléments de réponse importants dans le développement des SMA organisationnels, dans ce travail nous avons choisi d'utiliser le modèle organisationnel OMACS (DeLoach & al., 2007), car nous l'avons trouvé suffisant pour les exigences de l'approche proposée.

Dans cette section nous introduisons le modèle organisationnel OMACS et nous présentons les différentes situations qui nécessitent réorganisation du système OMACS en exécution.

4.1. Modèle organisationnel OMACS

Scote DeLoach et al. (DeLoach, 2008) ont proposé un Framework pour le développement des systèmes complexes, capables de s'adapter avec leurs environnements d'exécution.

Ce Framework permet au système de construire sa propre organisation (se réorganiser) au cours de son exécution. Le composant clé de ce Framework est un modèle organisationnel, appelé OMACS (Oraganizationel Model for Adaptative Computational Systems) (Deloach & al., 2008).

Le modèle OMACS (Organizationel Model for Adaptative Computational Systems) (Deloach & al., 2008), constitue une base solide pour le développement des SMA organisationnels et adaptatifs. Ce modèle permet de représenter la structure et le fonctionnement de l'organisation et permet également de fournir les informations nécessaires pour que cette organisation puisse évaluer son comportement et mette en œuvre de façon autonome des mécanismes d'adaptation et de réorganisation si nécessaire.

Le modèle organisationnel OMACS a été inspiré du méta-modèle de la méthodologie MaSE (Mutiagent System Engineering) (DeLoach & al., 2001). Cette dernière est fondue sur le modèle AGR (Ferber et al., 2004). Une organisation selon le modèle OMACS est définie par le tuple O=⟨G, R, A, C, ϕ, P, Σ, achieves, requires, possesses, capable, potential , oaf ⟩ (Deloach & al., 2008) où :

- G est l'ensemble des objectifs que l'organisation poursuit
- R est l'ensemble des rôles que les agents de l'organisation jouent pour atteindre les objectifs
- A : est l'ensemble des agents qui font partie de l'organisation,
- C : est l'ensemble des capacités que les agents possèdent. Cet ansemble détermine si un agent a la capacité de jouer un rôle et, dans l'affirmative, dans quelle mesure il peut le jouer.
- ϕ : est l'ensemble des affectations courante <Agent, Rôle, But> décrivant le rôle joué par un agent particulier et le but poursuivi par ce rôle.
- P : définit les politiques de l'organisation qui concernent les affectations, le comportement des agents et la réorganisation.
- Σ : est le modèle de domaine qui permet la définition des types d'objets dans l'environnement de l'organisation et les relations entre ces types.
- A*chieves*: G × R → [0..1], est une fonction qui estime l'efficacité d'un rôle R dans la réalisation d'un but G.
- *Capable* : A × R → [0..1], elle estime le rendement d'un agent A lorsqu'il joue un rôle R.

- *Requires* : R → P(C), est une fonction qui spécifie l'ensemble des capacités requises par un agent pour jouer un rôle
- *Possesses* : A × C → [0..1], est une fonction qui définit la qualité de la capacité d'un agent
- *Potential* : A × R × G → [0..1], est une fonction qui permet d'estimer l'efficacité d'un agent A à jouer un rôle R afin de réaliser un but G.
- *oaf (*Organization Assignement Function) : ϕ → [0..∞], est une fonction définissant la qualité des affectations en Φ

4.2. Déclencheurs de la réorganisation

Divers événements peuvent se produire dans le cycle de vie d'un système multi-agents OMACS, ce qui nécessite une réorganisation du système. En général, la réorganisation est lancée lorsqu'un événement se produit et modifie l'état de l'organisation actuelle. Comme nous étudions actuellement uniquement les changements fondés sur le comportement (état), nous avons actuellement identifié deux types d'événements : les changements des objectifs du système et les changements des objectifs de la maintenance. Chacun de ces évènements, peut entraîner une réorganisation du système par le changement de l'assignation des agents aux rôles afin de réaliser les objectifs du système. Nous discutons ces situations en détail ci-dessous.

• Changement des objectifs du système qui sont pris en compte par le système de contrôle (OMACS) afin d'assurer l'adaptation du système aux différentes situations de changement dans les objectifs suivants :

- Instanciation de l'objectif
- Achèvement de l'objectif
- Echec de l'objectif

• Échec de l'objectif de maintenance, pris en compte par notre système de maintenance

4.2.1. Changement des objectifs du système

- Instanciation de l'objectif

Lorsqu'un événement se produit et déclenche l'instanciation d'un nouvel objectif, cet objectif est entré dans l'ensemble d'objectifs de l'organisation. L'insertion d'un nouvel objectif nécessite que le système prenne des mesures pour satisfaire ce nouvel objectif. Si l'organisation est en mesure de trouver une paire agent-rôle capable d'atteindre l'objectif, l'agent est affecté à jouer ce rôle afin d'atteindre l'objectif.

-Achèvement de l'objectif.

Lorsqu'un objectif *g* est atteint, l'ensemble des objectifs actifs du système (G) est modifié pour refléter cet événement en (1) supprimant g de G et (2) ajoutant éventuellement de nouveaux objectifs, qui sont activés par la réalisation de g, dans G. Évidemment, l'agent affecté à atteindre l'objectif g est désormais libre de poursuivre d'autres objectifs.

- Echec de l'objectif

Lorsqu'un agent spécifique ne peut pas atteindre l'objectif g mais que g peut toujours être atteint par un autre agent, l'échec de l'objectif de l'agent se produit. En cas d'échec de l'objectif de l'agent, une réorganisation doit avoir lieu pour permettre à l'organisation de choisir un autre agent pour atteindre g.

4.2.2. Échec de l'objectif de maintenance

Pour s'adapter à une variété de situations imprévisibles, notre système de maintenance est capable de détecter plusieurs types de changements dans l'organisation du SMA en exécution grâce à son système de monitoring et permet en conséquence d'effectuer des modifications là-dessus. Un grand nombre de ces changements sont des changements dans l'environnement ; cependant, certains changements se produisent au sein de l'organisation (par exemple, une défaillance de capacité d'un agent). De tels changements deviennent des déclencheurs de réorganisation lorsqu'ils empêchent l'organisation d'atteindre son objectif global dans les délais / précision de donnés ou empêchent le système d'être plus efficace pour atteindre son objectif.

Dans le cadre de notre travail de maintenance préventive nous nous intéressons en particulier à la maintenance du système par la surveillance de certains critères de qualité correspondants au système en exécution afin de prévoir toute défaillance pouvant apparaitre. L'idée de notre système de maintenance consiste donc à garder le niveau de qualité du système et des agents au-delà de certains seuils définis par le concepteur. Lorsque la qualité du système se dégrade en dessous du seuil défini, nous appelons cela un échec de l'objectif de maintenance. Dans une telle situation, la réorganisation du système est nécessaire afin de rétablir sa qualité au niveau défini.

Dans les sections suivantes, nous allons utiliser *la métrique d'efficacité* (proposée dans la section 3.1) comme critère de qualité afin d'analyser le système en exécution et de détecter toute défaillance pouvant conduire à sa défaillance.

L'échec de l'objectif de maintenance est détecté dans notre système par :

- La détection de la dégradation de l'efficacité du système en exécution.
- La détection de la dégradation de l'efficacité des agents composants le système.

Comme notre objectif de maintenance concerne les problèmes fonctionnels pouvant affecter le système, nous avons alors identifié quatre scénarios pouvant conduire à la dégradation de l'efficacité du système (échec de l'objectif de maintenance) :

1. Lorsque les agents renoncent aux rôles requis.
2. Lorsque les agents subissent des défaillances qui les empêchent de remplir leurs rôles.
3. Lorsque les agents ne sont pas compétents pour remplir les rôles qui leurs sont attribués.
4. Lorsque les capacités des agents ne leurs permettent pas de donner des résultats de bonne qualité.

Dans les sections qui suivent, nous décrirons davantage ces scénarios ainsi que le mécanisme de maintenance nécessaire pour faire face à ce genre de situations dans les systèmes OMACS.

5. Architecture du système proposée

Le système de maintenance que nous proposons est nommé MAINOMACS (pour : MAINtenance of multi-agent system based OMACS). Ce système fournit un support pour la maintenance préventive des systèmes multi-agents se basant sur le modèle organisationnel OMACS.

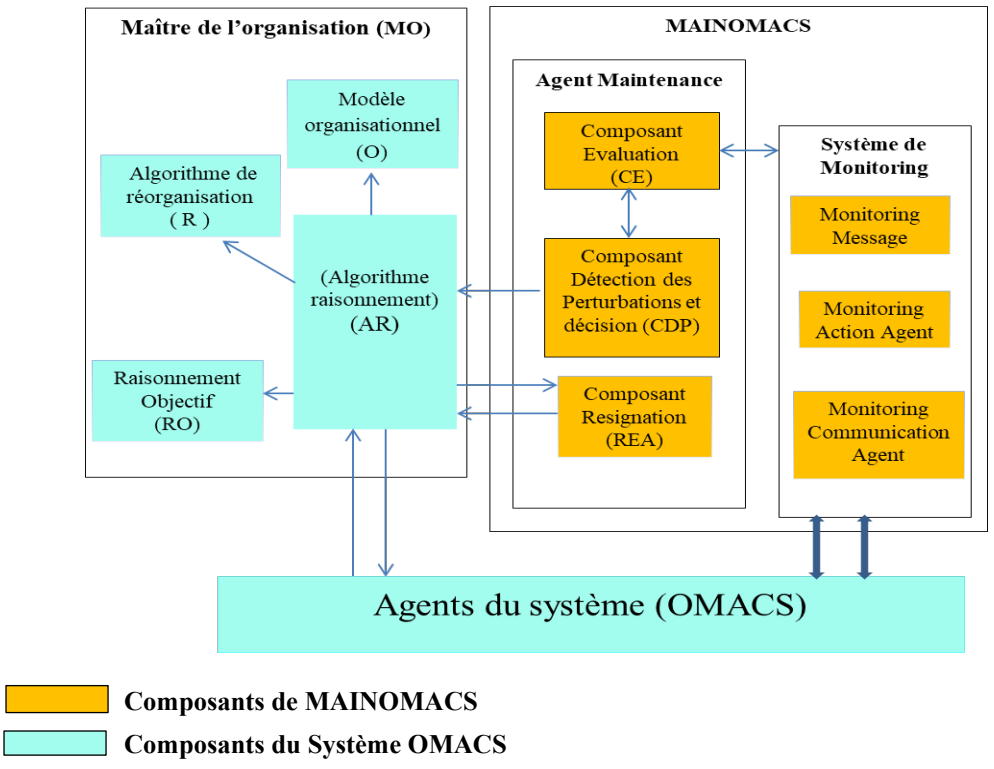


Figure 4.2. Architecture globale du système proposée (Ghrieb et al., 2021)

L'architecture globale du système présentée sur la figure 4.2 est constituée du : Système Multi-Agents basé OMACS (système OMACS) et le système de Maintenance (MAINOMACS).

Le contrôle dans le système OMACS est réalisé par le *Maitre de l'Organisation (MO)* qui effectue tous les raisonnements au niveau de l'organisation afin de permettre au SMA en exécution de s'adapter aux différentes situations et d'atteindre l'état cible désiré. Ces situations peuvent se produire, par exemple, lorsque l'environnement a changé, le but du MAS a changé, une nouvelle demande de tâche arrive et l'organisation actuelle n'est pas appropriée, etc. Il est constitué des composants de base suivants : *Raisonnement Objectif (RO), Modèle Organisationnel (O), Algorithme Réorganisation (R) et Algorithme Raisonnement (AR).* Le composant *Raisonnement Objectif (RO)* met à jour l'arborescence d'instances d'objectif. Le Composant *Modèle Organisationnel (O)* garde la trace des objectifs actifs et des affectations courantes des agents aux différents rôles. *L'Algorithme Réorganisation (R)* détermine le nouvel ensemble d'affectations des agents aux rôles. *L'Algorithme Raisonnement (AR)* lie les trois autres composants ensemble. Il reçoit les différents évènements des agents de l'application en exécution, met à jour les affectations des agents aux rôles pour adapter l'organisation des agents aux nouvelles situations (différents évènements) et envoie les mises à jour d'affectation aux agents de l'application pour exécution.

Le système de maintenance MAINOMACS permet d'assurer une maintenance préventive conditionnelle des systèmes OMACS. Ce système de maintenance évalue le système OMACS en permanence afin de remédier à tous les problèmes qui peuvent affecter son organisation et l'empêcher d'atteindre son but. Ce système est composé d'un *Système de Monitoring* et d'un *Agent de Maintenance.* L'*Agent de Maintenance* est composé à son tour d'un *Composant d'Evaluation (CE), d'un Composant Détection des Perturbations (CDP) et d'un Composant de Réassignation (CR).* Le *Système de Monitoring* récupère les informations pertinentes sur le SMA. Le composant d'évaluation *CE* évalue l'efficacité de l'organisation et celle des agents. Le *Composant Détection des Perturbations CDP* utilise ces évaluations afin d'identifier les situations dans lesquelles les exigences de qualité ne sont pas satisfaites, les agents ne sont pas capables de bien jouer leur rôle et l'organisation actuelle ne satisfait plus les besoins du SMA en exécution. Le *Composant Réassignation CR* effectue une réallocation des agents afin de remplacer les agents qui n'assument pas leurs rôles convenablement. Enfin, le *Maitre de l'Organisation MO* utilise le *Composant de Réassignation CR* pour la réorganisation du système OMACS, afin d'améliorer son efficacité et lui permettre d'atteindre son objectif en toute sécurité

Dans ce qui suit, nous présentons un aperçu du Système OMACS, puis nous décrivons notre système de maintenance MAINOMACS.

5.1 Système OMACS

Le système OMACS (Deloach & al., 2008) est conçue selon une approche centralisée (Figure 4.3). Cette approche permet la réutilisation de divers algorithmes de

raisonnement organisationnel en découplant la partie raisonnement de l'organisation (qui peut être générique) du système réel composé des agents de l'application.

Le *Maitre de l'Organisation (MO)* est le seul agent possédant une connaissance complète de l'organisation et qui est capable d'exécuter les algorithmes de réorganisation. Le *MO* utilise ses connaissances des objectifs et des agents actuels, effectue les affectations appropriées et envoie les affectations aux agents. Le *MO* reçoit également des événements à propos de l'exécution des objectifs de chaque agent et se réorganise de manière appropriée en cas de besoin.

Dans cette approche centralisée que nous adoptons, les agents applicatifs reçoivent des missions du *MO,* jouent les rôles qui leur sont assignés et rendent compte de leur statut au *MO.* La communication entre le MO et les agents se fait par l'envoi de messages.

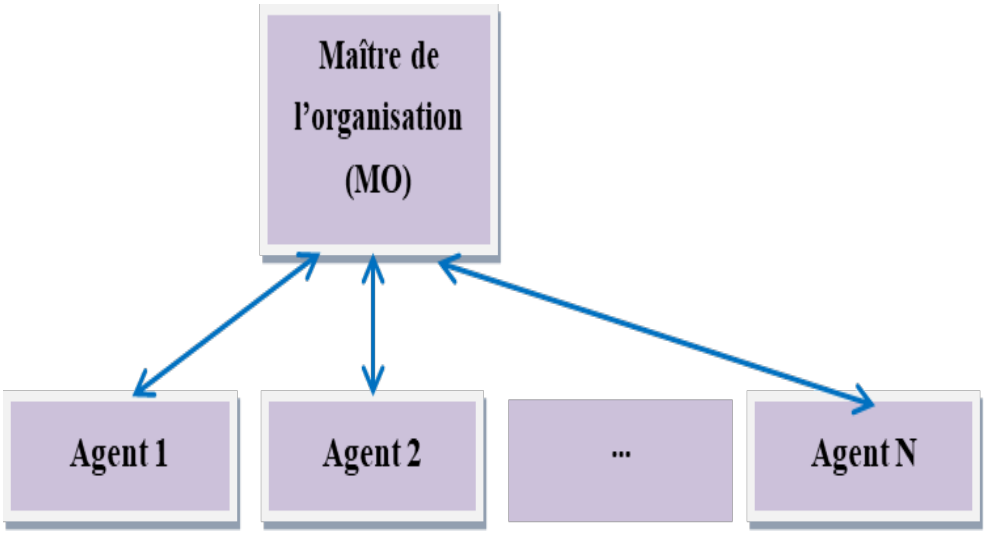


Figure 4.3. Architecture centralisée du système OMACS

L'architecture présentée ci-dessus est une approche centralisée dans laquelle, tous les composants de contrôle sont regroupés dans l'agent MO (Organisateur) qui gère l'organisation et assure son bon fonctionnement. Les autres agents de l'organisation sont constitués uniquement de composant d'exécution, recevant de l'organisateur leurs affectations (Agent, Rôle, But). Par contre, une architecture décentralisée dérivée de celle-ci est possible. Dans une telle architecture, les composants de contrôle des différents agents de l'organisation travaillent ensemble pour garder la cohérence de l'organisation en constituant un contrôle organisationnel distribué.

5.2. Système de maintenance MAINOMACS

Le système de maintenance MAINOMACS évalue en permanence le système OMACS afin de remédier à tout problème pouvant affecter son organisation et l'empêcher d'atteindre son objectif. Ce système est composé d'un *Système de Monitoring* et d'un *Agent Maintenance*.

5.2.1. Système de Monitoring

Le moniteur s'exécute en tierce partie et consiste à récolter les informations en utilisant l'Aspect Engine qui intercepte les évènements pertinents de la plateforme agent : Monitoring de la communication, Monitoring des messages, Monitoring des actions des agents afin de signaler tous les évènements à *l'Agent de Maintenance***.** Dans le cadre de ce travail, de maintenance préventive des systèmes multi-agents organisationnels (OMACS), nous nous sommes intéressés en particulier aux évènements entrainant la régression de l'efficacité du système. Ces évènements sont surveillés par notre *Système de Monitoring* et seront utilisées par *l'Agent Maintenance* pour la réorganisation du système afin de rétablir sa valeur d'efficacité au niveau adéquat et de prévenir le système ainsi d'une éventuelle défaillance.

5.2.2. Agent Maintenance

L'agent maintenance (figure 4.4) dans notre système **MAINOMACS** est conçu pour assurer la maintenance préventive du système OMACS, il assure 3 fonctions :

- ***Perception***

  L'agent de maintenance guette les perturbations du système à partir de la trace d'exécution établie par le système de monitoring. Ce composant est continuellement à l'écoute et permet à l'agent d'analyser le résultat de l'exécution des rôles par les agents et de garder trace des perturbations rencontrées à son niveau.

- ***Décision***

  Il évalue ces perturbations et décide à quel moment faudra-t-il lancer le processus de réorganisation. La décision dépend du taux global de perturbations, quand le degré d'efficacité dans le système descend au-dessous d'un certain seuil, le système est alors jugé en besoin d'adaptation globale.

- ***Exécution***

  Il applique un algorithme de réorganisation qui agit sur les assignations des agents aux différents rôles, dans le but d'améliorer à chaque pas de l'algorithme l'attribution des agents aux rôles. Nous présenterons en détails le fonctionnement de cet agent dans la section suivante.

Le modèle de l'agent maintenance est présenté ci-dessous :

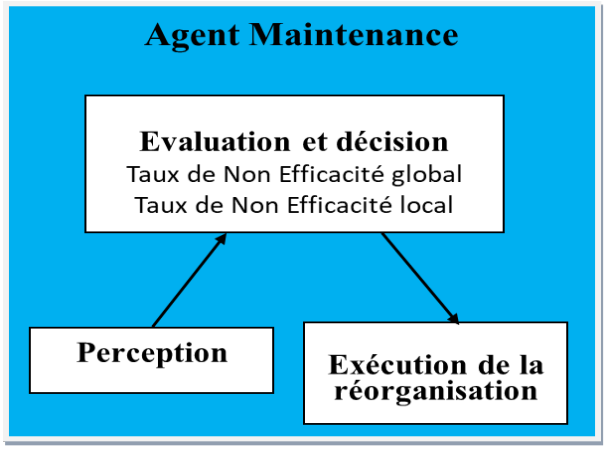


Figure 4.4. Modèle de l'agent maintenance (Ghrieb et al., 2021)

La réorganisation appliquée actuellement par notre système de maintenance MAINOMACS, est réalisée par une stratégie de réassignation des agents aux rôles afin de remédier aux problèmes de l'organisation actuelle. Néanmoins l'architecture de notre système de maintenance permet l'extension, par l'ajout d'autres stratégies de réorganisation dans l'agent de maintenance afin de traiter d'autres problèmes de l'organisation.

6. Processus de maintenance préventive se basant sur l'efficacité des agents

L'approche de maintenance des SMA organisationnelle que nous proposons peut se récapituler sommairement aux actions « Surveiller – Evaluer – Détecter- Contrôler ». A l'image de cette démarche de résolution, le processus de maintenance que nous proposons s'articule autour de quatre modules :

- Module de surveillance

- Module d'évaluation

- Module de détection des perturbations et décision

- Module de contrôle

6.1. Surveillance (Monitoring)

L'objectif du monitoring est d'étendre le système à maintenir, par un aspect afin d'obtenir les informations nécessaires pour son évaluation. Notre système de monitoring va observer d'une manière permanente le comportement de l'application et utilise les techniques de la Programmation Orienté Aspects (POA) afin de collecter les informations désirées.

Le système de monitoring est composé d'un aspect qui est responsable du :

- Monitoring des messages

- Monitoring de la communication des agents

- Monitoring des actions des agents

Les informations collectées durant le monitoring sont utilisées dans l'étape suivante du processus de maintenance pour l'évaluation de la qualité du système et des agents le composant.

6.2. Evaluation

La capacité d'évaluation permet d'examiner et d'analyser la qualité du système OMACS. Cette évaluation de qualité est utilisée par la suite pour éliminer les alternatives de contrôle qui peuvent conduire à des instabilités dans le système.

6.2.1. Indicateurs de qualité

De toute évidence, il est très difficile de dissocier une mesure de qualité du contexte de son utilisation. En effet, l'interprétation d'une mesure est toujours liée au système en question et est spécifique à son fonctionnement. De plus, une analyse réussie nécessite la connaissance profonde et la compréhension parfaite du système évalué.

Les critères de qualité à évaluer par notre système de maintenance dépendent en général de l'objectif du système à maintenir. Le choix de ces critères peut donc être laissé au mainteneur du système.

Cependant, dans le cadre de ce travail nous avons volontairement tenté, malgré cette difficulté, de dégager des indications de qualité qui dépendent du contexte du système à maintenir :

La réactivité

Dans les systèmes temps réel le facteur temps est primordial, donc il faut enlever dans ce cas tous les agents ayant une réactivité inférieure à un certain seuil.

La réactivité de l'agent représente la capacité de l'agent à percevoir son environnement et à générer des réponses dans le temps adéquat.

Afin de calculer la réactivité d'un agent, on peut donc utiliser la métrique Temps moyen de réalisation des buts qui est définie comme suit :

$$\textit{Temps moyen de réalisation des buts} = \frac{Temps\ total\ de\ réalisation\ des\ buts\ de\ l'agent}{Nombre\ de\ but\ réalisés}$$

L'efficacité

Dans les systèmes de production par exemple, il faudra s'intéresser à l'efficacité des agents, qui peut être calculée par : Nombre moyens de produits servis dans l'unité de temps par exemple.

Agrégation de plusieurs critères

La qualité des agents peut aussi être évaluée par l'agrégation de plusieurs critères de qualité tout en associant un poids à chaque critère selon l'importance qu'il représente dans le système :

$$Qualité\ de\ l'agent\ i = \sum_{N=1}^{M}(PoidsN * TauxMétriqueNi) \qquad /M : nombre\ de\ critères$$

Tel que, la valeur de qualité de chaque agent *Qualité de l'agent i* est normalisée entre 0 et 1, car :

- La valeur du *TauxMétriqueNi* = valeur de la MétriqueN de l'agent i / valeur de la MétriqueN du système global (entre 0 et 1)

- Le $PoidsN$ est toujours entre 0 et 1 et $\sum_{N=1}^{M}(PoidsN)=1$

Par exemple, si nous considérons dans notre système de maintenance que les agents pouvant améliorer l'efficacité d'un système sont en général des agents qui :

- répondent dans des temps courts (réactifs),

- acceptent souvent les demandes de services reçues de la part des autres agents (sociables)

- et qui exécutent leurs tâches sans l'aide des autres (autonomes).

Ces spécifications de qualités peuvent être utilisées pour l'évaluation de la qualité des agents par la formule de calcul suivante :

*Qualité de l'agent= 0.3*réactivité+0.3*sociabilité+0.4*autonomie*

Ainsi l'agent ayant une qualité inférieure à la valeur tolérée par le système sera considéré comme agent défaillant ayant causé la dégradation de la qualité du système. Dans ce cas, une réorganisation doit donc avoir lieu pour permettre à l'organisation de choisir un autre agent ayant une qualité meilleure. Le remplacement de tous les agents considérés défaillants permet ainsi de maintenir le système et améliorer son efficacité globale. A noter que notre système actuel, ne prend pas en considération le calcul des coûts de la réorganisation.

Dans les parties qui suivent, nous allons montrer comment notre système de maintenance analyse la qualité du système OMACS, tout en se basant sur l'évaluation de l'efficacité du

système et celle des agents le composant. Cependant d'autres critères de qualité peuvent être utilisés afin de détecter d'autres problèmes du système.

6.2.2. Efficacité des agents

Dans notre modèle, nous proposons de représenter le succès des agents à jouer leurs rôles par ce que nous appelons *l'efficacité des agents***.**

Définition 1

L'efficacité des agents *Ai* par rapport au rôle *Rj*, notée *Eff (Ai, Rj),* est un degré exprimant le succès de l'agent *Ai* à atteindre son but grâce à l'attribution d'un rôle *Rj*.

Cette efficacité dépend principalement des compétences et/ou intérêts de l'agent *Aj* par rapport au Rôle *Rj.*

Table d'attribution Rôle-Agent

L'attribution des rôles aux agents est faite grâce à un mécanisme d'allocation des rôles. Cette allocation est effectuée dans le système OMACS par son Maitre d'Organisation (MO), qui assigne les agents aux différents rôles dans le but d'atteindre les différents objectifs de l'organisation.

Dans le cadre de notre travail de maintenance basé sur le modèle OMACS, nous nous intéressons aux résultats obtenus suite à l'assignation des rôles aux agents. Ce résultat sera représenté sous la forme d'une table appelé ***table d'attribution Rôle-Agent*** **(Table 4.1).**

Table d'attribution Rôle-Agent : la table d'Attribution Rôle-Agent **TRA** est définie par le triplet **<*Rj, Ai, Eff*>**, avec :

Rj : le rôle exécuté par l'agent *Ai* ;

Eff (Ai, Rj) : le succès de l'agent *Ai* à jouer le rôle *Rj*

Rôle	Agent assigné	Efficacité de l'agent assigné
R1	*A1*	Eff (***A1, R1***)
…	***…***	…
Rj	*Ai*	Eff (***Ai, Rj***)

Table 4.1. Table d'attribution Rôle-Agent

6.2.3. Evaluation de l'efficacité des agents

Nous admettons, dans cette thèse, qu'une organisation fonctionne convenablement et pourra aboutir à ses objectifs si les agents la constituant exécutent leurs rôles d'une

manière efficace. Nous cherchons alors à avoir constamment une table globale d'attribution Rôle-Agent qui exprime le succès des agents à jouer leurs rôles.

En cas de perturbation du système, l'agent de maintenance doit pouvoir retrouver les agents les plus aptes à jouer les rôles en difficultés d'exécution à partir de sa Table d'attribution Rôle-Agent. Afin de maintenir et de remédier aux perturbations du système, les degrés d'efficacité des agents doivent être ajustées et corrigées selon les circonstances d'évolution des agents. Comme nous l'avons déjà précisé, cette évolution doit être repérée par l'agent de maintenance. A cette fin, nous proposons de nous baser sur l'évaluation de l'efficacité des agents qui permet d'indiquer le taux de satisfaction de l'exécution du rôle par l'agent.

Exemple d'une situation d'assignation

Dans une équipe de recherche, par exemple où les chercheurs sont représentés par des agents, un agent *Ai* peut jouer le rôle (fournir des documents sur un thème particulier), dans une telle situation si *Ai* fournit des documents qui ne soient pas intéressants, l'assignation de l'agent *Ai* au rôle (fournir des documents sur un thème particulier) est considérée comme non satisfaisante et l'efficacité de l'agent *Ai* à jouer ce rôle doit être corrigée.

La Figure 4.5 présente les différentes étapes du processus de maintenance proposé (Monitoring, évaluation, détection des perturbations, contrôle).

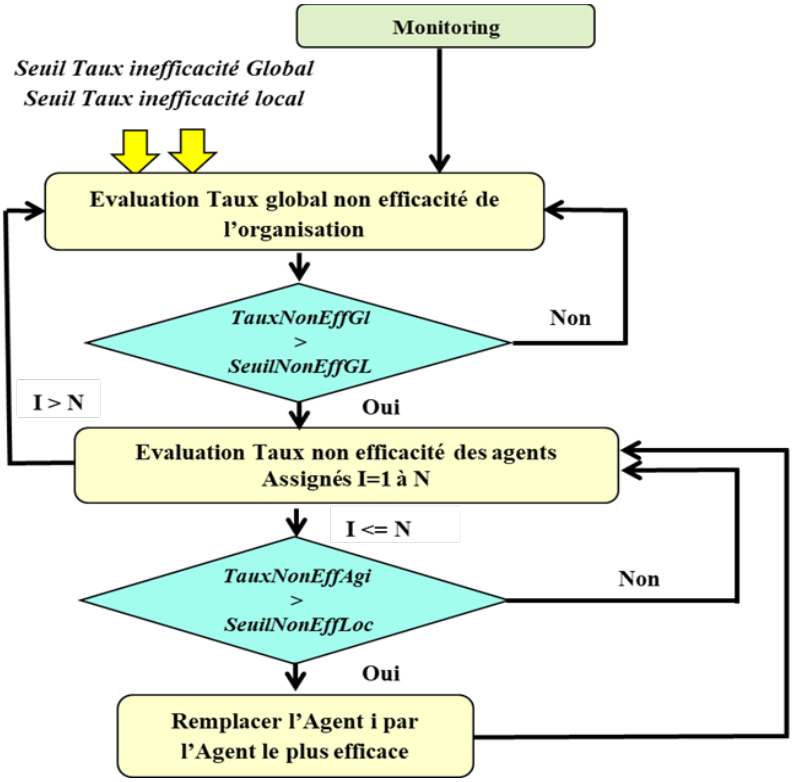


Figure 4.5. Processus de maintenance préventive basé sur l'efficacité des agents

6.2.3.1. Evaluation de l'assignation des agents

L'évaluation **de l'assignation des agents** demeure un moyen important pour la perception des perturbations. Plusieurs critères relatifs aux assignations peuvent être considérés dans ce cas tels que la qualité de l'exécution, le coût de cette exécution, le temps d'exécution, etc. Nous considérons, dans le cadre de cette thèse, la qualité de l'assignation qui est étroitement liée au domaine d'application et le temps d'exécution du rôle.

Nous introduisons pour cela les notions de temps de réalisation et qualité d'assignation.

Définition 2

Temps de réalisation noté ***TR*** est le temps que met l'agent *A* pour réaliser l'objectif du rôle *R*.

Un seuil de temps de réalisation de l'objectif doit être défini pour permettre d'une part d'évaluer l'exécution du rôle *R* par l'agent *A* et d'autre part de prendre en charge le cas où l'agent ne répond pas pour une raison de panne par exemple.

Pour une exécution satisfaisante le temps d'exécution du rôle ne doit pas dépasser un seuil donné,

soit :

TR ≤ TRs où ***TRs*** est le seuil de temps d'exécution

Définition 3

Qualité de l'assignation notée ***QuA*** permet d'estimer le taux de satisfaction de l'exécution du rôle Rj par l'agent Ai selon le gain perçu. La *QuA* est définie relativement au domaine d'application, où *QuA* est un nombre réel appartenant à **[-1, +1].**

L'agent de maintenance (composant d'évaluation) utilise les paramètres, *TR* et *QuA* dans l'évaluation de l'exécution du rôle *Rj* par l'agent *Ai*.

Définition 4

Evaluation de l'assignation notée ***Eval (QuA, TR)*** est la fonction qui permet de fournir une appréciation sur l'exécution de l'assignation d'un agent à un rôle. Cette fonction rend une valeur réelle appartenant à **[-1, +1].**

Nous posons :

Eval (QuA, TR) =

+1	si exécution très satisfaisante
0	si exécution moyenne
-1	si exécution non satisfaisante

L'évaluation de l'exécution est calculée en fonction des deux paramètres *TR* et *QuA* selon l'algorithme suivant :

Si **TR > TRs** // l'exécution du rôle est abandonnée

Alors **Eval (QuA, TR) = -1**

Sinon **Eval (QuA, TR) = QuA**

Une assignation très satisfaisante doit permettre à l'agent *Ai* de renforcer son efficacité à jouer le rôle *Rj*.

Définition 5

Une *assignation d'un* agent *Ai* est dite ***fructueuse*** par rapport au rôle *Rj* si l'exécution du rôle s'est achevée dans un temps acceptable et que la valeur de *QuA* correspondante est bonne, une assignation fructueuse est une assignation où :

Eval (QuA,TR) > 0

L'évaluation des différentes assignations (*Ai, Rj*) dans le temps permet à l'agent de maintenance de calculer l'efficacité de l'exécution du Rôle *Rj* par l'agent *Ai*. La fonction d'évaluation doit être clairement définie conformément au domaine d'application.

6.2.3.2. Calcul de l'efficacité des agents

Les agents ont toute la volonté de coopérer ensemble, s'ils le peuvent pour atteindre l'objectif de l'organisation ; dans le cas contraire nous pouvons affirmer que les agents choisis n'ont pas la volonté ou les capacités nécessaires pour atteindre leurs objectifs convenablement et leurs assignations aux rôles doivent alors être corrigées.

Nous proposons alors que l'agent de maintenance corrige (met à jour) au fur et à mesure l'efficacité des agents dans la table d'attribution Rôle-Agent en fonction de l'évaluation des affectations établies des agents aux rôles. Nous proposons une ***correction adaptative*** qui soit établie *d'un pas* assez faible mais de manière *continue*, nous la qualifions d'adaptative.

Par analogie, suite à chaque assignation d'un agent *Ai* au rôle *Rj*, l'agent de maintenance (le composant d'évaluation) corrige l'efficacité de l'agent Ai à jouer le rôle *Rj*, en fonction de l'évaluation de l'assignation de l'agent Ai en cours ; une assignation fructueuse permet d'augmenter l'efficacité de l'agent *Ai* à exécuter le rôle *Rj*, tandis qu'une assignation dont l'évaluation est négative, diminue de la possibilité de l'agent à jouer ce rôle.

Nous considérons la correction de l'efficacité des agents comme étant un *ajustement* dans le sens où elle est de valeur très petite voire même négligeable (qu'elle soit positive ou négative) mais de manière continue. La même efficacité peut être augmentée et par la suite diminuée suite à deux assignations dont les valeurs des évaluations sont respectivement positives et négatives ;

Nous proposons, à cet effet, d'utiliser un ***pas de correction*** noté β à fixer au préalable (qui soit de l'ordre de 0,01), L'efficacité de l'agent *Ai* à jouer le rôle *Rj* est alors corrigée comme suit :

$$Eff(Ai, Rj) = Eff(Ai, Rj) + \beta \, . \, Eval(QuA,TR) \quad (2)$$

Naturellement la valeur calculée doit toujours être dans **l'intervalle [0, +1].**

6.3. Calcul des taux d'inefficacité et détection des perturbations

6.3.1. Calcul des taux d'inefficacité locaux et détection des perturbations locales des agents

Les perturbations dans un système, où l'objectif primordial est de maintenir un degré d'efficacité acceptable, peuvent être exprimées par l'affaiblissement de ce degré.

Au niveau local des *agents* cet affaiblissement est perçu suite aux assignations non fructueuses.

Suite à chaque assignation établie d'un agent *Ai* au rôle *Ri* le composant d'évaluation (agent de maintenance) calcule d'une part ***Eval (QuA, TR)*** et selon sa valeur il calcule le taux de non efficacité à un instant *t* appelé ***Fij (Ai/Rj)*** et qui représente le taux d'affectations non fructueuses de l'agent *Ai* au rôle *Rj* (Taux d'objectifs non atteint).

Nous définissons *le taux local de non efficacité* ***Fij (Ai/Rj)*** par :

$$Fij(Ai/Rj) = (nAss(Ai/Rj) \,/\, tAss(Ai/Rj)) \quad (1)$$

où

nAss (Ai/Rj) *:* Nombre des assignations non fructueuses de *Ai* au rôle *Rj*

et ***tAss (Ai/Rj)****:* Nombre total des assignations de *Ai* au rôle *Rj*

Ce taux ne doit pas dépasser un seuil prédéfini dit ***seuil de non efficacité locale*** de l'Agent, noté ***Se.***

Dans le cas contraire, l'agent *Ai* se trouve dans une situation de perturbations et est considéré comme agent défaillant.

Dans le cas de détection d'une perturbation globale du système, l'agent de maintenance envoie donc, un signal à l'OM pour remplacer cet agent défaillant par un autre plus efficace.

Si *Fij (Ai/Rj) > Se*

Alors *l'Agent de maintenance envoie un signal de perturbations au Maitre de l'organisation (MO) pour remplacer l'Agent défaillant.*

6.3.2. Calcul du taux d'inefficacité et détection des perturbations au niveau global

L'objectif de la maintenance préventive est de garder un niveau de performance global acceptable permettant au système d'atteindre son objectif. Ce niveau peut être exprimé selon le domaine de l'application, par exemple par : le nombre d'articles produits/unité

de temps, nombre de papiers évaluées/unité de temps, nombre d'articles transportés/unité de temps, etc.

Dans le cadre de ce travail, nous nous sommes intéressés aux perturbations du système qui sont dus aux mauvaises assignations des agents aux rôles, telles que : l'assignation des agents non compétents, l'abondant des agents à leurs objectifs dû à des changements d'intérêts, baisse de capacités des agents, panne d'un agent, etc. Nous avons alors choisi l'évaluation du taux global de perturbations en fonction des taux locaux de non efficacité des agents, comme il a été précisé dans la section précédente. Le taux global des perturbations est exprimé par un taux global de non efficacité dans le système que nous définissons ci-après :

Soit ***F(O)*** le taux global de non efficacité de l'organisation actuelle.

L'agent maintenance calcule *F(O)* comme suit :

$$\mathbf{F\,(O)} = \frac{\dfrac{\sum_{i=1}^{N} \left(\sum_{j=1}^{Ki} \boldsymbol{nAss}\,(\boldsymbol{Ai/Rj}) \right)}{\sum_{j=1}^{Ki} (\boldsymbol{tAss}\,(\boldsymbol{Ai/Rj}))}}{\boldsymbol{N}} \qquad \textbf{(3)}$$

Où **N :** est le nombre total des agents.

Ki : le nombre de rôles total joués par l'agent i

nAss (Ai/Rj) *:* Nombre des assignations non fructueuses de Ai au rôle Rj

et ***tAss (Ai/Rj)****:* Nombre total des assignations de Ai au rôle Rj

Le système doit tout le temps avoir un taux de non perturbation qui ne dépasse pas un certain seuil, soit ***So*** le seuil global de non efficacité de l'organisation et ce selon L'algorithme suivant :

Si *F(O) > So*

Alors *Agent Maintenance procède à la Détection des perturbations locales des agents défaillants.*

6.4. Contrôle

Afin de corriger les perturbations détectées au niveau du système basé OMACS, notre agent de maintenance procède au contrôle du système en appliquant les techniques de réorganisation.

L'approche que nous utilisons actuellement au sein de notre système MAINOMACS pour la réorganisation du système, consiste à utiliser la stratégie de réassignation des agents de l'organisation aux différents rôles afin de remédier aux problèmes de l'organisation courante. Néanmoins, des extensions peuvent être ajoutées à notre système de maintenance, en utilisant d'autres méthodes de réorganisation, tel que la réorganisation par le réajustement des interactions entre agents par exemple.

Dans la section suivante, nous détaillons davantage cette étape de contrôle ainsi que les différents algorithmes que nous avons proposés pour la réorganisation des agents du système.

7. Algorithmes

Cette section, présente les différents algorithmes que nous avons proposés pour la maintenance préventive des systèmes basés OMACS.

7.1. Evaluation locale et Calcul des taux locaux de non efficacité

Le système immergé dans l'environnement s'adapte aux changements imprévus et qui se manifestent par des comportements non efficaces de ses agents.

Le composant d'évaluation (CA) évalue l'efficacité *Eff (Ai, Rj),* des agents *Ai* du système OMACS et leurs Taux de non efficacité *Fij (Ai/Rj)* dans chaque rôle *Rj* qu'ils jouent (en se basant sur les informations qui proviennent du système de Monitoring).

L'algorithme présenté sur la figure 4.6 illustre ce mécanisme :

Algorithm Evaluation locale **;**
Input: Ai, Rj, QuA, TRs

Output: Eff (Ai, Rj), Fij (Ai/Rj)

For each assignation in Rôle Rj

// Calcul Eval (QuA, TR)

If TR > TRs // l'assignation est abandonnée

Then Eval (QuA, TR) = -1

Else Eval (QuA, TR) = QuA

//correction de l'efficacité de l'agent Ai qui joue le rôle Rj

Eff (Ai, Rj) = β . Eval (QuA, TR) + Eff (Ai, Rj) (2)

// Calcul du Taux de non efficacité

If Eval (QuA, TR) < 0 // assignation non fructueuse

Then Fij (Ai/Rj) = (nAss (Ai/Rj) / tAss (Ai/Rj) (1)

Save (Ai, Rj, Fij (Ai/Rj))

End If

End If

EndFor

Figure 4.6. Algorithme Evaluation locale

7.2. Détection de perturbations et décision

Le Composant Détection Perturbations (CDP) consulte en continue le Taux de non efficacité global du système OMACS. Dès la détection d'une progression anormale du taux de non efficacité global du système global par le (CDP), ce dernier, entame donc une procédure de recherche des agents ayant un taux de non efficacité local qui soit supérieur au seuil autorisé. Ces agents sont considérés par notre système de maintenance (MAINOMACS) comme des agents défaillants ayant entrainé la dégradation du système global et sont signes de problèmes éminents qui peuvent toucher le système basé OMACS. Dans la suite du manuscrit on nommera cet agent l'agent défaillant.

Notre système de maintenance de nature préventive considère dans ce cas, que l'organisation actuelle n'est plus valable pour atteindre son but global et anticipe par la réorganisation du système OMACS.

Un algorithme *Détection Perturbation* () a été proposé afin de détecter les perturbations globales du système et de repérer les assignations (*T*) des agents défaillants parmi les agents assignés dans le modèle organisationnel (*O.AssignSet*) (Figure 4.7).

Algorithm Detection Perturbations;

Input: O.AssignSet, So, Se, Ki

Output: T

1. Loop

/* calcul du taux global de non efficacité de l'organisation */

2. F (O) $= \dfrac{\sum_{i=1}^{N} (\sum_{j=1}^{Ki} nAss\ (Ai/Rj)}{\sum_{j=1}^{Ki} (tAss\ (Ai/Rj))} \Big/ N$ (3)

3. IF F(O) > So **Then** */* Perception de perturbation globale*

4. For T € O.AssignSet

5. Fij= GetSeuilNonEfficacité(T.Ai, T.Rj)

6. IF Fij > Se **Then** */* Perception de perturbation* locale

7. SendEventMain (T) *// Envoyer une demande de maintenance au MO concernant l'assignation (T) en exécution*

8. EndIf

9. EndFor

10. EndIf

11. End loop

Figure 4.7. Algorithme Détection Perturbations

L'algorithme nécessite une opération qui permet d'obtenir les Taux de non efficacité locaux des agents participant au système. Cette opération est définie comme suit :

GetRateNotEfficientAgentInRole (Ag, R) : indique le Taux de Non efficacité de l'agent *Ag* jouant le rôle *R*.

Les **lignes 1-3**, calculent et évaluent le taux global de non efficacité de l'organisation par rapport au seuil de non efficacité global autorisé.

Dès la détection d'un taux global de non efficacité global dépassant le seuil autorisé, les **lignes 4-6**, parcourent l'ensemble d'affectations courant dans le modèle organisationnel afin de détecter tout agent défaillant ayant un Taux de non efficacité qui dépasse le seuil autorisé d'exécution de leurs rôles.

La ligne 7 envoie un évènement de demande de maintenance (un Échec de l'objectif de maintenance) au MO concernant l'assignation T de l'agent défaillant.

7.3. Raisonnement de l'organisation

Un Algorithme de raisonnement de l'organisation (AR) traitant les demandes de maintenance est illustré à la figure 4.8.

Algorithm Raisonnement Organisation;
Input : e, T, O, A
Output: Update (O)
1. loop
2. e = GetEvent ( )
3. <Ga, Gadditions, Gdeletions> = G.occurred(e)
4. O.setActiveGoalSet (Ga)
5. O.removeAssignments (Gdeletions)
6. BestAssign = Reassign (A, T)
7. O.addAssignments (Bestassign)
8. sendUpdates (Bestassign)
9. End loop

Figure 4.8. Algorithme de raisonnement de l'organisation

Dès réception d'un évènement de demande de maintenance (e) et l'assignation (T) de l'agent défaillant, l'algorithme *AR* cherche l'assignation optimale *BestAssign* pouvant remplacer l'ancienne assignation (T) de l'agent défaillant. Par optimal, nous nous référons à l'assignation de l'agent de meilleure qualité d'exécution dans le rôle (T.R). En fonction de la nouvelle assignation *BestAssign*, l'*AR* réeffectue toutes les mises à jour nécessaires

dans le *Raisonnement Objectif* et le Modèle l'Organisationnel de l'organisation (O) et envoie l'assignation BestAssign à l'agent concerné pour exécution.

Les lignes 1-2 reçoivent un évènement (e) de demande de maintenance du composant *CDP*. Dans la ligne 3 l'événement (e) est transmis au composant Raisonnement Objectif (RO), qui met à jour l'arborescence des instances d'objectif en supprimant l'objectif de l'agent défaillant (et les objectifs qui en dépendent) de G et en insérant l'objectif initial de l'agent défaillant à nouveau dans G. Le composant RO renvoie donc un tuple qui inclut le nouvel ensemble d'objectifs actifs (Ga), les ajouts à l'ensemble d'objectifs actif (Gadditions) et les suppressions de l'ensemble d'objectifs actifs (Gdeletions). Les lignes4-5 définissent l'ensemble d'objectifs actifs (Ga) et supprime toutes les affectations liées aux objectifs supprimés (Gdeletions) dans le Modèle Organisationnel (O). La ligne 6 appelle la fonction Reassign () afin d'obtenir parmi les agents A de l'organisation, la meilleure assignation BestAssign pouvant remplacer l'assignation T de l'agent défaillant. La ligne 7 ajoute la nouvelle affectation BestAssign dans le Modèle Organisationnel (O). La ligne 8 envoie la nouvelle affectation BestAssign à l'agent concerné pour exécution.

7.4. Contrôle (réassignation des agents défaillants)

Pour la réassignation des agents aux rôles, nous avons proposé l'algorithme de la figure 4.9. L'algorithme de réassignation *Reassign ()* fournit l'assignation optimale BestAssign du meilleur agent pouvant remplacer l'assignation T de l'agent défaillant. Par meilleur agent nous nous référons à l'agent avec la meilleure qualité d'exécution du rôle T.R de l'agent défaillant.

La ligne 1 fournit les assignations possibles PA au rôle T.R de l'agent défaillant parmi les agents A de l'organisation. Les agents potentiels sélectionnés doivent avoir les capacités nécessaires pour jouer le rôle T.R et doivent disposer d'un taux de non efficacité inférieur au seuil autorisé. La fonction GetAgentsForRoles (), fournit toutes les affectations possibles PA entre les agents A du système et la paire objectif-rôle (T.R, T.O) de l'agent défaillant.

La ligne 2 utilise la fonction GetAgentsValidForRole (pa) et fournit l'ensemble des affectations valides PAV qui répondent aux politiques d'affectation parmi les affectations possible PA. La politique qui limite le nombre d'affectations autorisées pour un agent par exemple, entraine la suppression de l'assignation d'un agent Ai au rôle Rj, si ce dernier dispose déjà du nombre maximum d'assignation dans le rôle Rj.

La ligne 3 utilise les valeurs d'efficacité individuelles des agents candidats pour sélectionner l'assignation BestAssign du meilleur agent parmi les assignations valides PAV.

La fonction *GetBestAgentForRole (pav)* fournit la meilleure assignation *BestAssign* parmi l'ensemble d'affectation possible *pav*.

Function Reassign (A, T);

1. PA :=GetAgentsForRole (A, T.R, T.O)

2. PAV :=GetAgentsValidsForRole (PA)

3. *BestAssign* :=GetBestAgentForRole (PAV)

Return *BestAssign*

Figure 4.9. Algorithme de Réassignation des agents aux rôles

Remarques

Selon le type du système à maintenir, deux types de maintenance préventive peuvent être appliqués :

- Une maintenance préventive conditionnelle telle que dans les systèmes de production
- Une maintenance préventive planifiée tel que dans les systèmes de gestion des conférences

8. Etude de cas

Dans l'objectif de valider notre approche, nous avons choisi une étude de cas qui concerne la gestion d'une agence touristique. Ce cas d'étude est intéressant, car il présente des scénarios permettant à un ensemble varié d'éventements (baisse de performance) d'apparaître au cours de la vie organisationnelle. De plus, comme l'organisation est située dans un environnement qui influence son comportement, non seulement des évènements internes peuvent apparaitre, mais aussi des événements externes peuvent affecter les activités de l'agence de voyage. Ce cas d'étude est également intéressant car il présente un exemple qui vient du monde humain et il est traduit dans la perspective des agents, illustrant que les OCMAS sont utiles pour faire des simulations d'organisations du monde réel.

La réorganisation dans ce cas d'étude serait appliquée comme solution à une mauvaise performance organisationnelle. Les solutions apportées impliqueront des modifications à différents éléments organisationnels, tel que la réassignation des agents aux différents rôles par exemple.

8.1. Description de l'étude de cas

L'application touristique que nous avons choisie est partitionnée en trois groupes : les agents clients, les agents courtiers et les agents fournisseurs. Les agents clients ont besoin de services et demandent aux courtiers des informations sur les différents forfaits de voyage, ce qui implique la réservation d'hôtels, de vols, de trains, etc. Les agents

fournisseurs sont les agents qui appartiennent aux hôtels, aux compagnies aériennes et aux compagnies ferroviaires spécifiques, etc. Les agents courtiers sont chargés d'offrir des forfaits de voyage aux clients suite à leur négociation avec les différents agents fournisseurs.

Les agents clients interagissent avec les agents courtiers afin d'obtenir les services de réservation requises. Le courtier est donc chargé de trouver les meilleurs packages aux clients. Pour cela, il interroge les fournisseurs via des appels à proposition (CFP : Call For Proposal). Le fournisseur va chercher les services de voyage disponibles pour les proposer au courtier. Dès que le courtier arrive à regrouper tous les services demandés dans le package, le client paye et récupère son package de voyage. Sinon il lance une nouvelle demande. La figure 4.10 montre le diagramme d'activité UML du client.

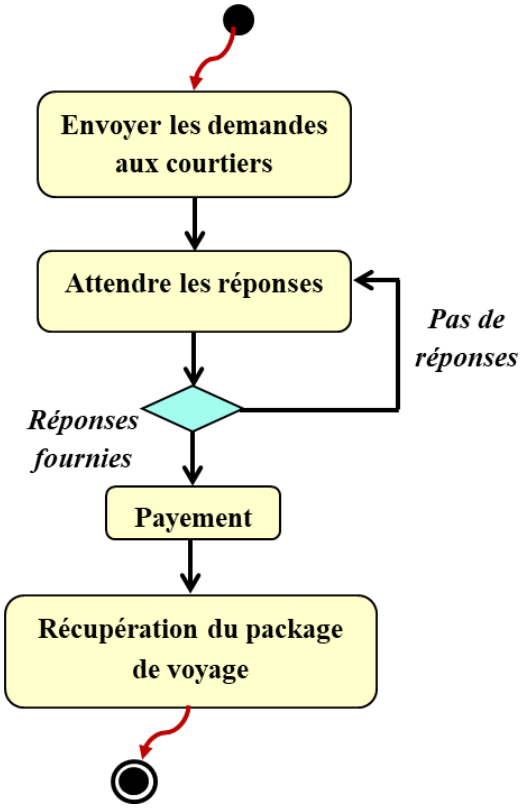


Figure 4.10. Diagramme d'activité du client.

L'agent courtier se place comme étant l'intermédiaire entre le client et le fournisseur. Sa tâche principale est d'interroger les fournisseurs pour avoir des offres, sélectionner les meilleures offres proposées et composer le meilleur package de voyage. La figure 4.11 montre le diagramme d'activité UML du courtier :

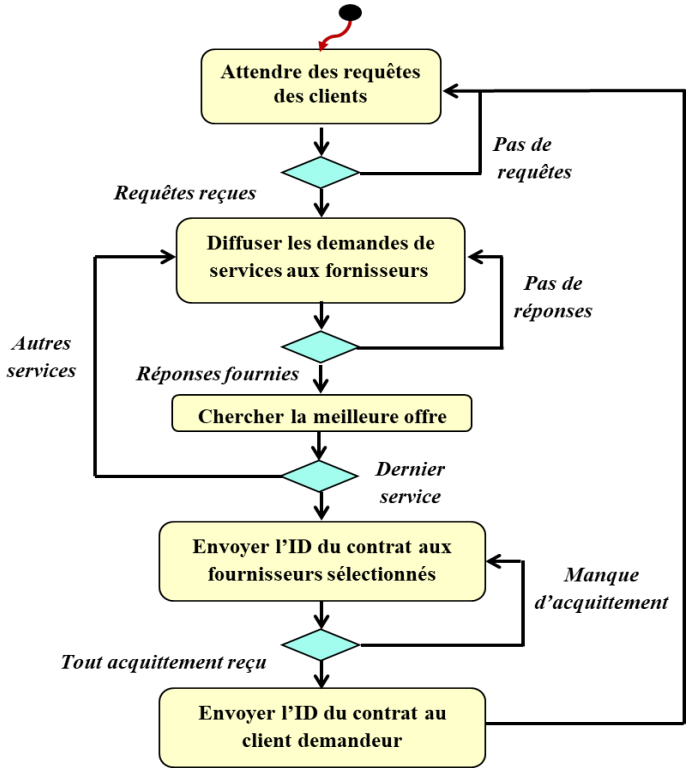


Figure 4.11. Diagramme d'activité du courtier.

En ce qui concerne l'agent fournisseur (figure 4.12), sa mission se résume à la recherche des services disponibles pour les vendre à ses clients. À l'arrivée d'une demande de la part des courtiers concernant une requête d'un service donné, le fourniseur propose le prix du service trouvé. Sinon, si la demande concerne une confirmation d'achat d'un service, le fournisseur va envoyer un aquitement au courtier.

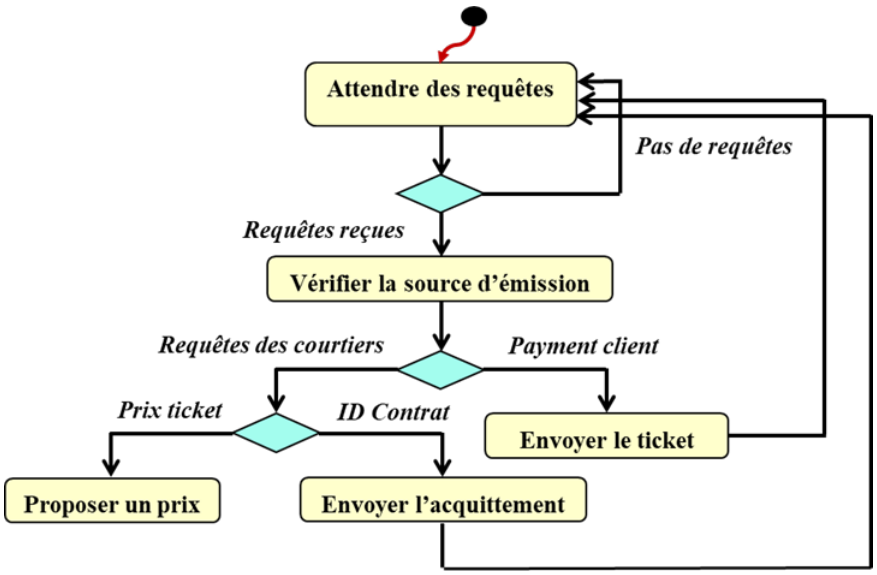


Figure 4.12. Diagramme d'activité du fournisseur.

8.2. Description en utilisant le modèle organisationnel OMACS

Dans ce cas d'étude, nous nous concentrons sur l'organisation des agents courtiers. Tout au long de cette étude, nous n'allons pas nous intéresser au mécanisme d'attribution des rôles aux agents, mais nous allons nous baser sur une entité organisationnelle de départ où un ensemble d'agents sont affectés à leurs rôles. Nous allons alors dans ce qui suit nous limiter à décrire les composants de l'organisation qui nous serviront juste à valider notre approche.

L'organisation des agents courtiers est définie selon le modèle organisationnel OMACS par l'ensemble des objectifs (G), les rôles (R) et les agents de l'organisation (A), tel que :

- G = *(VendPackageAffaire, VendPackageSportifs, VendPackageFamille)*

- R = *(CourtierAffaire, CourtierSport, CourtierFamille)*

Le but de cette organisation est de vendre des packages de voyages de différents types. Selon le type de voyage demandé cette organisation à trois objectifs : vente des voyages d'affaires (*VendPackageAffaire*), vente des voyages pour activités sportives (*VendPackageSportifs*), ventes des voyages en familles (*VendPackageFamille*).

Dans cette organisation, trois rôles différents sont définis pour chaque instant t. Chacun de ces rôles est spécialisé pour un type de voyage spécifique : voyages d'affaires (*CourtierAffaire*), activités sportives (*CourtierSport*) et famille (*CourtierFamille*). Chaque rôle réalise un objectif qui est lié à la gestion d'un type spécifique de forfaits de voyage dans lequel le rôle est spécialisé. A titre d'exemple, le rôle *CourtierAffaire* réalise l'objectif *VendPackageAffaire*.

La population d'agents A est définie pour un instant donné par l'ensemble des agents de l'organisation qui jouent les rôles Courtiers (R) et qui réalisent les ventes des différents types de packages de voyage (G). A un instant t, un agent *Ai* de (A) réalise l'objectif *Oj* de (G) si les capacités de *Ai* à cet instant lui permettent de jouer le rôle *Rj* de (R) qui vise l'objectif *Oj*. Néanmoins, cet agent peut jouer différents rôles qu'il ne joue pas actuellement.

Chaque agent *Ai* a une capacité associée pour jouer un rôle Rj qui est défini comme *Capable (Ai; Rj)*. Cette aptitude définit dans quelle mesure un agent, peut jouer un rôle. Par exemple, *Capable (Ai; CourtierSport)* estime le rendement de l'agent Ai à offrir les services nécessaires pour jouer le rôle *CourtierSport.*

Dans un laps de temps compris entre t et t', chaque agent courtier *Ai,* qui joue le rôle *Rj,* reçoit un certain nombre de requêtes d'agents Clients qui demandent des forfaits de voyage liés à l'objectif *Oj* (vente de package de voyage). Ce nombre de requêtes est représenté par $R(Ai, Oj)_t^{t'}$. La récupération d'une requête dans cette organisation est reconnue comme l'événement de démarrage, qui est envoyé au (Maitre de l'Organisation)

MO et qui provoque l'instanciation de l'objectif de vente de package et son insertion dans la liste des objectifs (G) de l'organisation.

L'utilité de chaque agent de l'organisation est représentée par le nombre de ventes de cet agent multiplié par le bénéfice obtenu de la vente de chaque package (forfait de voyage individuel). L'objectif de cette organisation est de maximiser l'utilité totale de l'organisation qui résulte de la somme des utilités de tous les agents faisant partie de cette organisation.

Au moment de la conception, il peut ne pas être possible de savoir quelle distribution de rôles entre les agents fournit l'utilité la plus élevée. En outre, même si l'allocation de rôle la plus performante est connue, elle peut devenir catastrophique pour l'organisation, car les demandes des clients et les capacités des agents peuvent changer dans le temps, ce qui nécessite une distribution différente. Par conséquent, des transitions préventives de l'organisation offrent des alternatives pour améliorer l'utilité de l'organisation si cette dernière est devenue mauvaise en dessous de certains seuils à cause de certains facteurs environnementaux ou autres. Dans cet exemple, nous employons la réassignation des agents aux rôles comme actions de réorganisation afin de remédier aux problèmes rencontrés dans les différents scénarios qui suivent.

8.3. Scénarios d'exécution

A travers les scénarios qui suivent nous montrons comment détecter les anomalies d'exécution du système au moyen des métriques proposées, puis nous décrirons les solutions spécifiques qui sont appliquées par notre système de maintenance aux différents scénarios. Enfin, nous présenterons les résultats obtenus suite aux expériences d'évaluation de notre système de maintenance.

8.3.1. Scènario1 : Utilisation de la métrique d'efficacité

A travers les expériences que nous avons menées, nous montrons dans cette partie l'utilité d'utiliser la métrique d'efficacité dans notre système de maintenance.

Dans le scénario suivant, nous nous intéressons uniquement à l'organisation des agents courtiers. Bien qu'ils se comportent clairement différemment, plusieurs types d'agents différents sont capables de jouer ces rôles. Chacun de ces rôles vise un objectif spécifique comme le montre la table 4.2.

Rôles	Objectifs
CourtierAffaire	VendPackageAffaire
CourtierSport	VendPackageSportifs
CourtierFamille	VendPackageFamille

Table 4.2. Objectifs associés à chaque rôle

Dans ce système de gestion des voyages, nous nous sommes concentrés sur les échecs des agents courtiers à atteindre l'objectif de vente des packages de voyages. Nous avons modifié les agents de telle sorte que certains d'entre eux n'atteignent pas leurs objectifs dans certaines conditions. Dans ce scénario, nous avons deux types d'agents courtier : un qui n'échoue jamais lors d'une affectation, et un autre, qui échoue 80% du temps lorsqu'il essaye de vendre certains types de voyages (courtier spécialisé).

Pour nos expériences, nous disposons de *l'historique des affectations* (Agent, Rôle, Objectif), *le taux global et les taux locaux de non efficacité* (nombre d'échecs) à ce jour ainsi que l'évaluation de *l'efficacité* d'exécution *des agents* courtiers aux différents rôles. Comme expliqué précédemment, Le taux local *Fij (Ai/Rj)* et le taux global de non efficacité *F(0)* à un instant *t* représentent respectivement le taux d'affectations non fructueuses de l'agent courtier *Ai* au rôle *Rj* (Taux d'objectifs non atteint) et le taux d'affectations non fructueuses de tous les agents courtiers *Ai* aux rôles *Rj* dans toute l'organisation (les Taux local et global de non efficacité sont calculés en utilisant les équations définies dans les parties 5.3.1 et 5.3.2 du chapitre 4).

L'efficacité des agents courtier *Eff (Ai, Rj)* est calculée par l'évaluation de la qualité de leurs assignations aux différents rôles. Vu l'objectif de notre système qui consiste à augmenter l'utilité de l'organisation, alors nous avons choisi de considérer les agents courtiers avec les meilleurs temps de réponses comme étant les courtiers les plus efficaces. Car les meilleurs temps de réalisation permettent d'augmenter le nombre de packages vendu et permettent ainsi d'augmenter l'utilité de l'organisation. L'évaluation de la qualité des assignations des agents aux différents rôles est calculée comme suit :

$$\textbf{Eval (QuA, TR)} = \begin{cases} \textbf{+1} & \textbf{Si } 0 < TR \leq TRs/2 \text{ (exécution très satisfaisante)} \\ \textbf{0} & \textbf{Si } TRs/2 < TR < TRs \text{ (exécution moyenne)} \\ \textbf{-1} & \textbf{Si } TR \geq TRs \text{ (exécution non satisfaisante)} \end{cases}$$

Dans cette simulation, l'organisation est composée d'un ensemble de six agents A0, …, A5. Chaque agent a une capacité associée à jouer chaque rôle qui est déterminée entre 0 et 1. Chaque agent joue un rôle différent R0, …, R2 selon le type de services de gestion touristique qu'il fournit.

Pour chaque test, nous reproduisons un système qui simule les demandes des agents clients pour les forfaits de voyage. Dans un scénario réel, la demande de services peut changer à tout moment. Par conséquent, afin de simuler une exécution plus réaliste et dynamique, nous avons fait varier la demande de services au cours des 80 pas de temps

A t0, l'organisation reçoit 30 demandes d'agents utilisateurs pour chacun des types de packages touristiques. Les 30 requêtes de vente de chaque type de package (Oj) sont reçues par deux agents qui jouent le rôle (Rj).

A l'instant initial t0, chaque rôle est joué par deux agents ayant un taux d'échecs élevés dans ce rôle (sans tenir compte de la capacité de chaque agent à fournir les services nécessaires pour jouer ce rôle). Autrement dit, ces agents ont un taux local de non efficacité énorme Fij qui atteint 0.8 dans cette expérience. Cette distribution est effectuée afin de déclencher les états d'échec. Par exemple, le rôle *Courtieraffaires* (pour la vente des voyages d'affaires) est attribué à des courtiers spécialisés qui échoue à 80% du temps dans la vente de ce type de voyages. Dans ce cas les échecs de vente vont se multiplier et le système risque d'avoir une dégradation considérable de ses performances.

Notre système de maintenance permet dans ce cas la prévention de ce type de problèmes à travers le monitoring et la réorganisation du système, comme suit :

Monitoring

Le système de monitoring surveille le taux d'inefficacité global de l'organisation et les taux d'inefficacité locaux des agents. Dès que le taux d'inefficacité global *F(O)* de l'organisation dépasse le seuil autorisé *So,* la condition de détection de non efficacité de l'organisation est vérifiée :

$$\mathbf{F(O) > So}$$

Une réorganisation est déclenchée par l'agent de maintenance dans ce cas, afin d'éviter une dégradation énorme de l'utilité de l'organisation.

Réorganisation

Les actions de contrôle générées par notre système consistent dans ce cas à réassigner les rôles des agents ayant un taux d'inefficacité dépassant le seuil autorisé (courtiers spécialisés) à d'autres agents ayant une efficacité meilleure dans ces rôles. Pour ce cas, notre système permet par exemple de remplacer les agents non compétents (Fij (Ai/ courtier affaires) > Se) dans la vente des packages des voyages d'affaires (courtiers spécialisés) par d'autres agents plus efficaces dans les voyages d'affaires, c'est-à-dire remplacer toute les assignations *(courtier spécialisé, courtier affaires, vend voyage (affaire))* à d'autres courtiers ayant une meilleure efficacité dans l'exécution du rôle *courtier affaire* afin d'éviter d'autres échecs dans l'exécution du rôle *courtier affaire*, une perte excessive dans les temps d'exécution et une dégradation supplémentaire de l'utilité de l'organisation.

Trace d'exécution

La trace d'exécution des actions de réorganisations dans ce cas est comme suit :

Supprimer.Assignation (courtier spécialisé, CourtierAffaires, VendPackageAffaire)

Ajouter.Assignation (Best Courtier, CourtierAffaires, VendPackageAffaire)

Afin de valider notre approche de maintenance, nous avons mené deux expériences et nous avons mesuré l'utilité totale d'exécution de l'application touristique avec et sans maintenance. Le but de ces expériences est d'évaluer l'utilité de l'organisation afin de montrer la capacité de notre système de maintenance à trouver une meilleure organisation vers laquelle effectuer la transition. Cette transition permettra de prévenir le système d'une dégradation considérable de ses performances et d'une probable défaillance.

L'utilité de l'organisation entre deux pas de temps [t, t'] est mesurée comme le profit généré par toutes les ventes des agents de l'organisation au cours de cette période :

$$\boldsymbol{U(O)_t^{t'} = \sum_{Ai \in A} U(Ai)_t^{t'}}$$

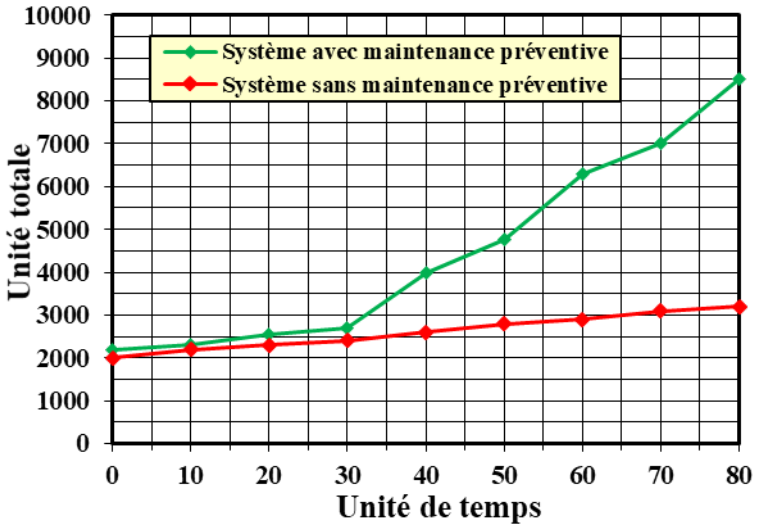
Sachant que, l'utilité de chaque agent (jouant un rôle) pendant la période de temps [t, t'] est obtenue par la multiplication du Nombre de Ventes (NV) de cet agent par le gain obtenu de la vente de chaque forfait de voyage individuel (Oj).

$$\boldsymbol{U(Ai)_t^{t'} = NV(Ai, Oj)_t^{t'} * \text{Gain (Oj)}}$$

Résultats

La figure 4.13 montre une comparaison de l'utilité totale de l'organisation dans les deux cas sans maintenance dans ***le premier cas*** et avec maintenance dans ***le second cas***. Cette utilité est indiquée sur l'axe des ordonnées en fonction du temps d'exécution (pendant 80 cycles d'horloge). Tel que nous pouvons le constater sur la figure, nous assistons à une baisse globale de l'utilité totale maximale du système à 3100 dans le premier cas relativement à un maximum de 8500 pour le même laps de temps dans le second cas. La différence entre les utilités des deux cas est de 3100 à 8500 ce qui équivaut à 5400 unités d'utilité. Cela est dû à la mauvaise attribution des agents aux différents rôles courtiers sans qu'il y ait aucune réorganisation des agents dans ***le premier cas***.

La figure 4.13 montre également que l'organisation dans ***le second cas*** a subi une réorganisation en t= 30 afin de trouver une organisation qui réalise des profits plus élevés. En raison de la mise en œuvre de l'algorithme de réorganisation dans le second cas, nous constatons une amélioration considérable de l'utilité du système. Avec la progression du temps, l'utilité du système après l'intervention de l'agent de maintenance a tendance à augmenter à mesure que l'algorithme de réorganisation déploie le meilleur courtier pour le meilleur rôle. Ce résultat montre aussi clairement l'intérêt d'utiliser la réorganisation dans notre système de maintenance.



Afin d'éviter une dégradation excessive de l'utilité de l'organisation, une autre politique peut également être appliquée dans ce cas. Cette seconde politique de maintenance est basée sur les taux d'inefficacité locaux des agents dans l'exécution de leurs rôles. Etant donné un taux élevé d'inefficacité des agents *courtiers spécialisés* dans l'exécution du rôle *Courtier Affaire*, la politique suivie consiste à interdire l'attribution des ventes des voyages d'affaires aux *courtiers spécialisés* (ayant des taux élevés d'inefficacité locaux supérieur au seuil autorisé dans ce rôle). Autrement dit, toutes les assignations *(courtier spécialisé, courtier affaires, vend voyage (affaire))* seront suspendu. Cette politique et d'autres politiques peuvent également être appliquées pour la prévention des agents inefficaces dans ce cas.

Le choix d'une politique ou de l'autre est fixée selon des tests spécifiques. Ces tests permettront d'adopter la politique qui correspond le mieux aux objectifs de la maintenance (tel que l'optimisation du coût de maintenance par exemple).

8.3.2. Scénario 2 : Utilisation de la métrique Surcharge

Ce scénario permet de découvrir les effets de la surcharge des agents sur les performances du système et montre l'importance de notre approche de maintenance préventive pour atténuer ces effets.

L'analyse de cette étude de cas concerne les interactions entre les clients et les agents courtiers. L'expérimentation s'est déroulée sous une configuration de 10 agents jouant le rôle *CourtierAffaire*, et de 10 agents qui jouent le rôle *CourtierSport*. Il convient de mentionner que ce cas d'étude contient des agents qui utilisent des capacités différentes en jouant le même rôle. Plus précisément, tous les agents courtiers jouent le même rôle, mais chacun d'eux joue ce rôle avec des capacités différentes (par exemple une performance particulière, un certain prix, …etc.).

Dans ce scénario, nous supposons que l'agent courtier B4 est toujours sélectionné pour jouer le rôle *CourtierAffaire* en raison d'une performance particulière de cet agent. L'agent B4 qui joue le rôle *CourtierAffaire* aura donc un nombre élevé de demandes de packages (messages reçus) de la part des clients pendant que les autres agents jouant ce rôle ne reçoivent aucune demande.

La Table 4.3 montre les valeurs de la métrique de surcharge dans le cas où aucune intervention préventive n'est appliquée sur le système.

Agent ID	SR (Aj)
Agent A0	0.99
Agent A1	1.10
…	…
Agent A9	0.99
Agent B0	0.00
Agent B1	…
Agent B4	10.00
…	…
Agent B9	0.00

Table 4.3. Valeurs des métriques de surcharge des agents

Il convient de mentionner que la plupart des agents jouant le rôle *CourtierAffaire* sont complètement ignorés (au repos) et obtiennent la valeur zéro pour cette métrique. La raison de ce comportement défectueux est le mauvais choix des Agents courtiers.

Afin d'éviter une telle situation, notre système de maintenance appliquera des interventions préventives sur le système à travers les actions suivantes :

Monitoring

Le système de monitoring surveille régulièrement *le taux de surcharge d'exécution SR (Ai/Rj) de chaque agent dans son rôle* (Voir section 3.3). A un moment donné *L'agent B4* jouant le rôle *CourtierAffaire* aura donc un nombre élevé de demandes de services (messages reçus). Dès que le taux de surcharge de cet agent *SR (B4/ CourtierAffaire)* dépasse le seuil défini *SeuilChargeRole,* (qui a été fixé à 1 dans cette expérience) la condition de réorganisation suivante est vérifiée :

$$SR\ (B4/\ CourtierAffaire) > SeuilChargeRole$$

Dans ce cas une réorganisation est déclenchée par l'agent de maintenance.

Réorganisation

Afin d'éviter une dégradation de la performance globale du système par l'augmentation des temps de réponses aux demandes des clients, l'agent de maintenance permet de :

- Avertir le MO de la surcharge de l'agent A4 qui joue le rôle Fournisseur.
- Réattribuer le rôle du CourtierAffaire à d'autres agents avec des valeurs de Surcharge *SR (Ai/ CourtierAffaire)* minimales et de meilleurs efficacités *Eff (Ai, CourtierAffaire).*

Résultats

La figure 4.14 montre le temps de réponse du système avec et sans maintenance. Ce temps est indiqué sur l'axe des ordonnées en fonction des demandes de packages. Pour évaluer les temps de réponse, nous avons mesuré la moyenne du temps écoulé depuis la première demande d'un package de voyage jusqu'à la confirmation de la vente soit obtenue par le client pour le nombre de demandes suivant : 10, 20, 30, 40, 50, 60, 70, 80 et 90. Les résultats présentés sur la figure montrent que le SMA avec maintenance fonctionne mieux que celui sans maintenance. On peut également observer que la différence de QoS (représentée par les temps de réponse) dans les deux cas augmente lorsque le nombre de requêtes est plus élevé. Par conséquent, les résultats confirment également que l'effet de la surcharge au niveau de la qualité de service est amplifié lorsque le nombre de demandes augmente.

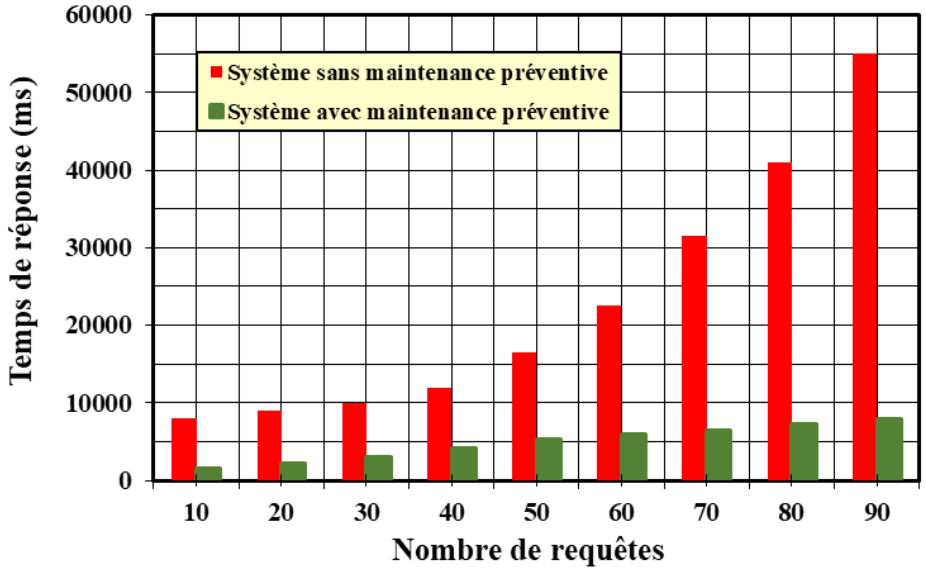


Figure 4.14. Temps de réponse moyen de l'organisation

Ce scénario montre clairement que la métrique proposée peut détecter des comportements de surcharge lorsque les agents jouent le même rôle avec des capacités différentes et qu'une intervention préventive dans une telle situation est nécessaire.

8.3.3. Scénario 3 : Utilisation de la métrique Obtention ressource

Comme expliqué précédemment, l'échec d'obtention des ressources est déclenché lorsqu'un agent n'est pas en mesure d'obtenir les ressources qui sont ses conditions préalables. Dans ce cas d'étude, les ressources nécessaires aux courtiers pour exécuter

leurs objectifs de vente de packages de voyages sont les services offerts par les agents fournisseurs *(BilletAvion, BilletTrain, PlaceHôtel, TicketParcAquatique, TicketThèatre).* Par conséquent, nous considérons qu'un courtier échoue à réaliser un objectif de vente de voyage s'il n'est pas en mesure d'obtenir un service après plusieurs demandes auprès d'un agent *Fournisseur*.

Dans ce cas d'étude, une réorganisation est donc nécessaire lorsque l'expression suivante est vérifiée :

TauxEchecService (Res, t1, t2) > SeuilEchecService,

où

Res = (BilletAvion, BilletTrain, PlaceHôtels, TicketParcAquatique, TicketThèatre) représente l'ensemble des services (prestations) obtenus par les fournisseurs.

Dans ce scénario nous nous intéressons aux interactions entre les agents courtiers et les agents fournisseurs. L'expérimentation s'est déroulée avec 6 agents courtiers. A *t0*, l'organisation reçoit 30 demandes de la part des clients pour chacun des types de packages touristiques. Les 30 requêtes de vente de chaque type de package (VendPackageAffaire, VendPackageSportifs, VendPackageFamille) sont reçues par deux agents courtiers qui jouent successivement les rôles (CourtierAffaire, CourtierSportifs, CourtierFamille).

Tous les agents *courtiers* de cette organisation doivent, entre autres obtenir des réservations de billets d'avions *(Ressource BilletAvion*) de la part des agents *Fournisseurs* pour réaliser leurs objectifs de vente des différents types de packages de voyages. Dans un cycle d'exécution régulier, les courtiers envoient des demandes de services aux fournisseurs. Cette organisation dispose d'un seul Fournisseur *F1* (Compagnie Air Algérie par exemple) qui fournit la ressource *BilletAvion.* Nous considérons dans ce scénario, que cet agent ne peut fournir la ressource *BilletAvion* après plusieurs demandes successives envoyées par les différents *courtiers* dans une période [t1, t2] (échec de l'obtention de la ressource *TicketAvion de la part de l'agent F1*). Ce cas nécessite donc une intervention préventive afin d'éviter l'échec total de l'organisation.

Monitoring

Lorsque nous appliquons notre démarche de maintenance à ce cas, le système de monitoring surveille le *TauxEchecService* dans des intervalles t et t' en suivant la condition de détection de l'échec d'obtention des ressources (décrite dans la section 3.2). Dans le cas considéré, le *TauxEchecService* de la ressource *BilletAvion* dans la période [t1, t2] dépasse le seuil *SeuilEchecService* et la condition de réorganisation suivante est vérifiée :

TauxEchecService (BilletAvion, t1, t2) > SeuilEchecService

Dans ce cas, l'agent de maintenance détecte une perturbation et envoie une demande de réorganisation au *MO*.

Réorganisation

Afin de faire face à cette perturbation, l'agent de maintenance permet de conclure un accord avec un agent *Fournisseur* supplémentaire afin de répondre aux demandes de services des agents *courtiers* (offrir les ressources *BilletAvion* nécessaires). Dans ce cas, l'organisation nécessite l'ajout d'un autre agent F2 pour fournir la ressource *BilletAvion*.

Trace d'exécution

L'action de réorganisation effectuée dans ce cas est la suivante :

Ajouter (F2)

Résultats

La figure 4.15 montre l'évolution de l'utilité de l'organisation dans le temps, sans maintenance dans ***le premier cas*** et avec maintenance dans ***le second cas***.

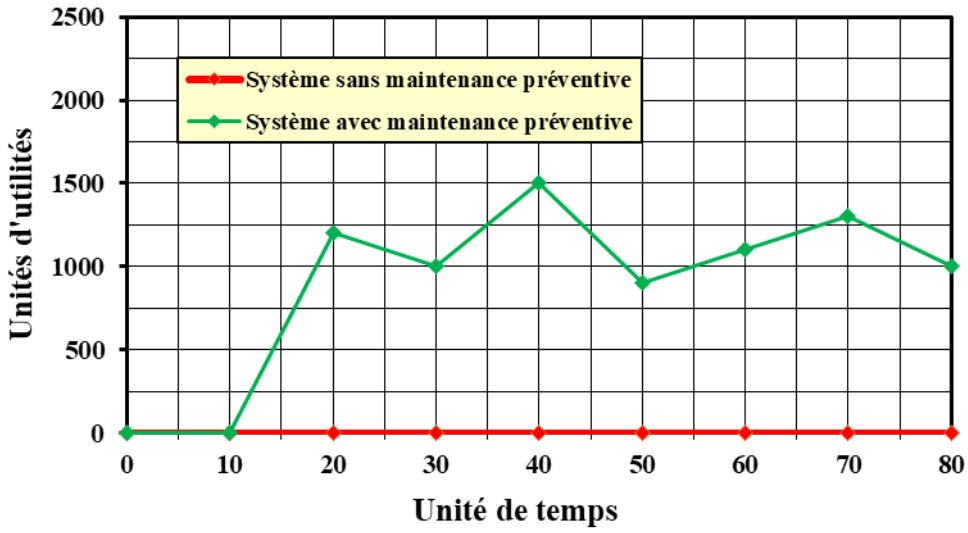


Figure 4.15. Evolution de l'utilité de l'organisation en fonction du temps

Cette utilité est indiquée sur l'axe des ordonnées en fonction du temps. Pour évaluer l'utilité de l'organisation, nous avons mesuré l'utilité totale des agents qui sont en cours d'exécution dans des intervalles de temps fixes [t, t'].

Tel que nous pouvons le constater sur la figure 4.15, nous assistons à une utilité nulle dans ***le premier cas*** relativement à un maximum d'utilité de 1500 dans ***le second cas***. Cela est dû au manque de la ressource *BilletAvion,* nécessaire à tous les agents *courtiers* pour réaliser leurs objectifs de vente de packages de voyage dans le premier cas. Comme le

système n'a subi aucune intervention, les échecs se sont multipliés et l'utilité de l'organisation s'est stagnée à une valeur de zéro.

La figure 4.15 montre également que l'organisation dans ***le second cas*** a subi une réorganisation en t= 10 afin de trouver une organisation qui réalise des profits. En raison de l'ajout de l'agent *Fournisseur F2* dans l'organisation, nous constatons que le système a repris son fonctionnement normal et que les agents reprennent leurs productions après avoir réussi à obtenir la ressource *BilletAvion* de la part du nouveau *Fournisseur F2*.

Ces résultats montrent que la métrique *TauxEchecService* a permis de détecter le manque de ressources dans l'organisation et a permis également d'intervenir au bon moment.

A la fin de toutes les expériences menées, il apparait clairement que notre mécanisme de maintenance a été capable d'éviter rapidement plusieurs états de défaillance ayant plusieurs sources indépendantes. Les performances du système ont augmenté à mesure que le système s'est adapté à son environnement. Une maintenance préventive adéquate à ce genre de situation permet donc d'éviter des situations pouvant être néfastes dans la vie de l'organisation.

9. Discussion

Suite à l'étude des approches de maintenance préventive que nous avons menée au second chapitre, nous avons constaté que notre système de maintenance MAINOMACS présente plusieurs avantages :

- Toutes les approches étudiées concernent la maintenance préventive des logiciels. Cependant, à l'exception de l'approche que nous avons proposée, aucun de ces travaux ne traite la maintenance préventive des SMA. L'approche proposée, était une première étape vers la proposition d'une approche générique de maintenance préventive pour les applications multi-agents organisationnels (OCMAS).

- MAINOMACS ramène une nouveauté en terme du type de la maintenance préventive suivie qui est conditionnelle (non planifiée). Ce type a été délaissé par la plupart des travaux qui portent sur la maintenance préventive des logiciels. Dans l'approche proposée, nous avons choisi de surveiller le système en exécution à travers des métriques dynamique afin de n'intervenir qu'en cas de besoin. Cette méthode à l'avantage d'économiser le temps et les ressources consommés systématiquement et probablement inutilement dans le cas d'une maintenance préventive planifiée « ***Vaut mieux prévenir que guérir*** »

- Afin de surveiller les différentes métriques, notre système est le seul qui utilise la POA dans la maintenance des logiciels. Ceci a pour objectif l'amélioration des performances liée à la charge de communication imposée par les agents impliqués dans le monitoring dans les approches classiques. L'utilisation des aspects réduit considérablement le nombre de messages échangés et libère en conséquence les canaux de communications.

10. Conclusion

Dans ce chapitre, nous avons présenté les contributions principales suivantes :

D'abord, nous avons proposé une seconde approche de maintenance préventive pour les SMA centrés organisation (OCMAS). Nous avons également proposé des métriques qui aident à détecter les perturbations pouvant surgir dans l'organisation de ces systèmes, et les solutions qui tireront parti de ces métriques afin d'éviter l'endommagement de l'organisation. Ces métriques, s'expriment non seulement avec un langage naturel (pour les descriptions) mais avec des formules mathématiques qui aide les développeurs d'OCMAS à l'identification et à la résolution de ces perturbations.

Ensuite, nous avons appliqué notre approche de maintenance sur les systèmes multi-agents basés OMACS et nous avons décrit l'architecture générale de notre système de maintenance, ainsi que le processus et les algorithmes utilisés pour la maintenance des systèmes multi-agents se basant sur le modèle OMACS.

Ainsi, notre proposition est baptisée MAINOMACS (pour : MAINtenance for multi agent systems based OMACS). Cette approche se base sur l'évaluation de l'efficacité des agents pour le monitoring du système, un ensemble de seuils définis pour détecter les perturbations et des techniques de réorganisation pour le contrôle des systèmes OMACS.

L'approche proposée profite amplement des techniques de la POA pour le monitoring du système. Le monitoring supporté par MAINOMACS s'effectue en temps réel ou encore en parallèle avec le Système en exécution. Ainsi, MAINOMACS se base principalement sur la technique de monitoring orienté-aspect qui est relativement nouvelle par rapport aux travaux existants.

Par ailleurs, les développeurs des systèmes OMACS peuvent profiter vivement de notre approche pour examiner, analyser et maintenir leurs applications :

- En utilisant les mesures proposées, nous pensons que les responsables de la maintenance des systèmes OMACS sont mieux équipés pour prévoir et éviter les problèmes qui peuvent surgir lors de l'exécution des systèmes multi-agents.
- En ayant la possibilité de détecter et de résoudre les perturbations qui mènent au changement organisationnel, cela donne l'opportunité à développer des systèmes OMACS bien maintenu et avec des coûts minimes par notre technique de maintenance de nature préventive.

Enfin, MAINOMACS a été validé, par une étude de cas afin de vérifier comment une perturbation pourrait être détectée et résolue dans un système OMACS.

CONCLUSION GÉNÉRALE ET PERSPECTIVES

1. Conclusion générale

Notre projet de thèse se place dans le contexte du génie logiciel orienté-agent. En particulier, nous nous sommes intéressés au problème de la maintenance préventive des SMA.

En fait, les spécificités des systèmes multi-agents et les environnements d'application imprévisibles réduisent considérablement la capacité du système à accomplir son objectif. Afin de maitriser le comportement d'un SMA et de le diriger vers un état cible souhaité, des contributions bénéfiques ont été proposées.

Nous avons commencé notre thèse par un état de l'art dispatché sur les deux premiers chapitres. Le premier chapitre porte sur les SMA dans lequel des concepts et des notions inhérentes aux SMA en général et aux systèmes multi-agents organisationnels en particulier ont été définis. Dans le second chapitre nous avons présenté des concepts clés sur la maintenance et la qualité des systèmes. Notre étude concernant les travaux effectués sur la maintenance préventive des logiciels a révélé que :

- Bien que les approches de maintenance aient apporté des réponses intéressantes aux différents problèmes relatifs à la maintenance des logiciels, peu de travaux ont abordé le problème de la maintenance préventive conditionnelle.

- La majorité des travaux sur le monitoring ne prennent pas en compte les spécificités des systèmes de nature dynamique et imprévisible.

De plus, l'étude de l'état de l'art nous a permis de souligner l'absence totale de travaux sur la maintenance préventive dans les SMA, et nous a aidé à nous positionner par rapport à la littérature.

Au cours de cette thèse, nous avons apporté deux contributions principales. Dans le troisième chapitre, nous avons présenté notre première contribution sur la maintenance des SMA. Une nouvelle approche de maintenance a été introduite dont l'idée principale est de s'appuyer sur les techniques de la POA pour le monitoring du SMA. En fait, la POA a suscité un grand intérêt dans la littérature surtout en ce qui concerne l'analyse dynamique de logiciels. L'originalité de notre approche de monitoring vient de la prise en compte des spécificités des SMA par un paradigme de programmation relativement nouveau. Nous avons montré le fonctionnement de notre approche à travers un outil logiciel nommé PMMAS (pour Preventive Maintenance of Multi-Agent Systems). L'outil développé a été illustré à travers une étude de cas bien concrète sous la plateforme Jade.

Dans le quatrième chapitre, nous avons proposé des métriques qui aident à détecter certaines perturbations pouvant surgir dans le système, et des solutions qui tireront parti de ces métriques afin d'éviter l'endommagement de l'organisation. Par la suite, une seconde approche de maintenance pour les SMA organisationnels a été discutée dans ce chapitre. L'approche proposée profite également des techniques de la POA pour le monitoring et des techniques de la réorganisation pour le contrôle (réparation) du système. Ainsi, Nous avons présenté une méthode pour surveiller la qualité (la

performance) du SMA en cours d'exécution. Cette méthode est particulièrement applicable aux SMA organisationnels qui peuvent changer leurs organisations au moment de l'exécution. Pour de telles classes de systèmes, l'analyse statique du système mis en œuvre n'est pas suffisante. En utilisant l'analyse dynamique, la qualité du système peut être extraite à un moment donné et peut être évaluée pour détecter une éventuelle dégradation de la qualité du système en exécution. Nous avons décrit une nouvelle approche pour le monitoring du système en cours d'exécution en utilisant des idées de la programmation orientée aspect. De plus, nous avons montré comment remédier aux problèmes de dégradation de la qualité du logiciel à l'aide des techniques de réorganisation. Ainsi, notre proposition est nommée MAINOMACS (pour MAINtenance for Multi-Agents Systems based OMACS.) Finalement, notre proposition MAINOMACS a été validée sur une étude de cas.

2. Perspectives

L'approche de maintenance proposée constitue un premier pas vers une approche de maintenance préventive dans les SMA. Néanmoins, nos contributions restent ouvertes et extensibles sur plusieurs directions.

- Afin de proposer une première approche pour la maintenance des SMA, nous nous sommes focalisés dans un premier temps sur le monitoring. Notre objectif principal, consistait à l'identification d'un ensemble de métriques qui nous permettront de surveiller l'exécution du système. Par la suite, nous avons cherché à employer des méthodes adéquates pour mesurer ces métriques. Néanmoins, dans nos contributions nous avons abordé superficiellement l'évaluation de la qualité des SMA. Une piste prometteuse consiste à développer des modèles d'évaluation de la qualité des SMA, qui permettront la mesure de la qualité globale des SMA et l'interprétation des résultats obtenues. Ces modèles d'évaluation augmenteront l'utilité de la maintenance des SMA et apporteront une aide précieuse aux développeurs de ces systèmes.

- Dans le cadre de ce travail, nous avons conçu à chaque cas (pour chaque cas nous avons proposé une seule solution afin de réparer le système)), une seule solution pour la réparation du système. Afin d'améliorer les procédures de maintenance, nous envisageons l'inclusion de toutes les étapes du processus de réorganisation dans notre approche de maintenance des SMA organisationnels (OCMAS). Cela peut être effectué par la conception de plusieurs solutions et la sélection de la meilleure solution pour réparer le système. La sélection de la meilleure solution peut être réalisé à travers la mesure de l'impact des actions de maintenance de chaque solution proposée ou par l'évaluation des coûts des différentes solutions proposées.

- Nous envisageons l'utilisation de l'apprentissage automatique afin d'éviter de futurs problèmes graves de l'application en maintenance. L'utilisation de l'apprentissage automatique pour la maintenance préventive des SMA est un moyen intéressant qui pourrait améliorer l'efficacité du processus de maintenance.

- Dans cette thèse, nous prévoyons l'extension de notre approche par la mesure de nouvelles métriques de qualité pour prévenir d'autres problèmes pouvant survenir dans

les SMA. Pour le contrôle de ces systèmes, nous envisageons également, d'étendre notre travail de maintenance, par l'ajout d'autres modes d'intervention, tels que l'intervention par l'emploi des actions de réorganisation structurelle et normative par exemple.

Bibliographie

Abdallah, V., and Lesser, V., (2007). Multiagent Reinforcement Learning and Self-Organization in a Network of Agents. In Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems (AAMAS07), 172–179.

Abdu, H. Lutfiyya, H., and Bauer, M. A., (1999). A model for adaptive monitoring configurations. In Proceedings of the 6th IFIP/IEEE IM Conference on Network Management, 371–384.

Abernethy, R. B., (1996). The New Weibull Handbook, 2nd Edition, Abernethy, North Palm Beach, FL, USA.

AFNOR X60G “Maintenance industrielle", Ed. Afnor, Mai 2002, ISSN 0335-3931

Aggarwal, K. K., and Singh Y., (2005). Software Engineering: Programs, documentation, operating procedures. New Age international publishers.

Alberola, J. M., Julian, V., and Garcia-Fornes, A., (2011). Open Issues in Multiagent System Reorganization, Perez, J.B., et al. (Eds.): Highlights in PAAMS, AISC 89, 151–158. Springerlink.com C Springer-Verlag Berlin Heidelberg.

Alberola, J. M., Julian, V., and Garcia-Fornes, A., (2011). A cost-based transition approach for multiagent systems reorganization. In Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS11), 1221–1222.

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

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

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

Alonso, F., Fuertes, J. L., Martínez, L., and Soza, H., (2010). 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.

Argente, E., Palanca, J., Aranda, G., Julian, V., Botti, V., Garcia-Fornes, A., and Espinosa, A. (2007). Supporting agent organizations. In Multi-Agent Systems and Applications V, 236-245. Springer Berlin Heidelberg.

Avritzer A., and Weyuker, E., J., (1997). Monitoring smoothly degrading systems for increased dependability, Joumal of Empirical Software Engineering, 2(I).

Bansiya J. and Davis-Carl G., (2002). A hierarchical model for objectoriented design quality assessment. Software Engineering, IEEE Transactions on, 28(1), 4–17.

Basili, V. R., Caldiera, G. and Dieter Rombach, H., (1994). The goal question metric approach. In Encyclopedia of Software Engineering. Wiley, 1, 528–532.,

Basili V. R. and Perricone B. T., (1984). Software errors and complexity: an empirical investigation0. Communications of the ACM, 27(1), 42–52.

Bellifemine, F.L., Caire, G. and Greenwood, D., Developing Multi-Agent Systems with jade. John Wiley & Sons, 2007

Benaicha H., (2015). Analyse des stratégies de maintenance des systèmes de production industrielle. Thèse de Doctorat, Université d’Oran Mohammed Boudiaf.

Bennet, K., H., (1998). Do Program Transformations Help Reverse Engineering, Proceedings for International Conference on Software Maintenance, Bethesda, Maryland, USA, November 16-20, 247-254.

Bernon, C., Camps, V., Gleizes, M. P., and Picard, G. (2005). Engineering adaptive multi-agent systems: The adelfe methodology. Agent-oriented methodologies, 172-202.

Bitonto, P. D., Laterza, M., Roselli, T., and Rossano, V., (2012). 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.

Bou, E., Lopez-Sanchez, M., and Rodrıguez-Aguilar, J. A., (2006). Adaptation of autonomic electronic institutions through norms and institutional agents. In Engineering Societies in the Agents World. Number LNAI 4457, 300–319. Springer.

Briand, L. C., (1993). Measuring and Assessing Maintainability at the End of High-Level Design, Proceedings for International Conference on Software Maintenance, Montreal, Quebec, Canada, September 27-30.

Campos, J., Esteva, M., Lopez-Sanchez, M., Morales, J., and Salamo, M., (2011). Organisational adaptation of multi-agent systems in a peer-to-peer scenario. Computing 91, 169–215.

Card, D. N., and Glass, R. L., (1990). Measuring Software Design Quality, 1st ed.: Prentice-Hall, Inc.

Carvalho, G., Almeida, H., Gatti, M., Vinicius, G., Paes, R., Perkusich, A., and Lucena, C., (2006). Dynamic law evolution in governance mechanisms for open multiagent systems. In Proceedings of the Second Workshop on Software Engineering for Agent-oriented Systems (SEAS06).

Case, J.D., Fedor, M., Schoffstall, M.L., Davin, C., Rose, M. T., and McCloghrie, K., (1990). Simple Network Management Protocol. RFC 1157.

Cheluvaraju, B., Pasala, A., Padmanabhuni, S., and Chevireddy, S., (2012). A quantitative measure for preventive maintenance in software, in: ACM SIGSOFT Software Engineering, Notes, 37(4), 1–5.

Chen, Z. S., Yang, Y. M., and Hu, Z., (2012). A technical framework and roadmap of embedded diagnostics and prognostics for complex mechanical systems in prognostics and health management system. IEEE Transactions on Reliability, 61(2), 314–22.

Cheng, B., Lemos, R., Giese, H., Inverardi, P., and Magee, J., (2009). Editors. Software Engineering for Self-Adaptive Systems. Springer-Verlag, Berlin, Heidelberg.

Cheng, G. Q., Zhou B. H., and Li, L., (2016). Joint optimization of production rate and preventive maintenance in machining systems, International Journal of Production Research, 54, 6378-6394.

Chhabra, J. K., and Gupta, V., (2010). A survey of dynamic software metrics. J. Comput. Sci. Technol. 25(5), 1016–1029. doi:10.1007/s11390-010-1080-9.

Chidamber, S. R., and Kemerer, C. F., (1994). A metrics suite for object-oriented design, IEEE Transactions on Software Engineering (TSE), 20(6), 476–493.

Coutinho, L. R., Sichman, J. S., and Boissier, O., (2009). Modelling dimensions for agent organizations. Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models.

D'Ambros, M., Lanza, M., and Robbes, R., (2010). An Extensive Comparison of Bug Prediction Approaches, in IEEE Working Conference on Mining Software Repositories (MSR), Cape Town, South Africa, 31-41.

Daskalantonakis, M. K., (1992). A practical view of software measurement and implementation experiences within motorola. Software Engineering, IEEE Transactions on, 18(11), 998–1010.

Davis. R., (1980). Report on the workshop on distributed artificial intelligence. Sigart Newsletter, 42-52.

DeLoach, S. A., Wood, M. F., and Sparkman, C. H., (2001). Multiagent Systems Engineering. The International Journal of Software Engineering and Knowledge Engineering, 11(3), 231-258.

Deloach, S. A., and Matson, E., (2004). An organizational model for designing adaptive multiagent systems. In: The AAAI 2004 Workshop AOTP, 66–73.

DeLoach, S. A., Oyenan, W. H., and Matson, E. T., (2007). A capabilities-based theory of artificial organizations. Journal of Autonomous Agents and Multiagent Systems.

Deloach, S. A., Oyenan, W. H., and Matson, E. T. (2008). A capabilities-based model for adaptive organizations. Autonomous Agents and Multi-Agent Systems 16, 13–56.

Dennis, J. A., Opzeeland, V., Christian Lange, F. J., and Michel Chaudron, R. V., (2005). Quantitative Techniques for the Assessment of Correspondence between UML Designs and Implementations, in ECOOP Workshop on Quantitative Approaches in Object-Oriented Soft, Glasgow, UK, 1-17.

Devarun G., and Sandip R., (2009). Maintenance optimization using probabilistic cost benefit analysis. Journal of Loss Prevention in the Process Industries, 22(4), 403-407.

Dignum, V., Dignum, F., and Sonenberg, L., (2004). Towards dynamic reorganization of agent societies. In Proceedings of Workshop on Coordination in Emergent Agent Societies, 22–27.

Dignum, V., (2004). A Model for Organizational Interaction: based on Agents, founded in Logic, SIKS Dissertation Series 2004-1, Utrecht University, PhD Thesis.

Dignum, V., Vázquez-Salceda, J., and Dignum, F., (2005). Omni: Introducing social structure, norms and ontologies into agent organizations. In Programming Multi-Agent Systems, Springer, 181–198.

Dimou, C., Symeonidis, A. L., and Mitkas, P. A., (2009). An integrated infrastructure for monitoring and evaluating agent-based systems. Expert Syst. Appl. 36(4), 7630–7643.

Doan Van Bien, D., Lillis, and D., Collier, R. W., (2010). Call graph profiling for multi agent systems. In: Dastani, M., El Fallah Segrouchni, A., Leite, J., Torroni, P., (eds.) LADS 2009. LNCS, 6039, 153–167. Springer, Heidelberg.

Doan Van Bien, D., Lillis, D., and Collier, R. W., (2010). Space-time diagram generation for profiling multi agent systems. In: Braubach, L., Briot, J. P., Thangarajah, J., (eds.) ProMAS 2009. LNCS, 5919, 170–184. Springer, Heidelberg.

Dufour, B., Driesen, K., Hendren, L., and Verbrugge, C., (2010). Dynamic Metrics for Java. In: Proceedings of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2003), 149–168. ACM, New York.

Erceau, J., and Ferbe, J., (1993). Introduction aux premières journées francophones d'intelligence artificielle distribuée et systèmes multi-agents." Actes des Premières Journées Francophones lAD et SMA, 1-8, Toulouse, Avdl.

Fenton Norman, E., and Pfleeger S. L., (1998). Software metrics: a rigorous and practical approach. PWS Publishing Co.

Ferber, J., (1995). Les systèmes multi-agents : vers une intelligence collective, 6, Paris : Interedition.

Ferber, J., and Gutknecht, O., (1998). A meta-model for the analysis and design of organizations in multi-agent systems. In Third International Conference on Multi-Agent Systems, Paris, IEEE, 128-135.

Ferber, J., Gutknecht, O., and Michel, F. (2004). From agents to organizations: an organizational view of multi-agent systems. In Agent-Oriented Software Engineering IV, Springer, 214–230.

Ferber, J., Michel, F., and Baez, J. (2005). AGRE: Integrating environments with organizations. In Environments for multi-agent systems, Springer, Berlin Heidelberg, 48-56.

Galliers, J., (1998). A theoretical framework for computer models of cooperative dialogue, acknowledging multiagent conflict. Phd thesis of Open University, UK.

Garg, S., Puliafito A., Telek M., and Trivedi K. (1998a). Analysis of Preventive Maintenance in Transaction Based Software Systems, IEEE transactions on Computers, 47(1), 96-107.

Garg, S., Van Moorsel, A., Vaidyanathan, K., and Trivedi, K. S., (1998b). A methodology for detection and estimation of software aging, In Proc. of the 9th International Symposium on Software Reliability Engineering, Paderborn, Germany, 9, 282–292.

Gasser, L., (1987). The 1985 workshop on distributed artificial intelligence. Al Magazine, 91-97.

Gaston M. E., and desJardins, M., (2005). Agent-organized networks for dynamic team formation. In Proceedings of the 4th International Joint Conference on Autonomous agents and Multiagent Systems (AAMAS05), 230–237.

Gertsbakh, I., (2000). Reliability Theory with Applications to Preventive Maintenance. Springer.

Ghrieb, N., Mokhati, F., Ghorab, M., A. and Guerram, T. (2020). Towards a preventive maintenance approach for multi-agent applications. Multiagent and Grid Systems – An International Journal (16), 83–99.

Ghrieb, N., Mokhati, F. and Guerram, T. (2021). Maintaining Organizational Multi-Agent Systems: A Reorganization-Based Preventive Approach. In Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Volume 1, pages 384-389

Glasser, N., and Morignot, P., (1997). The reorganization of societies of autonomous agents, in MAAMAW, 98–111.

Glaser, N., Vouton, V., and Box, P. O., (1997). The reorganization of societies of autonomous agents. Proceedings of the 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW97), 98–111.

Guessoum, Z., Ziane, M., and Faci. N., (2004). Monitoring and organizational-level adaptation of multi-agent systems. In Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS04), 514–521.

Hamza S., (2014). Une approche pragmatique pour mesurer la qualité des applications à base de composants logiciels. Langage de programmation [CS.PL]. Université de Bretagne Sud, NNT : 2014LORIS356.

Hanne Bauer, M., (2002). Autonomous Dynamic Reconfiguration in Multi-Agent Systems of LNAI, 2427, Springer-Verlag.

Hannoun, M., Boissier, O., Sichman, J. S., and Sayettat, C. (2000). MOISE: An organizational model for multi-agent systems. In Advances in Artificial Intelligence, 156-165. Springer Berlin Heidelberg.

Héng, J., (2011). Pratique de la maintenance préventive, 3ème Edition, Dunod, Paris, ISBN 978-2-10-074550-0.

Horling, B., Benyo, B., and Lesser, V., (1999). Using self-diagnosis to adapt organizational structures. In: Proc. of the 5th Int. Conf. on Autonomous Agents, ACM Press, New York, 529–536.

Huang, Y., Kintala, C., Kolettis, N., and Fulton, N. D., (1995). Software rejuvenation: Analysis, module and applications, In Proc. of 25th Symp. Fault-Tolerant Computing, Pasadena, CA, 6, 381–390.

Hübner, J. F., Sichman, J. S., and Boissier, O. (2002). A model for the structural, functional, and deontic specification of organizations in multiagent systems. In Advances in Artificial Intelligence, 118–128, Springer.

Hübner, J. F., Sichman, J. S., and Boissier, O. (2002). Moise+: towards a structural, functional, and deontic model for mas organization. In Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1, ACM, 501-502.

Hubner, J. F., Sichman, J. S., and Boissier, O. (2004). Using the MOISE+ for a Cooperative Framework of MAS reorganisation. In: Bazzan, A.L.C., Labidi, S. (eds.) SBIA 2004. LNCS (LNAI), 3171, 506–515. Springer, Heidelberg.

Hubner, J. F., Sichman, J. S., and Boissier, O., (2004). Using the MOISE+ for a Cooperative Framework of MAS Reorganisation. In Proceedings of the 17th Brazilian Symposium on Artificial Intelligence (SBIA04), 506–515.

IEEE (1994). Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990 (1991 Corrected Edition). The Institute of Electrical and Electronics Engineers, Inc.

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

ISO/IEC 9126 (2003). ISO/IEC 9126 Software Engineering – Product Quality. Technical Report, International Standards Organization.

ISO (2010). ISO/IEC/ IEEE 24765, Systems and software engineering – Vocabulary.

ISO/IEC 25010 (2011). Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models– Systems and software engineering.

Jarras I. and Chaib-Draa B., (2002). Aperçu sur les systèmes multi-agents. Série scientifique. Centre interuniversitaire de recherche en analyse des organisations. Montréal, Canada.

Jennings, N. R. (1999). Agent-Oriented Software Engineering. in Multiple Approaches to Intelligent Systems, 4-10, Springer Berlin Heidelberg.

Jennings, N., and Wooldridge, M. (2000). Agent-Oriented Software Engineering. ARTIFICIAL INTELLIGENCE, 277- 296.

Jones, C. (2008). Applied Software Measurement - Global Analysis of Productivity and Quality. THIRD EDN., The McGraw-Hill.

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

Kajko-Mattsson, M., (1999). Maintenance at ABB (I): Software Problem Administration Processes, Proceedings, Conference on Software Maintenance, Oxford, 9.

Kajko-Mattson, M., (2001). Can we learn anything from hardware preventive maintenance? in IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), Skövde, Sweden, 106-111.

Kamboj, S. and Decker, K. S., (2006). Organizational self-design in semi-dynamic environments. In Proceedings of the 2007 IJCAI workshop on Agent Organizations: Models and Simulations, 335–337.

Kephart, J., and Chess, D., (2003). The Vision of Autonomic Computing. Computer, 36(1), 41–50.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm J., and Griswold, W. G., (2001). An Overview of AspectJ, in: Knudsen J. L. (Ed.), ECOOP 2001- Object-Oriented Programming, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 327–354.

Kitchenham, B., and Pfleeger, S. (1996). Software quality: The elusive target. IEEE Software, 1, 12-21.

Kota, R., Gibbins, N. and Jennings, N. R. (2008). Decentralized Structural Adaptation in Agent Organizations, 54–71.

Kota, R., Gibbins, N. and Jennings, N. R. (2012). Decentralized Approaches for Self-Adaptation in Agent Organizations, 1–28.

Kotter, J. and Schlesinger, L., (1979). Choosing strategies for change. In Harvard Business Review, 106–114.

Laird, L. M., and Brennan, M. C., (2006). Software Measurement and Estimation a Practical Approach. Wiley, New York.

Lanubile, F., and Visaggio, F., (1995). Iterative Reengineering to Compensate for Quick-Fix Maintenance, Proceedings for International Conference on Software Maintenance, Opio October 17-20, Nice, France, 140-146.

Lass, R. N., Sultanik, E. A., and Regli, W. C., (2009). Metrics for multiagent systems. In: Madhavan, R., Messina, E., Tunstel, E. (eds.) Performance Evaluation and Benchmarking of Intelligent Systems, 1–19. Springer, Heidelberg.

Lieberherr, K. J., and Holland I. M., (1989). Tools for Preventive Software Maintenance, Proceedings for International Conference on Software Maintenance, October 16-19, Miami, Florida, 174-178.

Luck, M., Mc Burney, P., Shehory, O., and Willmott, S., (2005). Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing). University of Southampton.

Mahar, S., Bhatia, P. K., (2014). Measuring the intelligence of software agent. Int. J. Innovative Sci. Eng. Technol., 1(6), 1–11.

Mansour, S. (2007). Un modèle de gestion distribuée de groupes ouverts et dynamiques d'agents mobiles. Thèse de doctorat de l'université de Pau et des Pays de l'Adour, France.

Marir T., Mokhati F., Bouchelaghem-Seridi H., and Benaissa B., (2016). Dynamic Metrics for Multi-agent Systems Using Aspect-Oriented Programming Application to DIMA Platform. Springer International Publishing Switzerland 2016 Klusch, M., et al., (Eds.): MATES 2016, LNAI 9872, 58–72.

Mattern, F., and Floerkemeier, C. (2010). From the Internet of Computers to the Internet of Things. In From active data management to event-based systems and more, 242-259. Springer Berlin Heidelberg.

Morin, E. (1977). La Nature de la nature. Le seuil.

Mahar, S., Bhatia, P. K., (2014). Measuring the intelligence of software agent. Int. J. Innovative Sci. Eng. Technol. 1(6), 1–11.

Marir T., (2015). Une démarche d'assurance qualité pour les systèmes multi-agents, thèse de Doctorat Informatique. Université Badji Mokhtar - Annaba.

Marir T., Mokhati, F., Seridi-Bouchelaghem, H., Acid, Y., and Bouzid, M. (2016). QM4MAS: a Quality Model for Multi-agent, Systems, IJCAT 54(4), 297–310.

Mathieu, P., Routier, J. C., and, Secq Y., (2002). Principles for dynamic multi-agent organizations. In Proceedings of the 5th Pacific Rim International Workshop on Multi Agents: Intelligent Agents and Multi-Agent Systems (PRIMA02), 109–122.

McCall, J., Richards, P., and Walters, G. (1977). Factors in software quality. concepts and definitions of software quality. Technical report, DTIC Document.

Mira, K. M., (2000). Preventive Maintenance! Do we know what it is ?, IEEE.1063-6773.

Moyahabo, D., Ramerea, O., and Timothy L., (2021). Optimization of condition-based maintenance strategy prediction for aging automotive industrial equipment using FMEA, Procedia Computer Science, 80, 229-238.

Nair, R. Tambe, M. and Marsella, S., (2003). Role allocation and reallocation in multiagent teams: towards a practical analysis. In Proceedings of the Second International Joint Conference on Agents and Multiagent Systems (AAMAS03), 552–559.

Norme AFNOR NF EN 13306, (2001). "Terminologie de la maintenance", Ed. Afnor, Paris.

Oman, P., and Pfleeger, S. L., (1997). Applying software metrics, 46. John Wiley & Sons.

Parikh G., (1986). Software Maintenance Notes (1), in ACM SIGSOFT, Software Engineering Notes, 11(2), 49-57.

Pasala, A., Yannick, L. H., Fady, A., Appala-Raju, G., and Gorthi, R. P., (2008). Selection of regression test suite to validate software applications upon deployment of upgrades, in Australian Software Engineering conference, Perth, Australia, 130-138.

Pearse, T., (1995). Maintainability Measurements on Industrial Source Code Maintenance Activities, Proceedings for International Conference on Software Maintenance, Opio October 17-20, Nice, France, 295-303.

Penny, A. G., and Takang, A. A., (2003). Software maintenance – concepts and practice, in: (2eme ed.). World Scientific, 1-349.

Pfleeger, S. L., and Altee-Joannw, M. (2005). Software engineering : theory and practice. , 3rd ed. New Jersey: Upper Saddle River, N. J. Pearson Prentice-Hall, 716p.

Picard, G., Hübner, J. F., Boissier, O., and Gleizes, M. P. (2009). Reorganisation and self-organisation in multi-agent systems. In 1st International Workshop on Organizational Modeling, ORGMOD, 66-80.

Picard, G., Hübner, J. F., Boissier, O., and Gleizes, M. P. (2009). Réorganisation et auto- organisation dans les systèmes multi-agents. Journées Francophones sur les Systèmes Multi-Agents (JFSMA 2009), 89–98.

Prajapati, A., Bechtel, J., and Ganesan, S. (2012). Condition based maintenance : a survey. Journal of Quality in Maintenance Engineering,18(4), 384–400.

Rahme, J., and Xu, H., (2017). Preventive Maintenance for Cloud-Based Software Systems Subject to Non-Constant Failure Rates, IEEE, 978-1-5386-0435-9.

Rausand, M., and Hoyland, A., (2004). System Reliability Theory-Models, Statistical Methods and Applications. Wiley, second edition.

Ringold, P., Alegria, J., Czaplewski, R., Mulder, B., Tolle, T., and Burnett, K., (1996). Adaptive Monitoring Design for Ecosystem Management. Ecological Applications, 6(3),745–747.

Ron Kenett, S., (2008). Cause and Effect Diagrams, in: Encyclopedia of Statistics in Quality and Reliability? First published: 15, March.

Rosenschein, J., and Genesereth, M. (1985). Deals Among Rational Agents. In Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85), 91-99.

Sarkar, S., Rama, G. M., and Kak, A. C., (2007). API-based and information theoretic metrics for measuring the quality of software modularization," IEEE Trans. Software Engineering, 33(1), 14-32.

Schneidewind, N. F., (1992). Methodology for validating software metrics. Software Engineering, IEEE Transactions, 18(5), 410–422.

Schwabacher, M. A., (2005). A survey of data-driven prognostics. In: Proceedings of the AIAA Infotech@Aerospace Conference; Virginia, USA.

Sivakumar, N., and Vivekanandan, K. (2012). Measures for testing the reactivity property of a software agent. Int. J. Adv. Res. Artif. Intell. 1(9), 26–33.

Seelam, A., (2009). Reorganization of massive multiagent systems: MOTL/O. Southern Illinois University Carbondale.

Shin, J. H., and Jun, H. B. (2015). On condition-based maintenance policy. Journal of Computational Design and Engineering 2, 119–127.

Shoham, Y., (1993). Agent-oriented programming. Artificial Intelligence, 51-92.

Singh, Y., and Goel, B., (2007). A Step Towards Software Preventive Maintenance," ACM SIGSOFT Software Engineering Notes, 32(4), 1-5.

Sivakumar, N., V and ivekanandan, K., (2012). Measures for testing the reactivity property of a software agent. Int. J. Adv. Res. Artif. Intell. 1(9), 26–33.

So, Y., and Durfee, E. H., (1993). An organizational self-design model for organizational change. In Proceedings of the AAAI93 Workshop on AI and Theories of Groups and Oranizations, 8–15.

Stephen, H. K., (2002). Metrics and Models in Software Quality Engineering. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition.

Sun, P., and Wang, X., (2012). Application of ant colony optimization in preventive software maintenance policy, in: Proceedings of IEEE International Conference on Information Science and Technology, Cihna.

Tahir, A., Ahmad, R., and Kasirun, K. M. (2010). Maintainability dynamic metrics data collection based on aspect-oriented technology. Malays. J. Comput. Sci., 23(3), 177.

Tambe, M., (1997). Towards flexible teamwork, Journal of Artificial Intelligence Research, 7, 83–124.

Tchoffa, D., and El Mhamedi, A., (2012). Decision aid in software maintenance, Proceedings of the 14th IFAC Symposium on Information Control Problems in Manufacturing, Bucharest, Romania, 23-25.

Tinnemeier, N., Dastani, M., and Meyer, J. J., (2010). Programming norm change. In: Proc. of the 9th Int. Conference on Autonomous Agents and Multiagent Systems, 957–964.

Ye, Z. S., and Xie, M., (2015). Stochastic modelling and analysis of degradation for highly reliable products, Applied Stochastic Models in Business and Industry, 31, 16-32.

Yoo, M. J., and Briot, J. P., (1999). Une approche componentielle pour la modélisation d'agents coopératifs et leur validation. Rapport technique Laboratoire d'Informatique de Paris 6 (LIP6).

Vaidyanathan, K., and Trivedi, K. S. (1999). A measurement-based model for estimation of resource exhaustion inoperational software systems, In Proc. of the 10th IEEE International Symposium on Software Reliability Engineering, Boca Raton, FL, 84–93.

Vaidyanathan, K., Dharmaraja, S., and Trivedi, K. S., (2002). Analysis of Inspection-Based Preventive Maintenance in Operational Software Systems, in: Proceedings of 21st IEEE Symposium on Reliable Distributed Systems, 286–295.

Valetto, G. Kaiser, G., and Gaurav, S. K., (2001). A mobile agent approach to process-based dynamic adaptation of complex software systems, in 8th European Workshop on Software Process Technology, 102–116.

Van Noortwijk, J. M., Kok, M., and Cooke, R. M., (1997). Optimal maintenance decisions for the sea-bed protection of the eastern-scheldt barrier. European Journal of Operational Research, 82(2), 25–56.

Von Mayrhauser, A., and Wang, J., (1999). Experience with Reverse Architecture Approach to Increase Understanding. Proceedings for Intenational Conference on Software Maintenance, August 30 – September 13, Oxford, England, 1, 31-138.

Weyns, D., Malek, S., and Andersson, J., (2010). FORMS: a formal reference model for self-adaptation. In Proceedings of the 7th International Conference on Autonomic Computing, 205–214.

Weyns, D., Haesevoets, R., Helleboogh, A., Holvoet, T., J and oosen, W. (2010). The MACODO middleware for Context-Driven Dynamic Agent Organzations. ACMTransaction on Autonomous and Adaptive Systems.

White, J., (1994). Telescript technology: The foundation for the electronic marketplace. White paper, General Magic, Inc., 2465 Latham Street, Mountain View, CA 94040.

Wittwer, J. W., (2009). Fishbone Diagram/Cause and Effect Diagram in Excel, From Vertex42.com. Oct 29, https://www.vertex42.com/ExcelTemplates/fishbone-diagram.html.

Wooldridge, M., and Jennings, N. R., (1995). Agent Theories, Architectures, and Languages: A Survey. International Workshop on Agent Theories, Architectures, and Languages, 1–39.

Wooldridge, M., Jennings, N. R., and Kinny, D. (2000). The Gaia methodology for agent-oriented analysis and design. Autonomous Agents and Multi-Agent Systems, 3(3), 285-312.

Wooldridge, M. (2002). Intelligent Agents: The Key Concepts. Proceedings of the 9th ECCAI-ACAI/EASSS 2001, AEMAS 2001, HoloMAS 2001 on Multi-Agent-Systems and Applications II-Selected Revised Papers, 3-43.

Wooldridge, M. (2002). An Introduction to Multiagent Systems. London: John Wiley & Sons.

Wooldridge, M. (2009). An Introduction to MultiAgent Systems. WILEY Publisher 2nd Edition. Chichester.

Xie, T., Zhao, J., Marinov, D., and Notkin, D., (2006). Detecting Redundant Unit Tests for Aspect J Programs, in: Proceedings of 17th International Symposium on Software Reliability Engineering, 179–190.

Zakaria, A., Hosny, H., (2003). Metrics for aspect-oriented software design. In: Workshop on Aspect-Oriented Modeling, Boston.