Democratic and Popular Republic of Algeria

Ministry of Higher Education and Scientific Research

LARBI Ben M'HIDI University
-Oum El Bouaghi-

Faculty of Exact Sciences and Sciences of Nature and Life
Department of Mathematics and Computer Science

N ° d'ordre : ……………………….

Série : ……………………..

A Perfective Maintenance Approach For Multi-Agent Systems

Thesis presented by:

ZERROUGUI Salim

To obtain the degree of

DOCTOR of SCIENCE in Computing Science

(Option: **Artificial Intelligence and Imagery**)

Presented publically in: 10th Nov, 2020

Before the jury composed of:

President	Dr. GUERRAM Tahar	*MCA*	Oum El Bouaghi University
Supervisor	Pr. MOKHATI Farid	*Professor*	Oum El Bouaghi University
Co-Supervisor	Pr. BABAHENINI Mohamed chaouki	*Professor*	Biskra University
Examiner	Pr. MAAMRI Ramdane	*Professor*	Constantine 2 University
Examiner	Dr. DJEFFAL Abdelhamid	*MCA*	Biskra University
Examiner	Dr. MARIR Toufik	*MCA*	Oum El Bouaghi University
Invited	Pr. BADRI Mourad	*Professor*	UQTR - CANADA

2019-2020

بسم الله الرحمن الرحيم

Acknowledgments

First and foremost, praises and thanks to ALLAH the Almighty

*I would like to express my deep and sincere gratitude to my thesis supervisor **Pr. MOKHATI Farid.** There are not enough words of appreciation to give him the thanks he deserves. This work would never see the light without him. Since my first day as his student to this day he was always there for me. I consider myself very lucky to have such mentor and role model.*

I thank you professor from the bottom of my heart

*A special appreciation and gratitude to **Pr. BADRI Mourad**, who has enriched this work with his knowledge and experience. Also, his modesty, generosity and personality are such an inspiration.*

*A special appreciation and gratitude to **Pr. BABAHENINI Mohamed Chaouki** my thesis co-supervisor for his professional engagement since my first PhD year in Biskra University*

I would like to thank all the members of the Jury,

Dr. GUERRAM Tahar

Pr. MAAMRI Ramdane

Dr. DJEFFAL Abdelhamid

Dr. MARIR Toufik

For the great honor they have done me by agreeing to examine this work and to provide constructive criticism and valiant comments.

Finally, I cannot finish without expressing my appreciation to my best colleagues ***Dr*. Kalache ayyoub** and **Dr. Chebout Mohamed seddik**.

Salim.

Dedication

In memory of my father

To my beloved mother

To my sisters : Nassima, Maissa, Soulef, Nadjette, Fatima Zohra and Sabrina

& to my nephews

Salim.

ملخص

البحث المقدم في هذه الأطروحة يندرج في اطار هندسة البرمجيات الموجهة للوكلاء. مساهمتنا في هذا البحث متعلقة بإعادة هندسة وتحسين الأنظمة متعددة الوكلاء. مجال لم يتم استكشافه بما فيه الكفاية على الرغم من أهميته. الدافع الرئيسي لمساهماتنا هو اقتراح طرق الصيانة التحسينية للأنظمة متعددة الوكلاء لتعزيز خصوصياتهم واهتماماتهم. من خلال تقييم أداء الأنظمة متعددة الوكلاء وقابليتها للتوسع عند اتساع نطاقها و زيادة عدد الوكلاء بها، مما يمكننا من اكتشاف العديد من الاختناقات والاختلالات. خلال دراستنا تم العثور على خاصيتين مهمتين من خصائص أنظمة متعددة الوكلاء التي تعاني من مشاكل الأداء في الأنظمة واسعة النطاق هي خدمة الدليل المعروفة أيضًا باسم التوفيق أو خدمة الصفحات الصفراء، و القدرة الاجتماعية للوكيل أثناء بروتوكولات التفاعل. المناهج المقترحة تعيد تصميم هذه الاهتمامات باستخدام البرمجة جانبية التوجه لتحقيق لامركزية خدمة الدليل، وخوارزميات التعلم الآلي لتوسيع بروتوكولات التفاعل التي تزيد من التفاعلات الناجحة وتحسن القدرة الاجتماعية للوكيل. تم تنفيذ عمليات التحقق من الأساليب المقترحة باستخدام دراسات حالة و بيئات تنفيذ مختلفة مع سيناريوهات متعددة على منصة JADE. النتائج واعدة للغاية وتظهر تحسنًا كبيرًا في الخصائص التي تم فحصها.

الكلمات المفتاحية: الصيانة التحسينية ؛أنظمة متعددة الوكلاء ؛ الأداء والقابلية للتطوير ؛ خدمة الدليل؛ بروتوكولات التفاعل ؛ القدرة الاجتماعية.

Abstract

The research presented in this thesis takes place in the context of agent-oriented software engineering. Our contributions relate to reengineering and enhancing Multi-Agent Systems. This area is not yet sufficiently explored despite its importance. The main motivation of our contributions is to propose perfective maintenance approaches for multi-agent systems to enhance their specificities and concerns. By evaluating the performance and the scalability of multi-agent systems when systems scale up we can detect many bottlenecks and anomalies. In our study, two important multi-agent systems properties were found to be suffering from performance issues in large scale systems which are the directory service, also known as the matchmaking or the Yellow Pages service, and the agent's sociability during interaction protocols. The proposed approaches reengineer the implementations of these concerns using aspect-oriented programming to decentralize the directory service, and machine learning algorithms to extend the interaction protocols which increase the successful interactions and improve the agent's sociability. The validations of the proposed approaches were performed using different case studies, execution environments and multiple scenarios under JADE platform. The results are very promising and show a great improvement of the investigated properties.

Keywords: Perfective Maintenance; Multi-agent Systems; Performance and Scalability; Directory Service; Interaction Protocols; Sociability.

Résumé

Le travail présenté dans cette thèse se place dans le contexte de l'ingénierie logicielle orientée agent. Nos contributions concernent la réingénierie et l'amélioration des systèmes multi-agents. Ce domaine n'est pas encore suffisamment exploré malgré son importance. La motivation principale de nos contributions est de proposer des approches de maintenance perfectives pour les systèmes multi-agents afin d’améliorer leurs spécificités et préoccupations. En évaluant les performances et l'évolutivité des systèmes multi-agents lorsque les systèmes évoluent, nous pouvons détecter de nombreux goulots d'étranglement et anomalies. Dans notre étude, deux propriétés importantes des systèmes multi-agents se sont révélées souffrir de problèmes de performances dans les systèmes à grande échelle : le service d'annuaire connu également sous le nom de Matchmaking ou service de Pages Jaunes, et la sociabilité de l'agent lors des protocoles d'interaction. Les approches proposées remodèlent ces préoccupations en utilisant la programmation orientée aspect pour décentraliser le service d'annuaire, et des algorithmes d'apprentissage automatique pour étendre les protocoles d'interaction, ce qui augmente les interactions réussies et améliore la sociabilité de l'agent. Les validations des approches proposées ont été effectuées à l'aide de différentes études de cas, environnements d'exécution et multiples scénarios sous la plateforme JADE. Les résultats sont très prometteurs et montrent une grande amélioration des propriétés étudiées.

Mots-clés: Maintenance Perfective; Systèmes Multi-agents; Performance et Évolutivité; Service d'Annuaire; Protocoles d'interaction; Sociabilité.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACC	Agent Communication Channel
ACL	Agent Communication Language
ACMAS	Agent-Centered Multi-Agent Systems
AID	Agent Identifier
AJDT	AspectJ Development Tool
AMS	Agent Management System
AOP	Aspect-Oriented Programming
APDF	Agent-Platform Directory Facilitator
API	Application Programming Interface
ARA	Average Number Of Roles Played By Agent
AUC	Area Under The Curve
BG	Behaviour Granularity
CBDF	Category-Based Directory Facilitator
CDA	Capability Dissemination Agent
CFP	Call For Proposal
CH	Choice Of Homophily
CHN	Choice Homophily Navigation
CNCP	Contract Net With Confirmation Protocol
CPPS	Cyber Physical Production Systems
CPU	Central Processing Unit
DB	Data Base
DBMS	Data Base Management System
DF	Directory Facilitator
DoA	Degree Of Availability
FIPA	Foundation For Intelligent Physical Agents
FN	False Negatives
FP	False Positives
FPR	False Positive Rate
GAA	Goal Achievement Acceleration
GAUR	Goal Achievement By Using Resources
GUI	Graphical User Interface
I/O	Input / Output
IBL	Instance-Based Learning
IEC	International Electrotechnical Commission
IEEE	Institute Of Electrical And Electronics Engineers
IIOP	Internet Inter-Orb Protocol
IoE	Internet Of Everything
IoT	Internet Of Things
ISO	International Organization For Standardization
JADE	Java Agent Development Framework
JDK	Development Kit - Wikipedia
JVM	Java Virtual Machine
KG	Knowledge Granularity
KQML	Knowledge And Query Manipulate Language
LDAP	Lightweight Directory Access Protocol
LOC	Line Of Code

LSMAS	Large-Scale Multi-Agent System
MAS	Multi-Agent Systems
ML	Machine Learning
MMOG	Massively Multiplayer Online Gaming
MMORPG	Massively Multiplayer Online Role-Playing Games
OCMAS	Organization-Centered Multi-Agent Systems
PPV	Positive Predicted Value
RAM	Random Access Memory
RAP	Ratio Of Achieved Purpose
RIU	Ratio Of Interaction Utility
RL	Reinforcement Learning
RLRSs	Ratio Of The Lack Of Requesting Services
ROC	Receiver Operating Characteristic
RRA	Ratio Of Resources Availability
RSC	Ratio Of States Changes
RTT	Round Trip Time
RU	Ratio Of Understand-Ability
SGD	Stochastic Gradient Descent
SMO	Sequential Minimal Optimization
SVM	Support Vector Machine
TAP	Time To Achieve Purpose
TN	True Negatives
TP	True Positives
TPR	True Positive Rate
TRC	Time To Respond To Changes
TTL	Time To Live

General Introduction

General Introduction

1 General Context and Problem Statement

During the last decade, the development of multi-agent systems (MAS) has aroused great interest in the research community. The agent paradigm offers, in fact, undeniable advantages in the development of software systems that can operate in complex, distributed, heterogeneous and open environments. Several MAS development methodologies have been proposed in the literature (Zambonelli et al., 2003) (DeLoach, 1999). The main objective is to facilitate and support the development of agent-based applications. These methodologies have allowed real progress in the design and implementation of MAS. However, maintenance of agent-based applications is not yet sufficiently addressed (Kraetzschmar, 1997).

Software maintenance is still the most complex and expensive phase of agent-based applications life cycle. Maintaining existing source code is more economical than rewriting or replacing it with a new one (Jarzabek, 2007) (Alkhatib, 1992) (Lientz and Swanson, 1981) (Boehm and others, 1981). If the maintenance activity of procedural and object-oriented programs (Capretz and Capretz, 1996) (Kamiya et al., 2001) was the subject of several research projects, it is not the case for multi-agent applications, which have their proper specificities like deliberation, communication, etc. Although MAS represent an active research area, agent-based applications maintenance remains preliminary and has not yet achieved a high level of maturity (Reuss and Althoff, 2014) (Hao and Leung, 2012) (Khosravifar et al., 2009). Indeed, there are still various crucial issues that have not yet been addressed.

Enhancing the scalability and the performance of Multi-agent systems (Lee et al., 1998) has been the interest of many researchers even recently by focusing on improving agent properties like deliberation (Rens and Moodley, 2017) (Yao and Logan, 2016) (Faccin and Nunes, 2015), cooperation (Wang et al., 2016), communication (Kadera and Novák, 2017), reactivity (Dam et al., 2013), etc. Unfortunately, these researches are not enough, only few enhancement approaches were proposed for a given agent concern.

Perfective maintenance is a very challenging task which requires a lot of empirical evaluations to be performed. The first challenge is that in order to enhance a concern, performance problems must be found first. This is performed by systems' profiling under different situations, execution scenarios and execution environments. If a problem is found then these evaluations can take a very long time to finish due to long execution times, waiting times, blocked situations, etc. The second challenge is to propose an adequate solution that significantly improves the evaluated concern and does not negatively affect the others. A third challenge is that the solution could be hard to implement and only applicable to a given application. Therefore, it is preferable for a solution to target more systems where it can be easily implemented and reused.

In multi-agent systems, some properties are more important and more frequently used than others. Some of them have an effect on the system's overall performance. Therefore, targeting these properties is a big priority. This priority varies and depends on the type of the application and the used agent platform. Even more, if we like our enhancements to be applied on more systems then it is best to target design or code patterns for a given agent platform. A design pattern is a solution or a code implementation for a recurring concern which is usable for different systems developed using the same platform.

One of these patterns is the Yellow Pages service (Kuokka and Harada, 1995) (directory or matchmaking service). It is a very important service for agents to be able to interact with each other. Many researchers have highlighted the importance of a scalable and efficient Yellow Pages service in many fields (Pascoa et al., 2017) (Kadera and Novák, 2017) (Del Val et al., 2014). In this service, the more the number of agents is high, the more performance degradation will result, due to the overload of requests sent to central Yellow Pages service, which cannot handle these requests especially on high frequency. This is a major problem and can lead even to a total system breakdown.

Another important concern is related to agent interactions using interaction protocols. These latter are possible communication scenarios or patterns between agents in multi-agent systems. FIPA offers specifications for these interaction protocols like Contract Net, English-auction, request, etc. Interaction protocols are much known in the agent community. Their implementations can be offered by the agent development platform. Therefore, any improvement in their design or implementation can target a wide range of applications. Some researchers have reported performance and scalability problems especially with the Contract-net protocol. When the system scales up with a high number of agents problems start to emerge (Deshpande et al., 2005). Agents become busier and their messages queues can be saturated, which may cause in interactions failures. Agent's sociability is very affected by such scenarios. If an agent cannot reply to interaction requests in their specified times then the agent's sociability will be decreased which wastes valuable chances in reaching the best goals.

In the literature, researchers have proposed several methods as general ideas to improve the performance and the scalability of multi-agent systems. Some of them are applied on agent's level (Aguirre et al., 2000), while others are applied on the system's level (Brazier et al., 2001) where each method addresses a specified problem type. For example, replicating centralized services that suffer from bottleneck problems can reduce their load (Brazier et al., 2001)(Hussain and Shabbir, 2008). Large number of agents or data can be distributed on more environments for better resources management (Brazier et al., 2001). Agents can perform other tasks while they are waiting for messages in systems with high communication latencies (Brazier et al., 2001), etc. It is very important to carefully design such enhancements for better performance, minimal maintenance costs/effort and easier documentation and reuse. The only way in our knowledge to achieve such design is by using aspect-oriented programming. Enhancements can be implemented in separate modules called aspects which can introduce more attributes and functions in the original system.

Large-scale multi-agent systems are more vulnerable for performance and scalability problems than any other systems. They are generally measured by response times, maximum number of agents, tasks or messages that can be handled by the system. Agent platform's developers usually implement mechanisms to prevent system crashes by imposing timeouts and capacity limits. In such large-scale MASs, concerns must be designed differently to handle the load as much as possible. Agents must be well designed, coordinated and scheduled to exploit the shared environment resources effectively. Even more, enriching agents with more capabilities can enhance some of their properties. Learning capabilities have been used in the literature to aide agents in making decisions. It is relatively easy to be implemented and integrated by using exiting learning packages like Weka (Holmes et al., 1994) and Orange (Demšar et al., 2013).

2 Contributions

As a starting point for proposing perfective maintenance approach for multi-agent systems, a performance and scalability problem had to be found first. Therefore, to obtain that, we have relied on: (1) the multi-agent systems' problems and their available solutions reported in the literature, (2) the agent development community in form of performance problems reported in mailing lists and forums, (3) the results of an extensive and exhaustive profiling and evaluations of various components in JADE platform, and (4) the importance of the agent's concern or property in multi-agent systems development. Our research has resulted in two known, important and widely used multi-agent systems' concerns which suffer from performance and scalability problems when systems scale up. The first concern is the directory service (Matchmaker) in JADE platform known as of the directory facilitator agent (DF), and the second is the agents' sociability during interaction protocols.

A Novel Approach for Improving the Matchmaker Service (Zerrougui et al., 2017)

In this contribution, we have focused on the Yellow Pages service which is very important for agents to be able to locate and interact with each other. The first task was its evaluation in JADE platform. The results of the evaluation show that the more the number of agents is high, the more performance degradation will result, due to the overload of requests sent to a central directory facilitator (DF) agent. This issue did not get much attention from researchers despite its importance. The second part was proposing an approach to solve this scalability issue. The solution was based on the aspect-oriented paradigm. An aspect was used to play the roles of a replicator and a distributor of requests to several new Yellow Pages. The use of aspect-oriented programming simplifies the implementation, reduces the maintenance effort and offers a good reuse of the proposed solution. The approach was validated on three JADE applications and evaluated on five different machines. Results, which are very promising, show a significant scalability and performance improvement of the Yellow Pages service especially on Linux operating system.

Enhancing Agent's Sociability: A Perfective Maintenance Approach

In this contribution, we have focused on evaluating and enhancing agent's sociability, which is one of the most important agent characteristics. Sociability can be measured using exchanged messages or interactions which are usually implemented using interaction protocols. These latter are commonly used in agent-based systems, such as auctions, negotiations, tasks allocation and simulation systems. They insure a good coordination between agents by proposing a certain pattern of message exchange. However, these interaction protocols are not perfect; they need more extensions to offer, among others, better performance and scalability. One of the interaction protocols' limitations is when a deadline is imposed. In this case, participants may not have enough time to answer to some requests before their deadlines, due to overload, bottlenecks, slow network, being busy or blocked. Designing agents without taking this issue into consideration may decrease their sociability, which wastes valuable chances to obtain the best goals.

The proposed approach uses the participant's experience to train several machine learning algorithms to predict future outcomes. The trained algorithms are then validated using the 10-fold cross-validation method to obtain the best model. The selected model is used to help the participant to immediately send replies which are predicted to reach the initiators before their deadlines or to postpone the requests if they are predicted otherwise. The proposed approach has been evaluated using multiple Contract-Net interaction scenarios of two case studies under JADE platform. The results which are very promising show a significant increase in agents' sociability, by being able to successfully respond to more requests before their deadlines especially when systems scale up.

3 Thesis Plan

This thesis is organized in six chapters. Since that our two contributions are independent from each other, their common backgrounds are presented in the first two chapters. The third chapter presents an overview of the used tools and techniques. The 4th chapter concerns the relevant state of the art for our contributions. The 5th chapter presents the proposed approach for perfective maintenance of the Yellow Pages service. The 6th chapter presents the proposed approach to enhance agent's sociability during interaction protocols.

- **Chapter 1 - Multi-agent systems:** In this chapter, general concepts related to agents and multi-agent systems will be presented. Starting by their definitions and characteristics followed by the concepts of agent's sociability, FIPA, design patters, Yellow Pages, and interaction protocols. Next, some large-scale multi-agent systems applications will be presented. Finally, JADE platform will be presented along with its library used to interact with the DF and implement Contract-net protocol interactions.
- **Chapter 2 - Software maintenance:** In this chapter, we will present the software maintenance definition including its four types and the available models to perform

this process. Next, the important concepts of reverse, reengineering and refactoring will be explained followed by the performance and scalability terminologies. The software maintenance of multi-agent systems will be addressed in terms of their specific performance metrics and profiling tools. Finally, the available techniques in the literature that can be used to enhance multi-agent systems will be presented.

- **Chapter 3 - Overview of used Tools and Techniques:** In this Chapter, we will present the essential concepts related to the tools and techniques used in the proposed approaches. First, the aspect-oriented programming along with the AspectJ tool. Second, machine learning and the Weka toolkit.
- **Chapter 4 - State of the Art:** In this chapter, we will present the state of the art related to performance enhancements of the Yellow Pages service (Matchmaking) and the Contract-net protocol interactions. We will also provide brief comparisons between related works and our proposals.
- **Chapter 5 - A Novel Approach for Improving the Matchmaker Service:** In this chapter, we will present a performance evaluation of the JADE's DF followed by the proposed approach to enhance it along with its validation. The results of the proposed approach will be presented, discussed and compared to the original JADE's DF agent.
- **Chapter 6 - Enhancing Agent's Sociability: A Perfective Maintenance Approach:** In this chapter, we will discuss the problem statement related to agent's sociability emerged during interaction protocols. Afterwards, the proposed approach to enhance agent's sociability during a Contract-Net protocol by extending agents with learning capabilities will be presented along with its validation. Finally, the results of the proposed approach will be discussed and compared to the original Contract-Net protocol in JADE platform.

Chapter 01

Multi-Agent Systems

Chapter 1 Multi-agent Systems

1 Introduction

A Multi-agent system (MAS) is a part of the distributed artificial intelligence research domain. It consists on several interconnected entities called agents to solve problems in a distributed manner. MASs bring new concepts to the software engineering world like autonomy, pro-activity, sociability, etc. Agent's interaction is considered to be an essential aspect of multi-agent systems. In order to interact, agents must be able to locate each other according to their needs through matching mechanisms called matchmaking or directory facilities. This latter is a central *Yellow Pages* which contains lists of agents with their offered services. Once agents have known each other's locations, they can interact using simple messages exchange, or more preferably, using a predefined interaction protocol. Interaction protocols allow agents to expect certain responses depending on the conversation scenario. FIPA organization offers standard specifications for such systems to preserve interoperability across different environments.

In our thesis, we tend to propose perfective maintenance approaches for multi-agent systems. Such maintenance is generally required in large-scale multi-agent systems where performance and scalability problems appear due to the size of the systems or the lack of sufficient resources. Furthermore, in order for our approaches to target more multi-agent systems independently of their purpose or their implementation and not to be specific for a given case study, we must focus on the common, recurring and the important concepts of multi-agent systems. These concepts are design patterns and more precisely social patterns given the importance of the sociability characteristic for multi-agent systems like interaction protocols and matchmaking patterns.

In this chapter, we are going to introduce the essential concepts of multi-agent systems. Starting by defining agents, their types, characteristics and how they can constitute multi-agent systems. Next, we will discuss the agent's sociability concept, including FIPA agent communication language, FIPA interaction protocols and Matchmaking. We highlight the main social patterns used by developers for multi-agent systems engineering then we will briefly mention some application domains of large-scale multi-agent systems. Finally, a brief introduction to JADE agent Platform will be presented.

2 Multi-Agent System Concepts

To introduce the concepts on which the multi-agent systems were founded, we use the VOWELS (**Figure 1.1**) approach defined by Demazeau (Demazeau, 2001). His definition consists of four general concepts: *Agent, Environment, Interaction and Organization (A+E+I+O).*

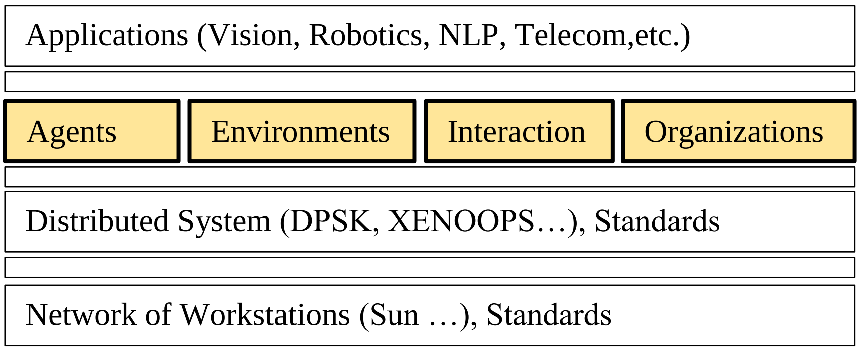

Figure 1.1 The VOWELS approach of Multi-agent systems (Demazeau, 1995)

2.1 Agents

What is an agent? This is the first frequently asked question in multi-agent systems research domain. Different fields have used the concept of agent as the object of study for a very long period (ex: computer science, robotics, psychology, biology, sociology, etc.). However, there is not a unique definition adopted by the community on the term *Agent*. In computer science, there are many variations of the term *Agent* depending on its application area e.g., intelligent agent, software agent, social agent, autonomous agent. Several definitions have been proposed in the literature, from where we cite the definition of Ferber and that of Wooldridge and Jennings.

2.1.1 *Ferber's Definition*

According to Jacques Ferber (Ferber, 1995), "*we call agent a physical or virtual entity which is:*

a. *able to act in an environment,*
b. *can communicate directly with other agents,*

c. *driven by a set of trends (in the form of individual objectives or a function of satisfaction, even survival, which it seeks to optimize),*
d. *has its own resources,*
e. *able to perceive (but in a limited way) its environment,*
f. *has only a partial representation of this environment (and possibly none),*
g. *has skills and offers services,*
h. *which can possibly reproduce, die and change state,*
i. *whose behavior tends to meet its objectives taking into account the resources and skills at its disposal, and according to its perception, of these representations of the communications it receives."*

2.1.2 ***Wooldridge's Definition***

Wooldridge and Jennings (Wooldridge and Jennings, 1995) have defined agents as follows:

"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives."

The authors have also distinguished between two notions of the term agent: weak and strong notion of agency (**Figure 1.2**). A weak agency is in which an agent is to denote a hardware or software system characterized with the following properties

- *Autonomy* - the agent is able to act without the intervention of another entity (human or agent) and controls its own actions as well as its internal state,
- *Sociability* - the agent must be able to interact with other agents (software or human) in order to accomplish tasks or help these agents to accomplish their own,
- *Reactivity* - the agent must be able to perceive its environment and develop a response to changes in the required time,
- *Pro-activeness* - the agent must exhibit proactive and opportunistic behavior, while being able to take the initiative at the right time.

As mentioned in Wooldridge definition, another important agent property is its situation in the environment in which agents can perceive and act (Wooldridge, 2002). In the contrary, Ferber (Ferber, 1995) considered that some agent types can run without been situated in an environment in the sense that they do not have any mechanism to perceive and act with the exterior such as the *purely communicating agents* (Marir, 2015).

A strong agency is in which an agent, has the weak agency properties enriched with concepts that are more usually applied to humans. These human-like properties can be mentalistic notions like knowledge, belief and intention (Shoham, 1993) or emotional notions (Bates et al., 1992) (Bates, 1994). Furthermore, presenting agents visually using icons or animated faces (Maes, 1994) is another way to give them human-like attributes which may be important for human-computer interfaces designers.

Other attributes of agency have been also added by Wooldridge which are:

- *Mobility* - which is the ability of an agent to move to other hosts in the network (White, 1994),
- *Veracity* - which means that the agent must not deliberately communicate false information (Galliers, 1988),
- *Benevolence* - is when an agent does not have any conflicting goals (Rosenschein, 1985),
- *Rationality* - is when the agent always act to achieve its goals and not otherwise (Galliers, 1988).

Agents can have either essential or optional properties depending on the application and the context of usage (Marir, 2015). In consequence, they can be distinguished from similar concepts such as objects and expert systems, especially when they have situation and sociability properties and reactive/proactive behaviors (Wooldridge, 2009).

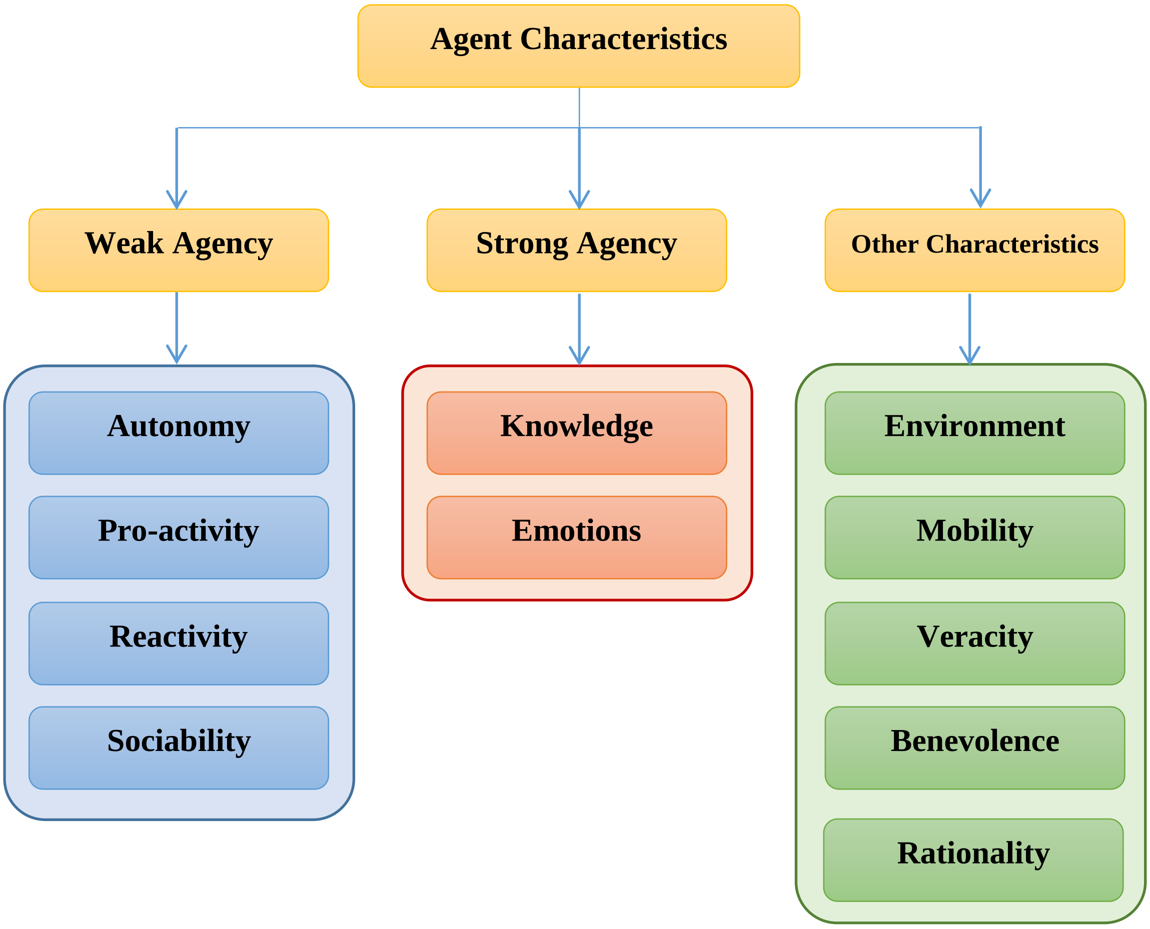

Figure 1.2 Agent's characteristics according to Wooldridge

2.1.3 *Agent Classification*

Classification is an important mechanism for understanding the different agent types. It depends on a given selected criteria on which the agents are based. The choice of classification criteria for a given taxonomy is arbitrary and it is based on a need to precisely name existing architectures according to their properties.

Situated and communicating agents are two agent types proposed by (Ferber, 1995) based on their environment. The mobility characteristic can be used also as a classification criterion which distinguishes between two agent types: mobile and fixed agents. The best known classification is based on the degree of intelligence (Savall, 2003) in which, agents can be classified into three essential categories: cognitive agents, reactive agents and hybrid agents.

2.1.3.1 Cognitive Agent

It is an agent which has an explicit representation of its environment. To achieve its objectives, it reasons on the state of the environment and chooses the right actions to execute (Meyer, 2005). Generally, a cognitive system includes a small number of agents, each of which can be compared to a more or less complex expert system. In this case, we speak of a highly granular agent (execution and processing of complex code).

2.1.3.2 Reactive Agent

This type of agent is only able to perceive and act on the environment. They are agents that have no explicit representation of the environment and have no memory of its history. They work according to the stimulus / action mechanism. The stimulus is an element of the environment (action, message, situation, etc.). Systems of this type generally contain a large number of agents, which are characterized by low granularity (little code, simple processing). Their communication is generally simple by means of traces or signals (local modifications of the environment perceptible by the target agents).

2.1.3.3 Hybrid Agent

This type of agent combines reactive abilities with other cognitive skills. This achieves the benefits of both cognitive and reactive architectures simultaneously, while eliminating their limitations by combining the agent's proactive, goal-directed behavior with behavior reacting to changes in the environment. The organization of this combination is done by a layered architecture where each is a cognitive component with symbolic representation of knowledge and reasoning skills, or a reactive component.

2.2 Environment

According to the authors in (Weyns et al., 2005), the term environment has many different meanings, which caused a lot of confusion due to the different application domains of MAS. The authors did a survey on the available environments of MASs in which they stated that the term environment has been defined by researchers as "*(1) the logical entity of a MAS in which the agents and other objects/resources are embedded, (2) the software infrastructure on which the MAS is executed, (3) the underlying hardware infrastructure on which the MAS runs.*" To tackle this confusion, they proposed a 3-layers model for MASs to better understand what an environment is (**Figure 1.3**).

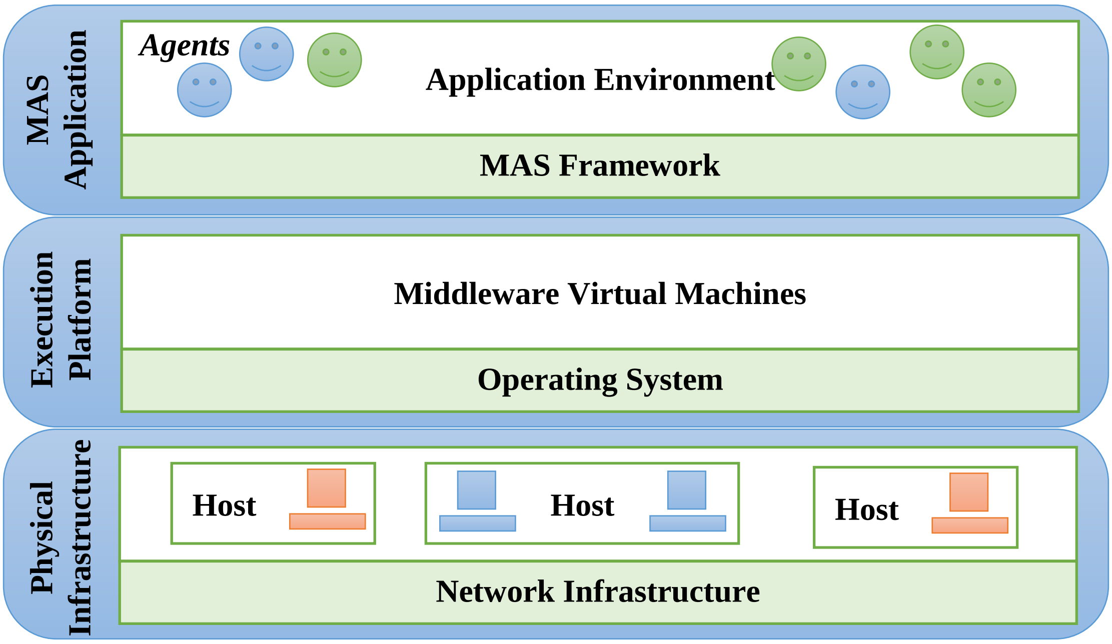

Figure 1.3 3-layer model for MASs' environment proposed by (Weyns et al., 2005)

The authors of (Weyns et al., 2005), have also concluded that the environment was not considered as a *first-class module.* A first-class module (environment) is an independent building block which implements its own functionalities in the MAS, regardless of the agents. These functionalities are often treated implicitly or in ad-hoc manner. According to (Tranier, 2007), there is a need to explicitly represent the problem to solve for agents, to define the perception which they may have of it and the actions which they can do in this representation in the application environment level. In (Mertens et al., 2004), the authors have also distinguished between the application environment in which corresponds to the modeling of the problem treated by the system (Ferber, 1999)(Weyns et al., 2003)(Odell et al., 2003), and the execution environment which corresponds to the execution platform (software and hardware infrastructures).

In (Russell and Norvig, 2009), the authors have discussed the way in which an agent relates to its environment, by perceiving its environment using sensors and acting upon the environment using effectors. They also presented some important properties of MAS environments which are now adopted by the agent community (Weyns et al., 2005):

- ✓ **Accessible or inaccessible**: indicates whether the full state of the environment can be accessed by the agents or not,

- ✓ **Deterministic or non-deterministic:** indicates whether the current environment state and the selected agents' actions can determine the next new state of the environment or not,
- ✓ **Static or dynamic:** indicates whether the environment can change while an agent deliberates or not,
- ✓ **Discrete or continuous:** indicates whether the number of percepts and actions are limited or not.

2.3 Interaction

In MAS, agents need to interact to achieve their goals. This is a basic and known concept in the agent paradigm. Some experts consider the interaction to be the reason for creating MASs in where agents influences the behaviors of each other (Ferber, 1995). Interaction does not only concern message exchange but it also appears in multiple forms or situations. Ferber (Ferber, 1995) has proposed a classification for the different interactions' forms which varies from simple to complex situations using three principal components of interaction: Goals compatibility, resources and competences sufficiency of agents. Goal compatibility refers to the possibility of achieving different agents' goals which benefits them. On the other hand, if two goals are incompatible, the agents can achieve only one goal. In addition, this classification differentiates between resources and competences. Resources refer to the different means used to accomplish a task. In contrast, competences are the attitudes of an agent to perform a task (Marir, 2016).

Table 1.1 Classification of interactions' situations (Ferber, 1995)

Goals	Resources	Competences	Situation type	Note
Compatible	Sufficient	Sufficient	Independence	Situation of indifference
Compatible	Sufficient	Insufficient	Simple collaboration	Cooperation situations
Compatible	Insufficient	Sufficient	Congestion	
Compatible	Insufficient	Insufficient	Coordinated collaboration	
Incompatible	Sufficient	Sufficient	Pure individual competition	Situations of antagonisms
Incompatible	Sufficient	Insufficient	pure collective competition	
Incompatible	Insufficient	Sufficient	individual conflicts for resources	
Incompatible	Insufficient	Insufficient	collective conflicts for resources	

Considering all the possible cases of the interaction components, eight situations have emerged. In the independence situation, agents do not need to interact because they have sufficient resources and competences. In the next three situations, the agents have compatible goals but insufficiency in one or both of resources and competences. These are cooperation situations and they are the simplest forms of interactions. The complications start with different degrees when agents have incompatible goals with insufficiency in one or both of resources and competences. If agents have sufficient resources regardless on their competences then they are in pure competition form of interaction. Now, if agents during the pure competition have sufficient competences then they compete individually, else they form groups and compete collectively. In the last two situations, agents with incompatible goals and insufficient resources are always in conflict situations in which they dispute individually if they have sufficient competences or collectively otherwise. From these interactions' situations, Ferber (Ferber, 1995) has considered three forms of interaction: collaboration, coordination and negotiation (Marir, 2015).

A visual representation of Ferber's classification of interactions' situations is presented in **Figure 1.4**. The inside of the circles represents the insufficient resources and competences in the red and blue colors respectively and the incompatible goals in yellow. The outside of each circle represents the other cases (compatible goals in the outside of the yellow circle along with the sufficient resources and competences). The more these circles intersect the more the interaction is complicated.

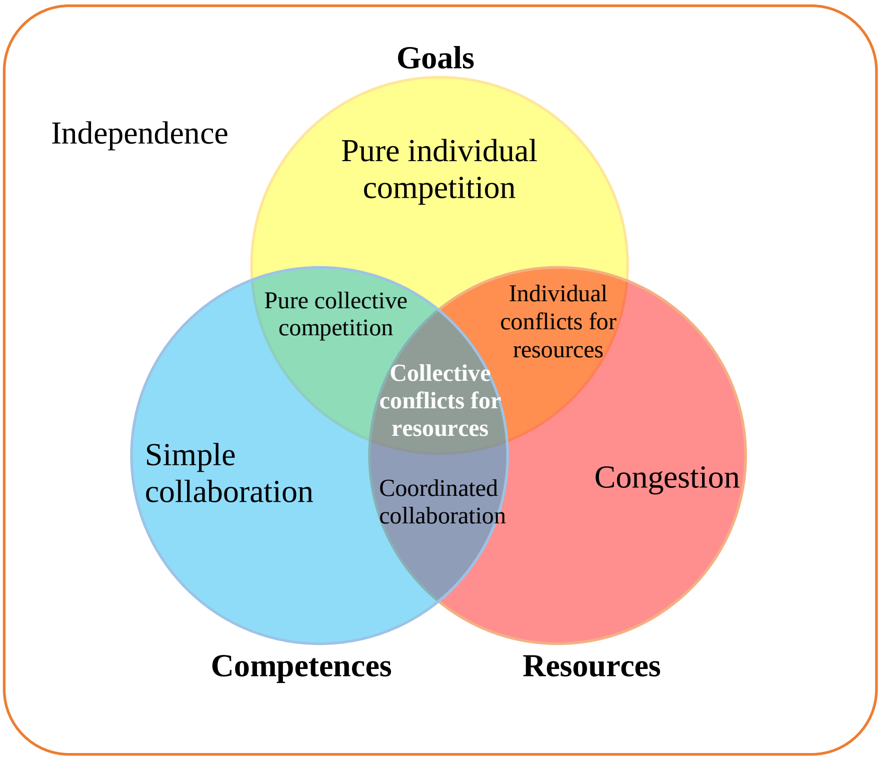

Figure 1.4 Ferber's classification of interactions' situations

2.4 Organization

In multi-agent systems, organization is a unity formed by a set of entities under a given order or structure which contributes to the coherence of the whole (Ferber, 1995). It can be addressed on two levels, micro level and macro level (Marir, 2015). The micro level organization also known as *agent-centered multi-agent systems* (ACMAS) (Ferber et al., 2004) focuses on the description of the internal structure of the agent without defining the collective structures of the system level, while the macro level organization also known as *organization-centered multi-agent systems* (OCMAS) (Ferber et al., 2004) is obtained using sociology concepts (Tranier, 2007).

According to Morin (Morin, 1977), an organization can be defined as *"an arrangement of relationships between components or individuals that produces a unit, or a system, endowed with qualities unknown at the component or individual level. Organization inter-relatively links various elements or events or individuals, which then become components of a whole. It assures solidarity and relative solidity, therefore assures the system a certain possibility of duration in spite of random disturbances*."

The problem in this definition is that the individual agents are considered as black boxes and their behaviors are unknown. Therefore, how is it possible to study the system's behavior without knowing agents' behaviors? In fact, in organizations, it does not matter how agents do their roles, what is important to know in an organization is which agent has certain role or a function (Tranier, 2007) (Marir, 2015).

In (Ferber, 1995), the author has introduced the organizational structure and the concrete organization in MAS. The organizational structure was described as the abstract characteristics of an organization in which agent classes endowed with roles and their relations. On the other hand, a concrete organization represents an instance of the organizational structure; where a set of roles instances are associated to a set of agents' instances. Consequently, the study of an organization must be done along three aspects:

- ✓ **The structural aspect:** is the topological structure (**Figure 1.5**) of the organization which describes the nature of the links between the different entities. Many structures have been proposed in the literature as: the hierarchical model in MAGIQUE (Bensaid and Mathieu, 1997), the groups-oriented model in AALAADIN (Ferber and Gutknecht, 1998), the holonic model, the peer-to-peer model in GNUTELLA (Ripeanu, 2001) and the broadcast model. (Secq, 2003)

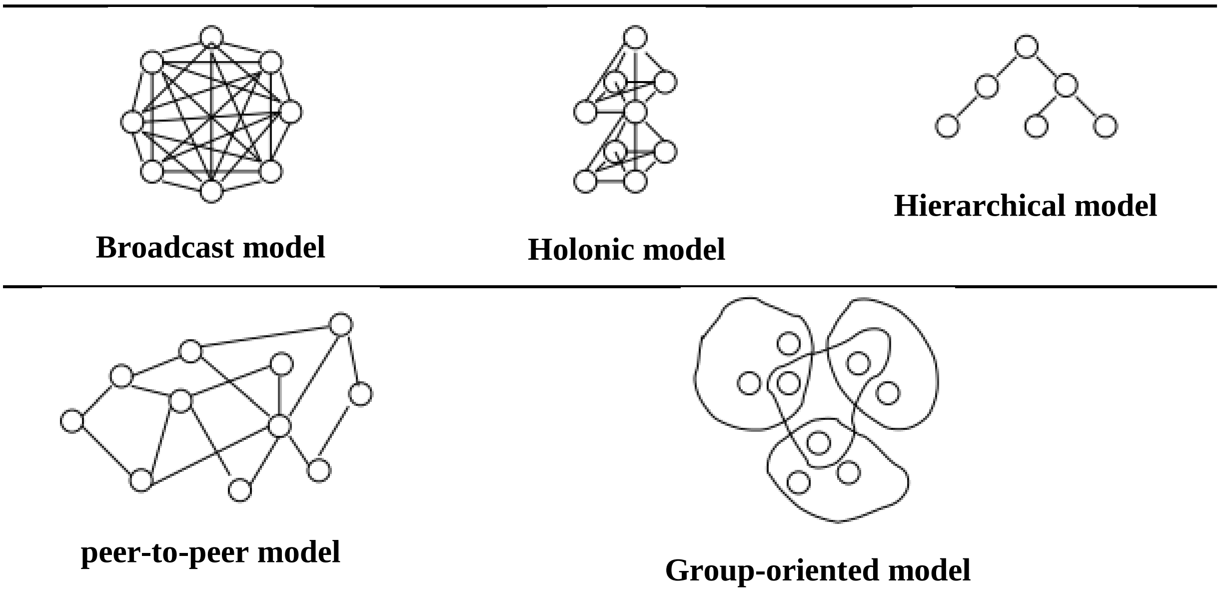

Figure 1.5 Different topologies for organization (Secq, 2003)

- ✓ **The functional aspect:** is about the functionalities that can be added to the organization's members. Therefore, some generic roles have been identified in the literature like the facilitators, brokers and mediators (**Figure 1.6**) (Shen and Norrie, 1999). A facilitator is responsible for local and distant agent groups' communications by routing outgoing messages to distant agents and transformation of incoming messages for local agents. The broker is facilitator with more functionality like supervision and notification which can be contacted by any agent in the system to find services or a particular agent. The mediator resumes the functionalities of the broker with more capabilities of coordinating agents (Secq, 2003).

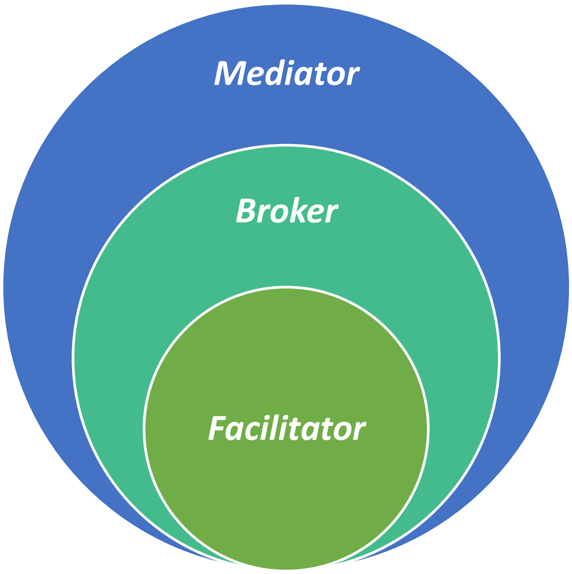

Figure 1.6 Facilitator, broker and mediator (Secq, 2003)

- ✓ **The concretization parameters:** determines the rules and standards that will drive the organization creation from the structural organization to the concrete organization. Two criteria are used by designers to assign roles to different agents: (1) specialization which means that an agent can specialize in only one role or generic with multiple roles, and (2) redundancy which means assigning one role to multiple agents (Marir, 2015).

3 Agent's Sociability

Agent's sociability is an important characteristic of intelligent agents. The authors in (Marir et al., 2016) have proposed a quality model for MAS in which they concluded that the sociability is the only sub-characteristic that affects all the characteristics of their quality model (functionality, reliability efficiency, etc.). The sociability means the capacity of an agent to interact via some kind of agent communication language (Wooldridge and Jennings, 1995). According to (Alonso et al., 2008), the agent's sociability is its ability to: (1) communicate by exchanging information with other agents and humans, (2) cooperate with other agents, and (3) negotiate.

3.1 Agent Communication

Communication is the base of agents' interaction. It is mostly known as a direct message exchange between agents. In fact, there is an indirect agents' communication type as well, performed by manipulating the environment to be perceived and interpreted by other agents (Marir, 2015). For direct communication, a unique communication language must be used by agents so they can understand each other. The best-known agent communication languages (ACL) have a lot on common and they are both based on speech act theory: Knowledge and Query Manipulate Language "*KQML"* (Finin et al., 1994) and *FIPA-ACL* (FIPA, 2001). Speech act theory (Searle, 1969) is derived from the linguistic analysis of human communication. It is based on the idea that with language the speaker not only makes statements, but also performs actions (Ahmad, 2002). Speech acts of conversations can be classified into five categories presented in **Table 1.2**: commissive, expressive, assertive, declarative, and directive.

Table 1.2 Speech acts types (Yule, 1996)(Marir et al., 2016)

Speech act type	Meaning	S = Speaker, X = Situation
Assertive	Giving information about the world by affirming something, such as: Thinking, Affirming, and Informing.	S believes X
Directive	Give directives to the recipient such as: Ask, Claim, Require.	S wants X
Commissive	Commit the speaker to perform certain actions in	S intends X

	the future such as: Promise, Guarantee and refuse.	
Expressive	Give the recipient indicators on the speaker's mental state such as: Congratulate, Excuse and Approve.	S feels X
Declarative	The speaker performs at the time of the utterance the action that says to accomplish such as: Stipulate, Declare.	S causes X

3.2 FIPA and Interaction Protocols

The Foundation for Intelligent Physical Agents (FIPA) (O'Brien and Nicol, 1998) is an international organization founded in 1996 to develop software specifications and standards for heterogeneous, communicating agents and agent-based systems. FIPA builds specifications that can be used to achieve interoperability between agent-based systems developed by different companies and organizations. FIPA was officially accepted by the IEEE as its eleventh standards committee in 2005 so that agent standards can be integrated with non-agent technologies. FIPA specifications are the first step towards an agent standard which focuses on specifying interfaces to support interoperability between agent systems. Therefore, four areas of agent standardization have been presented by FIPA (O'Brien and Nicol, 1998):

- ✓ **Agent communication:** interface to facilitate information exchange, negotiation and cooperation between agents by FIPA-ACL,
- ✓ **Agent management:** interface offers facilities for agents' creation, location, security and mobility,
- ✓ **Agent/software integration:** interface to support interaction between agents and non-agent software like legacy systems or hardware drivers, etc.,
- ✓ **Human/agent interaction:** interface to support interaction between human and agents.

A standardization of MAS architecture is needed for multi-agent systems development to build compatible systems that can be easily reused across projects. FIPA promotes a framework for MAS development, described by a reference model presented in **Figure 1.7**. The framework specifies the normative environment where agents reside and operate, and an agent platform which defines an infrastructure for the deployment and interaction of agents. The essential components of FIPA agent reference model are (Ahmad, 2002):

- ✓ **Agent Communication Channel (ACC):** This component plays the role of a message router. In order for agents to communicate they must access at least one ACC.
- ✓ **Agent Management System (AMS):** The AMS offers white-pages service to the agent system by maintaining a list of the registered agent identifier on the agent platform. It manages the agent platform and agents' creation, registration, deregistration and activities.
- ✓ **Directory Facilitator (DF):** also known as Matchmaker. This provides a Yellow Pages directory service in which agents can register their services and search for services offered by other agents.

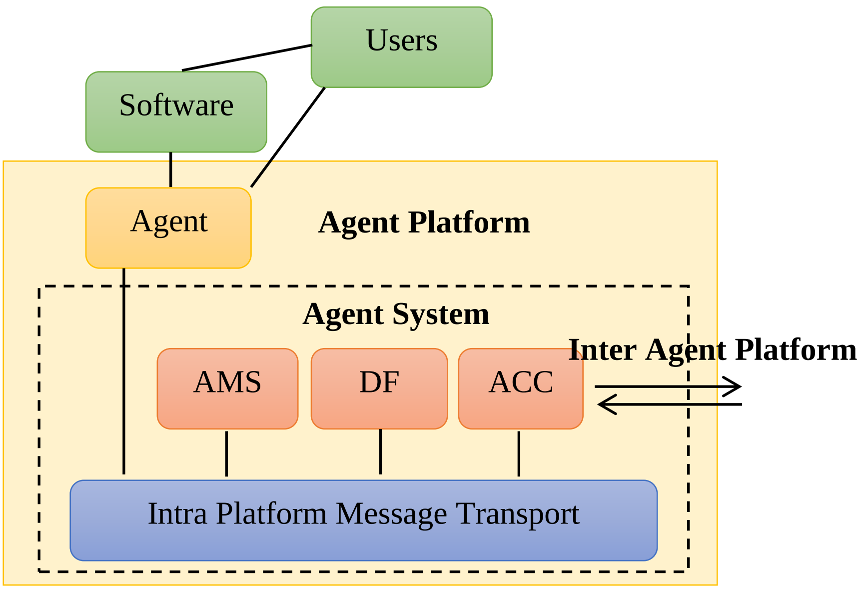

Figure 1.7 FIPA reference model of an agent platform (Ahmad, 2002)

3.2.1 *FIPA-ACL*

In order for agents to interact, they need a language powerful enough to express their needs. KQML was already the most used ACL at that time but it has its drawbacks which prohibited FIPA to consider it as a standard ACL. KQML did not have a formal definition for its semantics which made them open to different interpretations by the developers. When formal semantics was defined, it was not clear how widely this has been adopted by KQML users. Furthermore, commissives speech acts were not possible with KQML.

To overcome the limitations of KQML, FIPA proposed ARCOL a highly expressive agent communication language with well-defined semantics. The formality of ARCOL and the usability of KQML were combined later on to form the FIPA-ACL which consists of five levels (O'Brien and Nicol, 1998):

- ✓ **Protocol:** Define the social principles and rules for structuring inter-agent dialogues,
- ✓ **Communicative act:** defines the performed communication's type, like request, inform, etc.,
- ✓ **Messaging:** defines meta-information about the message like the sender, receiver, context, etc.,
- ✓ **Content language:** defines the language of the message's content,
- ✓ **Ontology:** specifies the vocabulary, concepts and terms' meaning used in the content.

FIPA-ACL is based on twenty-one communicative acts, expressed by performatives, which can be grouped according to their functionalities as follows (FIPA, 2002a):

- ✓ **Information transmission:** *Inform, Inform-if, Inform-ref, Confirm, Disconfirm,*
- ✓ **Information request:** *Query-if, Query-ref, Subscribe,*
- ✓ **Action request:** *Request, Request-when, Request-whenever,*
- ✓ **Negotiation:** *Accept-proposal, Cfp, Propose, Reject-proposal,*
- ✓ **Action management:** *Agree, Cancel, Refuse,*
- ✓ **Task delegation:** *Proxy,*
- ✓ **Error handling:** *Failure, Not-understood.*

A FIPA-ACL message can contain some or all of the elements described in **Table 1.3**. The elements necessary for conveying a message change depending on the situation. If an agent does not recognize or cannot process one or more items, then they can respond with the not-understood message.

Table 1.3 Elements of FIPA-ACL message (Jarraya, 2006)

Element	Signification
Performative	The type of communicative act
Sender	The sender of the message
Receiver	The recipient of the message
Reply-to	The participant in the act of communication
Content	The content of the message

Language	The language in which the content is represented
Encoding	Describes the encoding mode of the message content
Ontology	The name of the ontology used to give meaning to the terms used in the content
Protocol	The name of the interaction protocol,
Conversation-id	The conversation identification
Reply-with	An identifier of the message, for later reference
In-reply-to	It references the message to which the agent is responding (specified by the reply-with attribute, in the sender's previous message)
Reply-by	A delay to reply to the message (deadline)

Figure 1.8 shows an example of a FIPA-ACL message. Agent ***A*** informs its interlocutor that the price of an HP computer is 1500 $. The content of the message is expressed in the Prolog language. The ontology used is that of computers. This message is part of a conversation with the identifier Conv01. Agent ***B*** is constrained by 10 minutes to answer to this message (Jarraya, 2006).

```
(inform
        :sender A
        :receiver B
        :reply-with Pricing1
        :language Prolog
        :ontology Computer
        :content Price (HP,1500$)
        :conversation-id Conv01
        :reply-by 10 min)
```

Figure 1.8 An example of a FIPA-ACL message (Jarraya, 2006)

3.2.2 *Interaction Protocols*

An interaction protocol is a complete dialogue or conversation between agents represented by a sequence of messages. Agents can anticipate each other's messages according to a predefined conversation pattern. FIPA specified a number of standard interactions protocols such Contract Net, English-auction, Request, etc. **Figure 1.9** illustrates the FIPA-Request protocol which consists of an ordered exchange of communicative acts between two agents initiated by a request communicative act. The agent sending a request using this protocol can anticipate either a not-understood, a refuse or agree response (O'Brien and Nicol, 1998).

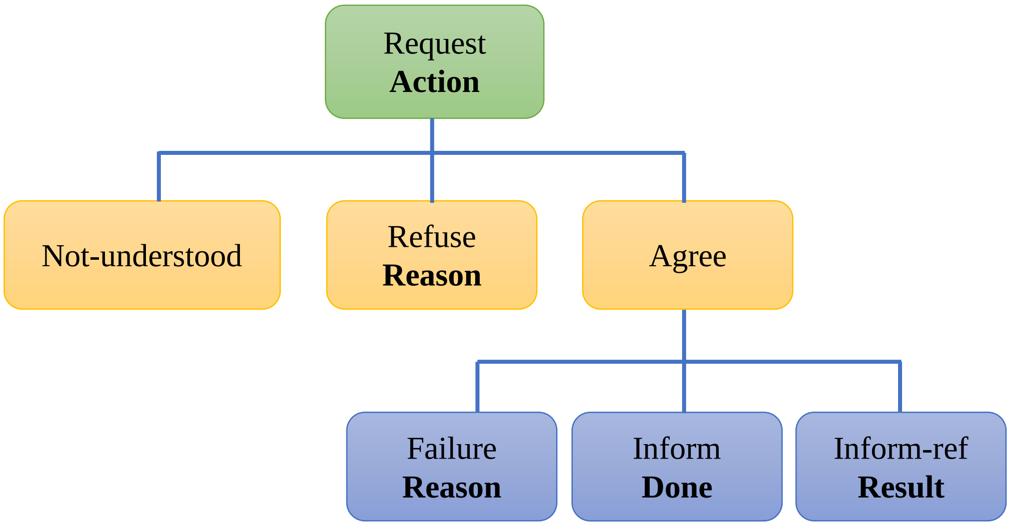

Figure 1.9 FIPA request protocol (O'Brien and Nicol, 1998)

3.2.3 *Contract-Net Protocol*

In multi-agent systems, the Contract-Net protocol (CNP) was introduced in (Smith, 1980) to coordinate agents and help tasks allocation among them. These agents have two different roles, initiators and participants (**Figure 1.10**). Initiators send call for proposals (CFP) to a number of participants and wait for their proposals. The participants answer to the CFP message with a reject or a propose message before a specified deadline. When the deadline reaches or all the participants have responded, the initiator evaluates the received proposals then it accepts the best one and rejects the others. At last, the participant can reply back with inform or failure message.

A problem that can arise with the contract-net interaction is when the best qualified participant for a given task is busy during its announcement. Therefore, the task can be assigned to a less qualified participant to solve it (Florea et al., 2002).

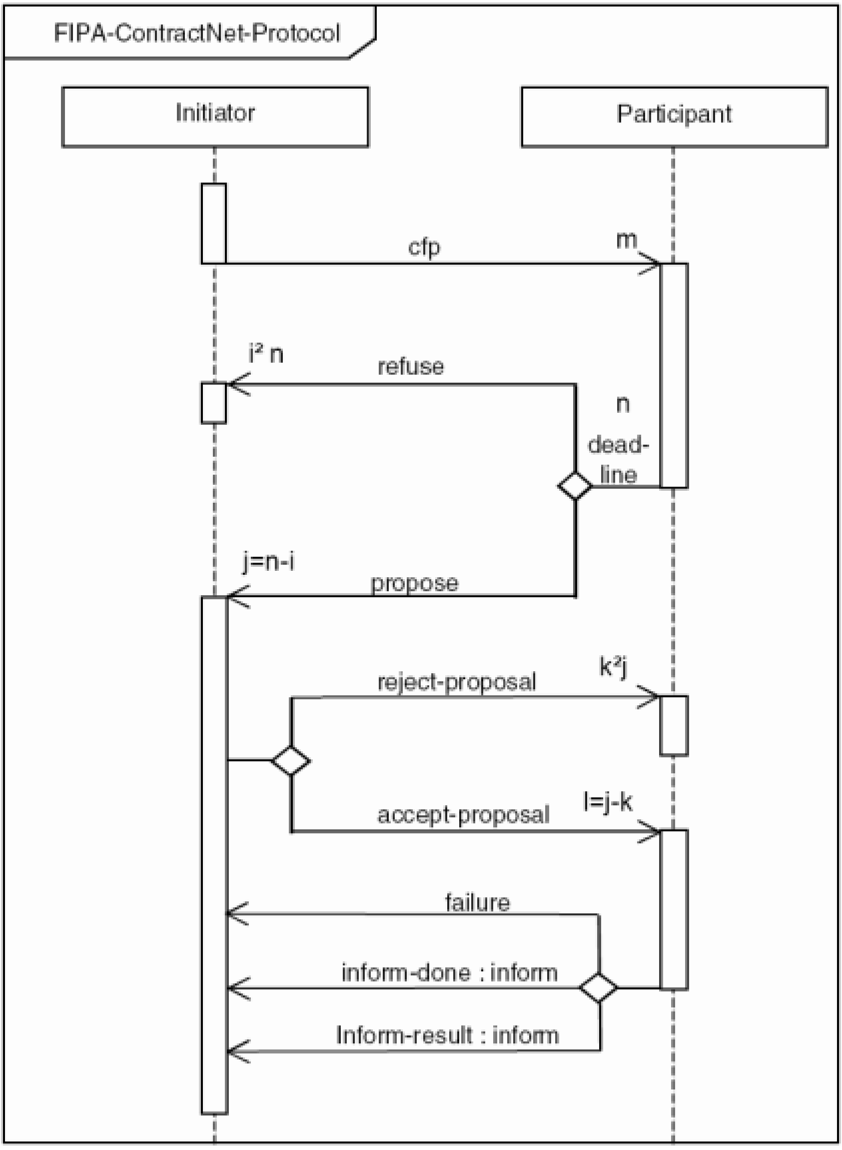

Figure 1.10 FIPA Contract-net protocol sequence diagram (FIPA, 2002b)

4 Design Patterns for Multi-agent Systems Engineering

Different systems can have similar sub-functions like database access, communication setups, security protocols, etc. These functions are already known solutions which are designed and implemented in the same way across different applications. Design Patterns (**Figure 1.11**) are explicit formulations of good experiences already used in software development that consists of well-tested solutions to recurring problems that arise in some systems. It can also be defined as a characteristic arrangement of modules, recognized as good practice in response to a software design problem (Do et al., 2005). It describes a standard solution, usable in the design of different software.

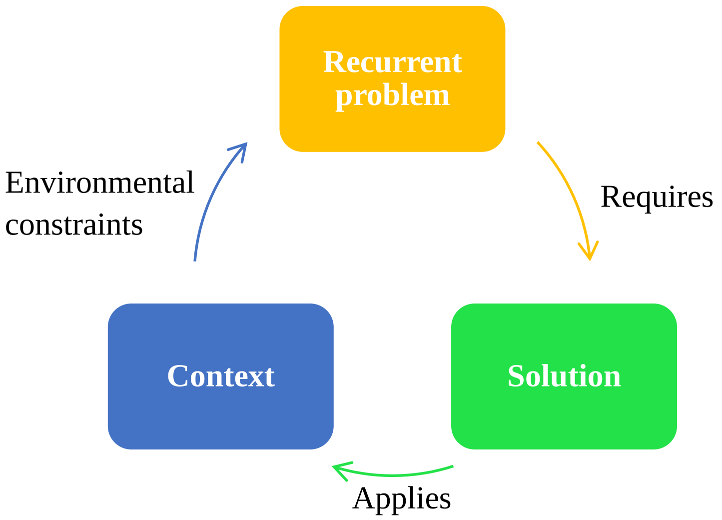

Figure 1.11 Pattern anatomy (Growin et al., 2014)

Patterns provide easy reuse of good software design. This concept has also become essential for the development of large-scale multi-agent software at low cost by promoting reuse (Tahara et al., 1999).

The authors in (Aridor and Lange, 1998) have proposed a classification for design patterns found in mobile agent applications. They have classified three designs patterns groups: *traveling*, *task* and *interaction* patterns (**Figure 1.12**). The traveling patterns are essential for mobile agents in which they manage their movements like routing and quality of service. The task patterns are used for decomposing tasks and delegating agents to them. A given task can be performed by one or multiple agents working in parallel. The interaction patterns are concerned with finding agents and facilitating their interactions.

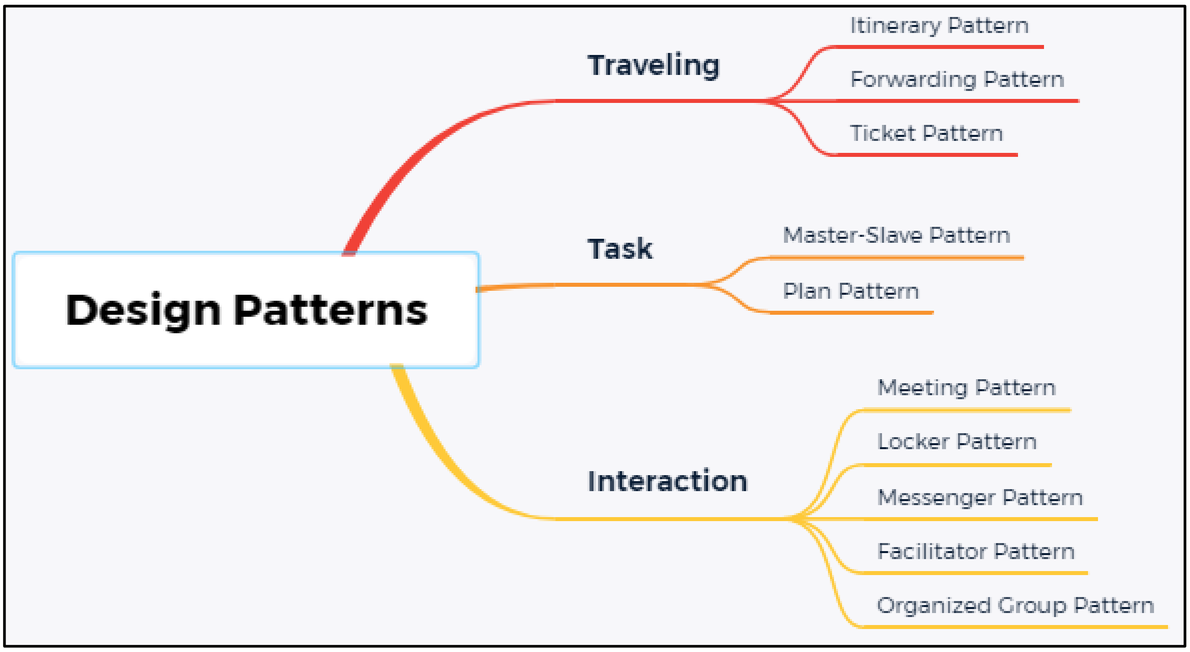

Figure 1.12 Design patterns for mobile agent applications by (Aridor and Lange, 1998)

Based on the design patterns presented in (Hayden et al., 1999) and (Kolp et al., 2001), the authors of (Tung et al., 2003) have also proposed a classification for design patterns of multi-agent systems. Their study focused only on the social patterns which will be presented next.

4.1 Social Patterns for Multi-Agent Systems Engineering

These are designs patterns that offer a social perspective on multi-agent systems (Do et al., 2005). The authors of (Tung et al., 2003) have presented patterns on the recurrent social and intentional aspects in multi-agent systems. They were classified into two categories peer and mediation patterns (**Figure 1.13**). The latter have intermediary agents (monitor, broker, mediator, embassy, matchmaker and wrapper) that help the other agents to reach an agreement on an exchange of services. The peer patterns (booking, call for proposal, and bidding patterns) describe the direct interactions between negotiating agents.

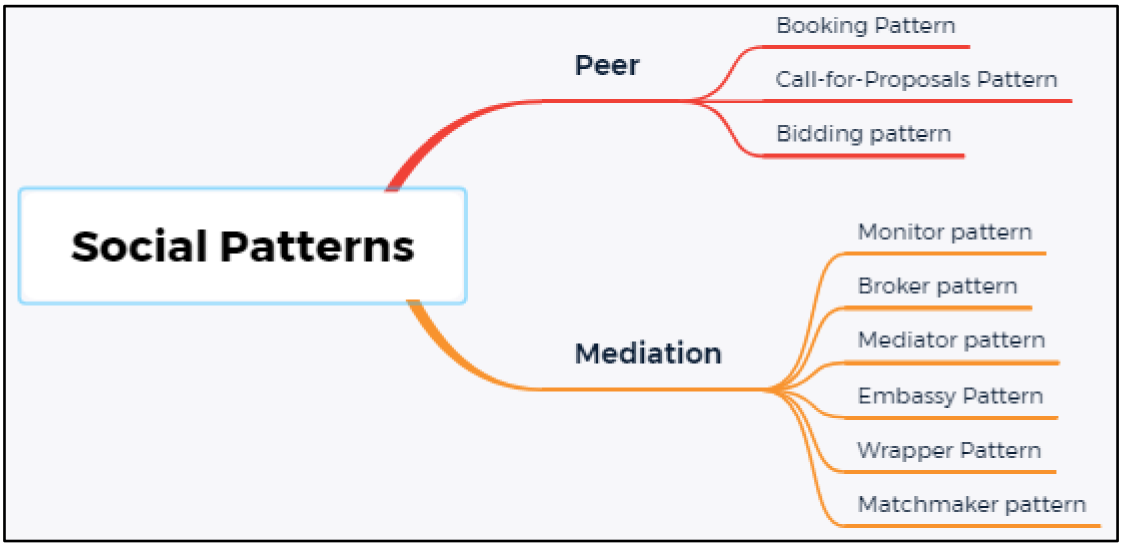

Figure 1.13 Social patterns for multi-agent systems by (Tung et al., 2003)

4.1.1 ***Peer Patterns***

4.1.1.1 Booking Pattern

The booking pattern is concerned with service or resource reservations. It involves a client and a number of service providers. The client requests resource reservation from a service provider. The provider can accept, deny, or suggest for the client to wait until the requested resource is available (Do et al., 2005).

4.1.1.2 Call-for-Proposals (Contract-Net) Pattern

It involves an initiator and a number of participants. The initiator calls for proposals for a service to all participants, and accepts the proposals that offer the service for a specified cost (Tung et al., 2003).

4.1.1.3 Bidding Pattern

It involves an initiator and a number of participants. The initiator organizes and manages the call for bids process, and receives proposals. At each interaction, the initiator publishes the current offer. Afterwards, the initiator can accept an offer, increases the offer, or cancel the process (Tung et al., 2003).

4.1.2 ***Mediation Patterns***

4.1.2.1 Monitor Pattern

In this model, subscribers register with a monitoring agent to receive notifications of change of state in certain subjects of their interest. The agent accepts subscriptions requests for notifications of topics of interest. It monitors the events and alerts subscribers to relevant ones (Do et al., 2005).

4.1.2.2 Broker Pattern

The broker agent is the intermediary in accessing the services of an agent (supplier) to meet consumer demand (Do et al., 2005). Broker agents locate suppliers corresponding to a consumer's service request. It requests and obtains the service from providers and then forwards it to the consumer.

4.1.2.3 Mediator Pattern

In this model, a mediating agent intervenes in the interactions between the different agents. An initiator addresses the mediator instead of asking another colleague directly (performer), the Mediator has models of knowledge of colleagues and coordinates cooperation between them. Conversely, each performer has a model of knowledge of the mediator. While the broker/matchmaker simply intermediates suppliers with consumers, a mediator encapsulates interactions and maintains patterns of initiators' behaviors over time (Kolp et al., 2001).

4.1.2.4 Embassy Pattern

An embassy routes a service requested by a foreign agent to a local agent then handles back the response. If access to the local agent is granted, the foreign agent can send messages to the embassy for translation. The content is translated according to a standard ontology. Translated messages are sent to local agents. The results of the query are forwarded to the foreign agent, translated in reverse (Kolp et al., 2002).

4.1.2.5 Wrapper Pattern

It integrates an existing system into a multi-agent system. This agent interfaces clients with the existing system by acting as a translator between them. In consequence, the existing system remains decoupled from the rest of the multi-agent system and the communication protocols are still respected (Kolp et al., 2010).

4.1.2.6 Matchmaker Pattern

A matchmaker agent locates a supplier corresponding to a consumer demand for a given service. The supplier address is transmitted to the consumer which interacts directly with the chosen supplier (Kolp et al., 2010). **Figure 1.14** shows a scenario for the Matchmaker model represented by a UML sequence diagram. The service provider sends a subscription to the Matchmaker which can accept or refuse the subscription request. When the customer sends a service request containing the characteristics of the service he wishes to obtain from a service provider, the matchmaker can respond with a refusal (refuse the request) or replies with the list of the addresses corresponding to the service providers. The service provider sends an unsubscription to the Matchmaker, where it can also respond with a refusal (refusal to terminate), or an acceptance (accept unsubscription).

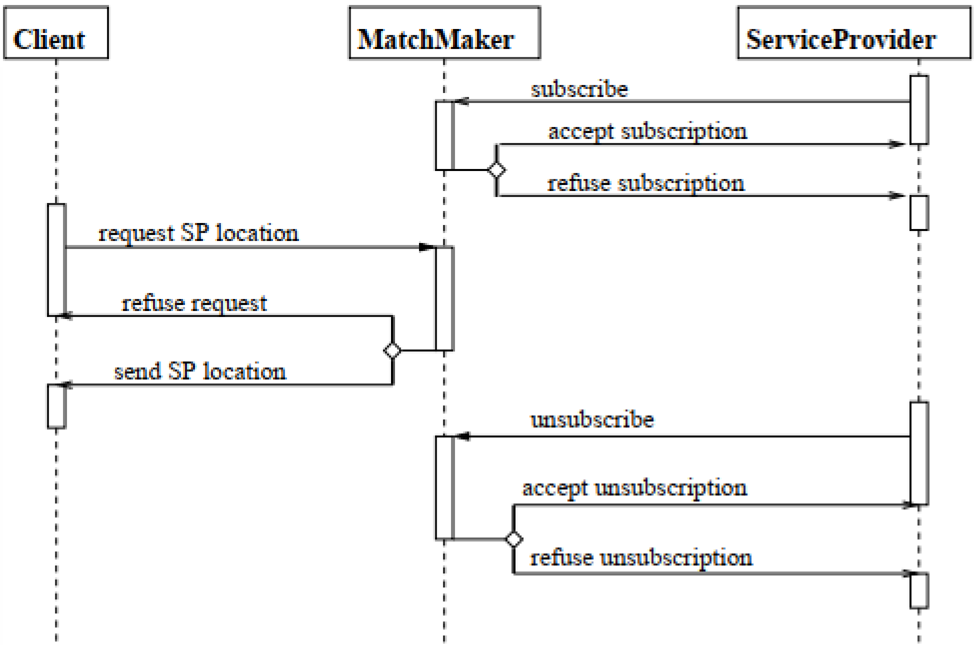

Figure 1.14 Communicational diagram for the Matchmaker pattern (Do et al., 2005)

5 Large-Scale Multi-Agent Systems

Nowadays, new technologies have emerged containing massive entities due to the rapid expands of the internet. A successful software can reach tens of thousands even millions of users all over the world (Đurić, 2017). Mobile devices are more powerful than before which can handle more complex tasks over faster networks like 4G and 5G. Not only smartphones and laptops are outdoors connecting devices. Others are also on the rise such as smart cars, smart bicycles, smart homes, smart cities, etc. These connecting entities shape the domain of *Internet of Things* (IoT) which is concerned with interacting things and objects. Even more, the *Internet of Everything* (IoE) which covers a wider range and a larger scale domain of interconnecting people, data, processes, things and objects (Schatten et al., 2016). These concepts were nicely presented by Cisco (Mihaylov et al., 2016): *"In terms of phases or eras,* *Cisco believes that many organizations are currently experiencing the Internet of Things (IoT), the networked connection of physical objects. As things add capabilities like context awareness, increased processing power, and energy independence, and as more people and new types of information are connected, IoT becomes an Internet of Everything — a network of networks where billions or even trillions of connections create unprecedented opportunities as well as new risks."*

Multi-agent systems can model such domains via its distributed nature of intelligent, autonomous, mobile and social agents. As MAS grows in its connecting agents and hosts, it becomes under the name of Large-Scale Multi-Agent Systems (LSMAS). The fact that IoT and IoE sound futuristic as LSMASs, it is only matter of time for this to emerge given to development rate in electronics (Schatten et al., 2016). However, LSMASs can be very adequate to model and implement many complex large-scale application domains.

The authors in (Schatten et al., 2017) have reviewed a number of applications domains for LSMASs (**Figure 1.15**). ***Massively multiplayer online gaming*** (MMOG) domain has gained a tremendous popularity recently. A MMOG consists on large numbers of players simultaneously playing online games. ***Massively multiplayer online role-playing games*** (MMORPG) (Schatten et al., 2015) are applications in which players can take control of their avatar using gaming desktops, laptops, smartphones and consoles. Players can compete, cooperate and form teams via matching mechanisms according to their language, country and tactics preferences. MMOGRPG also have virtual players (bots) controlled by autonomous and intelligent agents running on game servers. In **Table 1.4**, a list of the top 10 most played MMORPGs in one day is presented, knowing that these games are not for mobile gaming devices. **Table 1.4** shows that very large numbers of players were connected the moment these statistics were taken with hundreds of thousands online players. The peak values are the maximum number of players connected in the past 24 hours of that time.

Table 1.4 Top 10 most played MMORPGs in the last 24 hours of august 9^{th}, 2020 at 11:02AM according to dak.gg[1]

Game	Current Players	Peak Today
Counter-Strike: Global Offensive	453 616	843 194
Dota 2	355 697	640 933
PLAYERUNKNOWN'S BATTLEGROUNDS (PUBG)	126 924	411 710
Fall Guys	91 189	124 772
Destiny 2	68 193	73 705
Team Fortress 2	66 926	68 765
Grand Theft Auto V	62 349	116 862
Tom Clancy's Rainbow Six Siege	60 301	98 732
Rust	57 856	78 873
Source SDK Base 2013 Multiplayer	52 049	85 403

Simulations, virtual and augmented realities are other applications domains for LSMASs. Virtual reality is computer generated environment with graphics and animations. While the augmented reality uses real world environments augmented with virtual objects (Schatten et al., 2017). They both can be used for simulations, education, entertainment, etc. Certain simulations require large number of agents to model real life situations like emergency crowd evacuation management in public areas (Shendarkar et al., 2008)(Uno and Kashiyama, 2008).

Cooperative intelligent transport systems are also application domains of LSMASs. Complex traffic situations can be managed via interacting agents' organizations formed by traffic sensors, traffic services, ad-hoc vehicular networks, roadside infrastructures and autonomous and smart vehicles (Schatten et al., 2017).

Fourth Industrial Revolution (Industry 4.0) which concentrates on creating smart products and manufacturing processes based on industrial robotics, big data analysis and cloud services. LSMASs can be applied to this application domain since robots are physical incarnation of agents which are grouped with services and people to form many agents' organizations. These organizations cooperate by exchanging products, services and data to create the final product (Schatten et al., 2017).

Smart grids are enhanced electric grids which deliver electricity to consumers in electronically controlled world. The goal is to preserve efficient, sustainable, secure and

[1] Official PUBG game statistics and ranking website, https://dak.gg/statistics/playing

economic electricity supplies delivery to consumers who constitute the majority part of the grid. During peak times, smart grids manage to reduce power consumption from the consumers using sensors and specialized processors to manage power demands (Schatten et al., 2017).

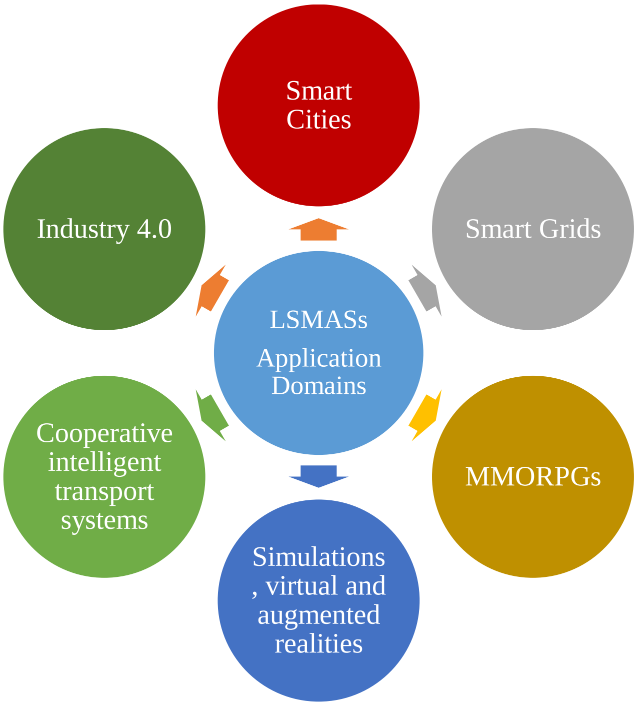

Figure 1.15 Large-scale multi-agent systems application domains by (Schatten et al., 2017)

6 Multi-Agent Development Platforms

In order to facilitate the development of multi-agent applications, several platforms have been proposed in the literature. Agent development platform is a software infrastructure which offers set of routines, protocols, and tools for designing, implementing, testing and deploying multi-agent systems. The list of proposed agent platforms is very long where many of them were closed. From the most used and still active agent platforms, we cite: JADE (Bellifemine et al., 2000), MadKit (Gutknecht and Ferber, 2000), Jason (Bordini and Hübner, 2005), Jadex (Braubach et al., 2004), Janus (Galland et al., 2010).

6.1 JADE Agent Platform

JADE (Java Agent DEvelopment Framework) (Bellifemine et al., 2000) is a java-based software framework. Using its library, JADE aids developers to create inter-operable intelligent systems according to FIPA specifications. JADE can also be seen as a runtime system for FIPA-compliant multi-agent systems to support application agents to exploit different features covered by FIPA specifications such as messaging and agent life management (Bellifemine et al., 2001). The JADE platform may also be distributed across multiple hosts interconnected via a network in which all the needed configurations can be managed remotely.

According to (Bellifemine et al., 2001), JADE offers many features for multi-agent systems development and deployment (**Figure 1.16**):

- ✓ FIPA compliant agent platform which contains: (1) a *Directory Facilitator* (DF) which provides a "Yellow Pages" service to the platform, (2) an *Agent Communication Channel* (ACC) which manages communication between agents, and (3) an *Agent Management System* (AMS) which oversees the registration of agents, their authentication, access and use of the system,
- ✓ Distributed agent platform on multiple hosts,
- ✓ Multi-domain environments can be built using additional DFs. A domain is a set of agents whose services are advertised via a common DF agent,
- ✓ For agent communication, ACL messages are offered as ordinary java objects to send and receive messages. The use of java objects to encode ACL messages rather than strings enhances intra-platform messaging experience,
- ✓ IIOP protocol to connect different agent platforms according to FIPA specifications,
- ✓ FIPA interaction protocols' library,
- ✓ Graphic user interface to monitor and manage agents.

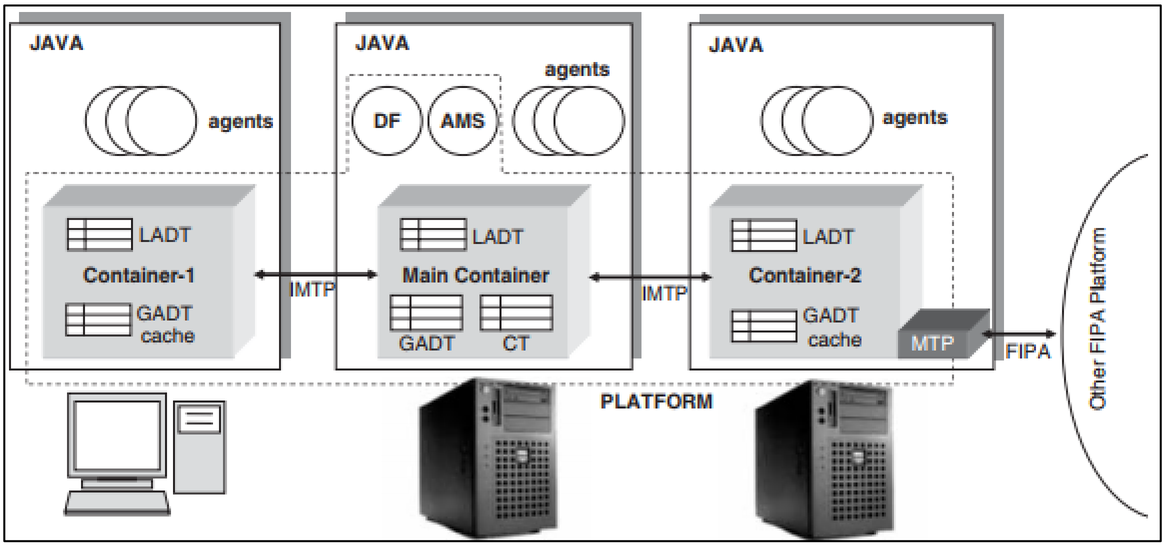

Figure 1.16 JADE's architecture (Bellifemine et al., 2000)

6.1.1 *The Directory Facilitator (Yellow Pages or Matchmaker)*

The Directory Facilitator (DF) is also known as the Matchmaker has already been presented previously. The DF is a centralized agent situated in the main container of the JADE platform which provides a Yellow Pages service that allows an agent to find the available services which they need to achieve their goals. In JADE's API, a module named *jade.domain.DFService* is used to implement requests for service registration, deregistration and search.

An agent wishing to publish one or more services must provide the DF with its agent description which includes its *agent identifier* (AID) and the list of its services descriptions (Snippet 1.1).

Snippet 1.1 JADE code pattern for service registration in the DF

```
DFAgentDescriptiondfd = new DFAgentDescription();
dfd.setName(genericAgent.getAID());
ServiceDescriptionsd = new ServiceDescription();
sd.setType(genericAgent.getServiceType());
sd.setName(genericAgent.getServiceName());
dfd.addServices(sd);
try {
DFService.register(genericAgent, dfd);
}
```

An agent wishing to search for services must provide the DF with a template description. A template is a service description object for the desired search. The result of the search is the list of all the descriptions which correspond to the supplied template. A service description corresponds to the template if all their attributes are matched like service name, type, etc. (**Figure 1.17**).

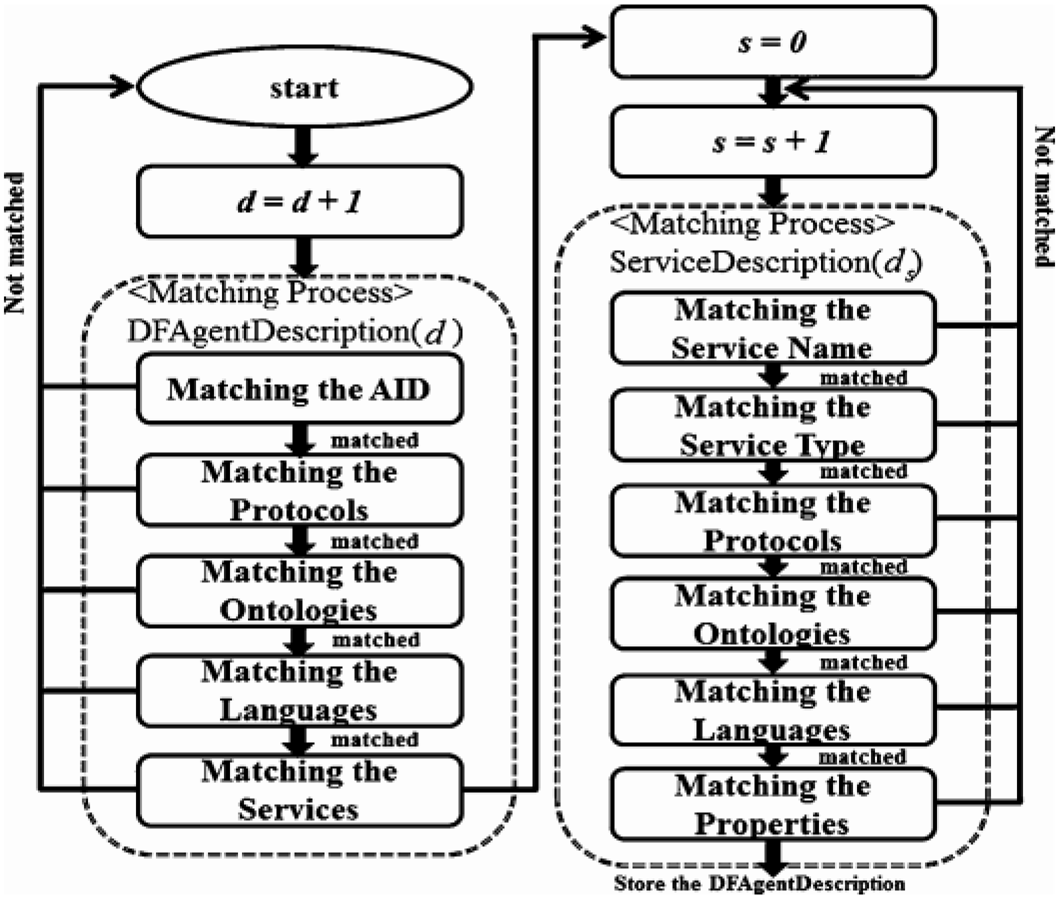

Figure 1.17 Service searching in JADE's DF (Han et al., 2012)

JADE's DF searches all the service descriptions instances one by one in a sequential manner. In consequence, this can cause an overhead and a slow search if there is a long list of registered services, and more waiting times if there are more search requests in the queue due to its centralized nature. The JADE code responsible for searching service is illustrated in Snippet 1.2, which is considered as a recurring pattern in any application that uses the JADE's DF. If an agent wishes to deregister its services from the DF it can simply call for the *DFService.deregister* function.

Snippet 1.2 JADE code pattern for service search request to the DF

```
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType(genericAgent.getServiceType());
template.addServices(sd);
try {
DFAgentDescription[] result = DFService.search(genericAgent,
template);
for (int i = 0; i <result.length; ++i) {
genericAgent.getProviders()[i] = result[i].getName();
}
```

6.1.2 *Implementing interaction protocols with JADE platform*

JADE has a library for implementing FIPA-compliant interaction protocols. For every protocol that JADE offers, a couple of classes are provided which are contained in *jade.proto* package. One of each couple classes is for agents playing the initiator role and another for the participant (or the responder) role. New interaction protocols can also be implemented using JADE library; however, preserving the interoperability nature of these protocols is essential to be able to interact with other FIPA-compliant systems (Bellifemine et al., 2001).

To implement a Contract-Net protocol interaction, JADE offers two behaviors *jade.proto.ContractNetInitiator* and *jade.proto.ContractNetResponder* (**Figure 1.18**). These two classes are instantiated later by the initiator and participant classes implemented by the developer. But before that, to handle the different messages of the conversation, the developer must override the existing functions such as *handlePropose*, *handleFailure*, etc. according to his needs. Once the *ContractNetInitiator* class is instantiated, the interaction is initiated and CFPs are sent to the desired participants.

A single participant can receive multiple CFPs from different initiators. In this case, using the *ContractNetResponder* will not be appropriate because it does not allow simultaneous interactions. The participant must finish the ongoing interaction to be able to read a new CFP. JADE offers a solution for this problem by instantiating the *SSIteratedContractNetResponder* class instead. Noting that this solution is not a pure parallelism since the participant does not perform two tasks at the same time even that it has multiple ongoing interactions.

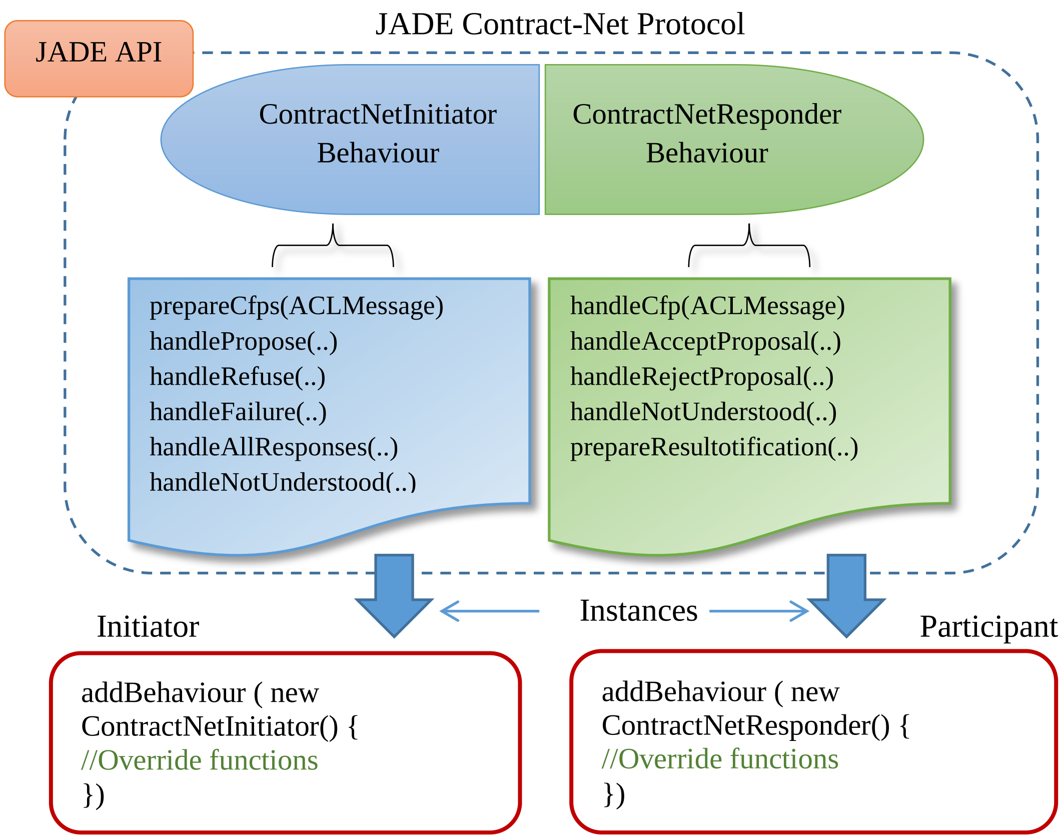

Figure 1.18 The JADE implementation for Contract-Net Protocol (Baldoni et al., 2013)

7 Conclusion

Agent oriented paradigm offers a great solution to solve problems that cannot be handled by monolithic systems. Developing multi-agent systems was the interest of many researchers in various domains. Multi-agent systems have long list of different concepts and utilities which their descriptions go beyond the content of one chapter. In consequence, we have limited ourselves in this chapter to present most of the concepts relating to our work.

In this chapter, we gave a brief overview of the concepts related to multi-agent systems such as agents, environment, interaction and organization. We presented too agent's sociability characteristic and its importance for multi-agent applications. Then, FIPA agent communication language and FIPA interaction protocols have been discussed especially Contract-Net protocol. A classification of social patterns for multi-agent systems engineering has been presented. Large-scale multi-agent systems were briefly discussed with some of their applications domains. Finally, the JADE platform has been presented for this thesis which we are accustomed to, and which is also considered to be one of the most used agent platforms (Bergenti et al., 2020).

Chapter 02

Software Maintenance

Chapter 2 Software Maintenance

1 Introduction

Software maintenance is considered to be the last phase in the software life cycle. Concepts related to software maintenance will be presented in this chapter like maintenance types and approaches. Reverse engineering, reengineering and refactoring are also briefly explained. We are going to focus on the perfective maintenance type, which includes concepts such as metrics, profiling, performance and scalability. Furthermore, the fact that this study is applied on multi-agent systems then only relevant metrics and tools for evaluating and profiling them are presented. Once the multi-agent system is evaluated and if any efficiency problems were detected then this system must be enhanced using agent or system level enhancement techniques.

2 Software Maintenance

Software maintenance is an integral part of the software development life cycle. It designates the modifications made to software after its delivery to correct faults, improve its efficiency or other characteristics, or even adapt it to a modified environment (ISO, 2006). In (ISO/IEC/IEEE, 2017) a software maintenance is

"(1) Totality of activities required to provide cost-effective support to a software system, (2) Entitlement of additional rights (such as additional functionality, upgrade or support) for a previously granted software entitlement."

The authors in (Jarzabek, 2007) have stated that companies spend more time on maintenance of existing software than on development of new ones. According to other various studies, software maintenance accounts for the following percentages of total software costs: 90% (Erlikh, 2000) (Moad, 1990), 75% (Eastwood, 1993), 60–70% (Huff, 1990) (Port, 1998), and 66% (Harrison, 1987) (Tan and Jarzabek, 1992). Earlier studies have also reported the following percentages: 50% (Lientz and Swanson, 1981), and 65–75% (McKee, 1984).

2.1 Why Software Maintenance Is Necessary?

A software product must pass through several important development phases before its delivery. When it is under deployment, new challenges always arise (Alkhatib, 1992) (Boehm, 1983) (Boehm and others, 1981) (Lientz and Swanson, 1980). The best-known challenge is fixing bugs. Even that a delivered product has been tested and validated is still susceptible to bugs, errors and failures. Correcting code errors is a high priority task for the software to run correctly. Another challenge is that new requirements and features can be requested by the user. Because of this challenge, the software maintenance is also called *software evolution* but only if there is a big change in the software. By removing, adding and enhancing features or adapting them to new technologies, the software life can be extended more from 15 to 20 years according to (Counet, 2007).

2.2 Inherent Difficulties in Software Maintenance

The main problem in performing software maintenance is that it cannot be done on a system that was not designed for maintenance (Schneidewind, 1987). When software is deployed it is no longer under support of the developers' team. The software maintainers are those responsible for such task. Thus, the maintainers must understand what and how the developers did their work. This requires good documentation, clean code structure and data models and dictionaries which were used for generating programmers' data. Unfortunately, according to (Martin and McClure, 1983) this is not always the case. Furthermore, users generally do not understand the software maintenance process requirements and they usually mix it with hardware maintenance. Users expect results in a timeframe similar to technical problems (Counet, 2007). The authors in (Schneidewind, 1987) have stated that:

"*Software maintenance is hard because:*

- *We cannot trace the product nor the process that created the product;*
- *Changes are not adequately documented;*
- *Lack of change stability;*
- *Ripple effect of making change;*
- *Myopic view that maintenance is strictly a post-delivery activity.*

In consequence, semi or fully automated tools are required for software maintenance process. They can facilitate software comprehension, change traceability/documentation and reduces maintenance cost and effort. The maintenance costs are very important for enterprises which are estimated to be 45% (Counet, 2007) of the total budget in America.

3 Types of Maintenance

Lientz and Swanson (Lientz and Swanson, 1980) have interviewed 487 data processing groups, where they identified four distinct types of maintenance as well as the proportion of time generally spent on each of them (**Figure 2.1**). Their survey showed that almost half of the maintenance effort was devoted to perfective maintenance tasks like enhancements for users, documentation and efficiency improvement. These tasks may be called discretionary maintenance activities because they may be considered to be optional. The other half was devoted to non-discretionary maintenance tasks which are mandatory activities like emergency fixes, debugging, and changes to input data and to hardware.

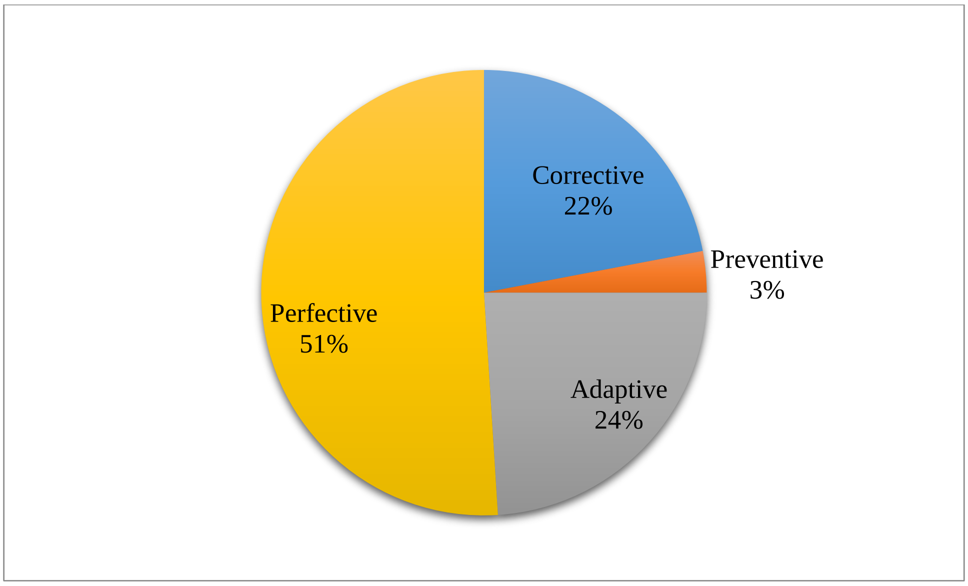

Figure 2.1 Proportion of total time spent on each type of maintenance (Lientz and Swanson, 1980)

Each maintenance type can be performed either reactively as a response to an event like errors and environment changes, or proactively as an initiative action to correct or enhance the software (**Table 2.1**).

Table 2.1 Types of software maintenance (Mens, 2017)

Why? / When?	Correction	Enhancement
Proactive	Preventive	Perfective
Reactive	Corrective	Adaptive

3.1 Corrective Maintenance

Corrective maintenance is the most burdensome part of software maintenance because it corrects design, coding, and implementation errors that should never have occurred (Burch and Grupe, n.d.). The system may not work as it should do according to its design plans. The objective of corrective maintenance is to locate the original specifications in order to determine what the system was originally intended to do (Counet, 2007). According to *ISO/IEC/IEEE 24765* (ISO/IEC/IEEE, 2017), a corrective maintenance is "*(1) The reactive modification of a software product performed after delivery to correct discovered problems, (2) Maintenance performed to correct faults in hardware or software.*"

3.2 Adaptive Maintenance

According to *ISO/IEC/IEEE 24765* (ISO/IEC/IEEE, 2017), an adaptive maintenance is "*Modification of a software product, performed after delivery, to keep a software product usable in a changed or changing environment.*" Adaptive maintenance fixes the various changes that must be made in order to follow the environment of the system in perpetual evolution. For example, the operating system can be updated, and modifications may be necessary in order to accommodate this new operating system. Unfortunately, the user does not see the direct changes in his application despite the efforts of the maintainer (Counet, 2007).

3.3 Perfective Maintenance

Perfective maintenance is performed to improve the performance, maintainability or other attributes of a computer program. It includes all changes made to a system to meet user needs. According to *ISO/IEC/IEEE 24765* (ISO/IEC/IEEE, 2017), a perfective maintenance is *"(1) Modification of a software product after delivery to detect and correct latent faults in the* *software product before they are manifested as failures, (2) Software maintenance performed to improve the performance, maintainability, or other attributes of a computer program, and (3) Improvements in software's performance or functionality, for example, in response to user suggestions and requests."*

Perfective maintenance occurs after the system has been in place and running fine for a while, then, end users start asking for minor tweaks improvements that could improve the way the system works (Rine, 1993).

3.4 Preventive Maintenance

Preventive maintenance proactively changes the software to avoid future problems. It can be defined as maintenance performed to prevent problems before they arise. According to *ISO/IEC/IEEE 24765* (ISO/IEC/IEEE, 2017), a preventive maintenance is "*(1) Modification of a software product after delivery to detect and correct latent faults in the software product before they become operational fault, (2) Maintenance performed for the purpose of preventing problems before they occur, (3) Designing a software system that is easy to maintain, and (4) Continuously upgrading a system to enable it to cope with current and future changes"*. This type of maintenance can be considered very important in the case of critical systems such as those related to aeronautics, for example.

4 Software Maintenance Process Models

Many software maintenance models have been proposed to help maintainers in their tasks. These models divide the maintenance process into smaller tasks for better management.

4.1 The Quick-Fix Model

It is a specified (ad hoc) approach for software maintenance. We wait for the problem to occur and then we try to fix it as quickly as possible (**Figure 2.2**).

In this model, the corrective measures would be carried out without detailed analysis of their effects on the code structure or other parts (ripple effects). Changes trace and documentation may or may not be updated. If the software is developed and maintained by the same person then this model can be very appropriate, because this developer is very familiar with his code and documentation may not be needed. However, in a larger environment where many developers and maintainers are involved, this model could be helpful in a short term because users need their software corrected rapidly. But, in a long-term, more difficulties will arise due to missing documentation, change traces, etc. (Grubb and Takang, 2003).

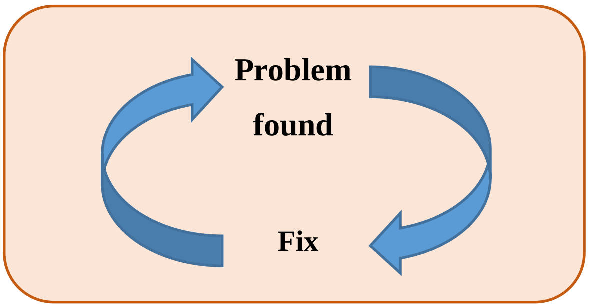

Figure 2.2 The Quick Fix model

4.2 Boehm's Model

In (Boehm, 1983), the author has proposed a model for the maintenance process based on economic models and principles. Boehm's idea was that economic models and principles could not only improve maintenance productivity, but also help understand the process. He represented the maintenance process as a closed loop cycle (**Figure 2.3**), and theorized that it is the step of management decisions that drives the process. In this step, a set of approved changes is determined by applying specific strategies and cost-benefit assessments to a set of proposed changes. The approved changes are accompanied by their own budgets which will be largely determined by the extent and type of spent resources (Grubb and Takang, 2003).

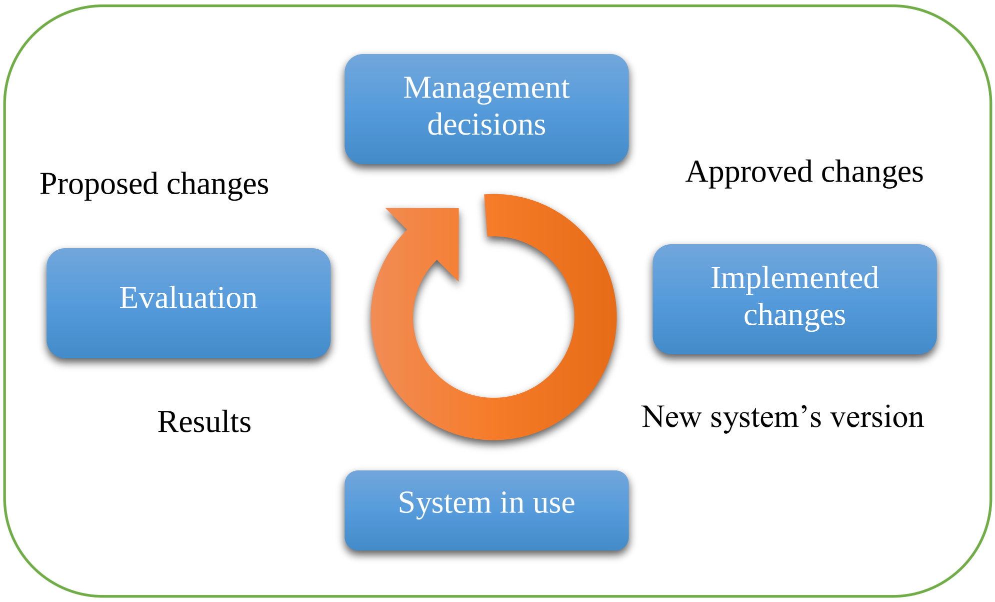

Figure 2.3 Boehm's Model (Boehm, 1983)

4.3 Osborne's Model

The Osborne model (Osborne, 1987) (**Figure 2.4**) is concerned with the reality of the maintenance environment. According to the author, poor control and communication in management are the reasons for the technical problems which can arise during maintenance (Erdil et al., 2003). Therefore, Osborne has recommended four strategies to answer these problems:

1) Change specification must include the maintenance requirements.
2) To establish quality assurance requirements, a quality assurance program is necessary.
3) Metrics need to be developed to verify that the maintenance objectives have been met.
4) Managers should be provided with feedback through performance reviews.

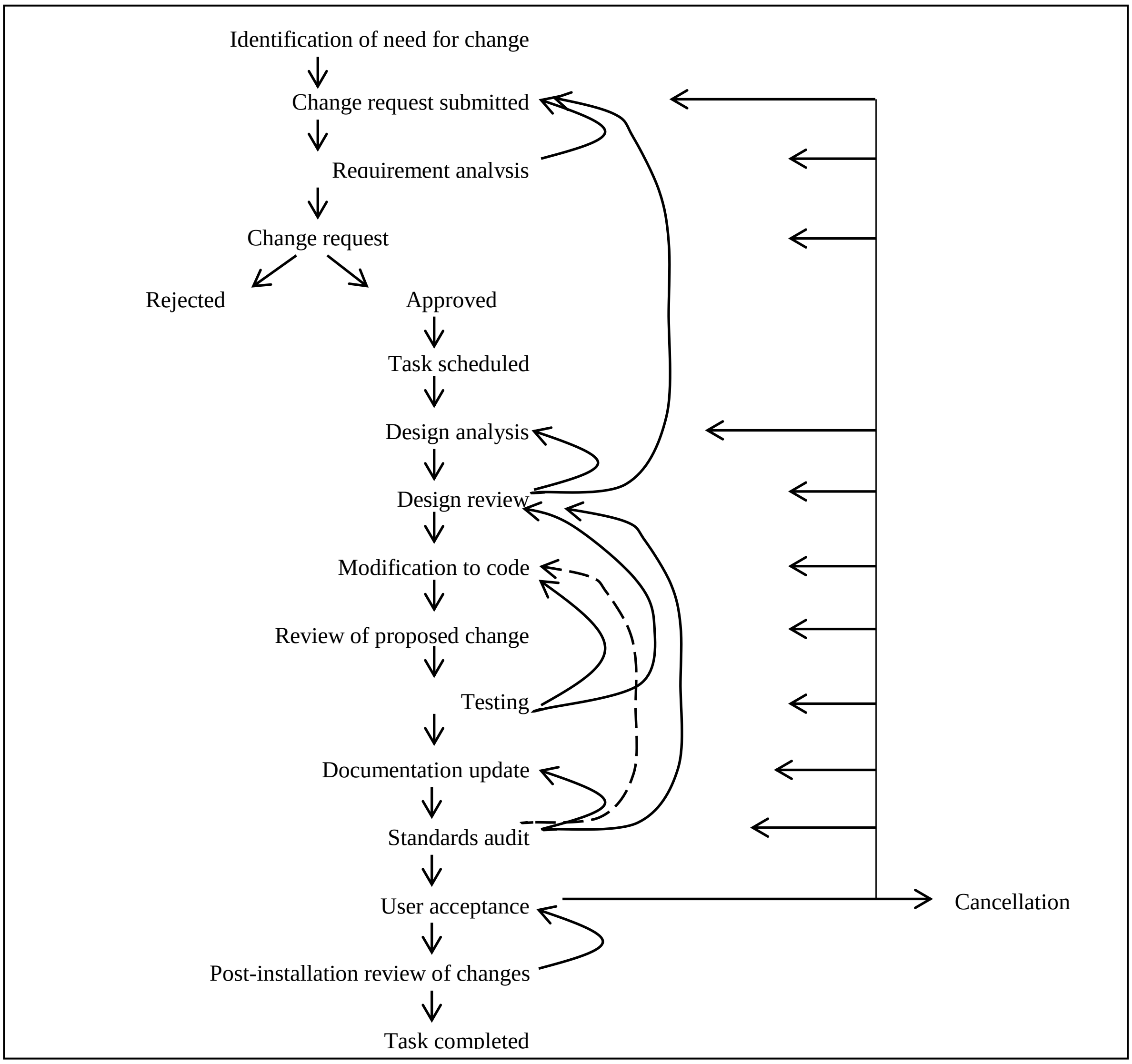

Figure 2.4 Osborne's model (Osborne, 1987)

4.4 The Iterative Enhancement Model

The iterative enhancement model considers changes to the system during software's life to be an iterative process (**Figure 2.5**). Originally, this is a software development model that can be applied to software maintenance. It evaluates an existing system for redesign and modification. The design also evolves with the product which will never be final and complete. The underside of this model is that it can require time to be performed which may not suit the customer needs (Genuchten, 1992). A mandatory condition is that this model requires a complete system's documentation in the beginning to be effective. It has three stages: (1) analyzing the system, (2) characterizing the proposed changes, and finally (3) implementing the changes.

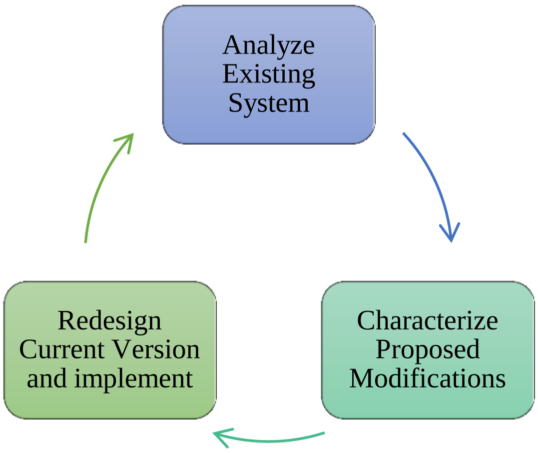

Figure 2.5 The iterative enhancement model

4.5 The Full Reuse Model

In order to reduce the maintenance's cost and effort, the full reuse model (Basili, 1990), as its name indicates, proposes the reuse of an old system's components into the new one (**Figure 2.6**). Old components are already tested and ensured to have high quality. Component-based software development is an adequate example of the reuse model (Ning, 1996). The full reuse model starts by identifying the reusable parts of the old system, have a good understanding of the system's parts then modify the old parts to new requirements, and integrate the recently modified parts in the new system. Similarly, to the iterative enhancement model, the full reuse model also uses the old system. But, while the iterative enhancement model only enhances the last version of the current system, the full reuse model frees us to design the new system's solution from other similar systems' solutions.

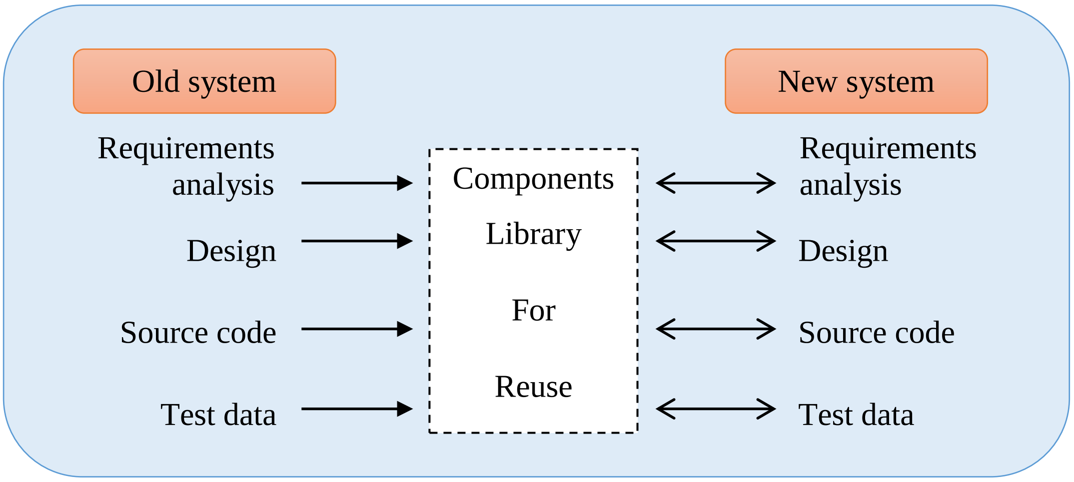

Figure 2.6 The Reuse Model (Basili, 1990)

4.6 Taute Maintenance Model

This model was developed by B.J. Taute in 1983 and elaborated by (Parikh, 1986). This model was presented as a cycle of eight consecutive phases (**Figure 2.7**). It is considered to be perfect for a corrective maintenance type but unfortunately it does not support perfective or adaptive maintenance. The whole process starts by a change request from the client. After registering the request by the maintenance team, they estimate the time and effort required for realizing the requested changes. In the schedule phase, the team prepares the planning's documents before they start the programming phase in which the modification takes place. In this phase, the source code will be modified as well as the design and the documentation. The implemented features will be tested in the next phase which needs to be treated carefully where new test cases may be required along with the existing ones. System and user's documentation must be prepared and delivered to the user who performs an acceptance test. If the user accepts the modifications, then the software can be used. If any new requirement arises during the operation phase, then the user requests a new software change which repeats the whole process.

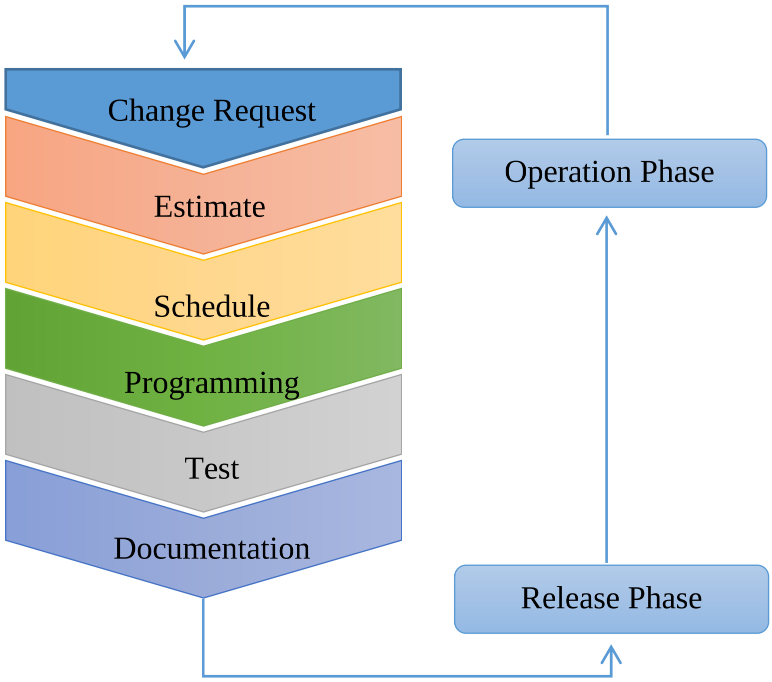

Figure 2.7 Taute maintenance model by B.J.Taute in 1983 (Parikh, 1986)

4.7 IEEE Model

The IEEE 1219 standard for software maintenance has proposed the maintenance process model presented in **Figure 2.8.** The process starts after software's delivery. When a modification request arrives, it will be classified into the appropriate maintenance type then identified with a unique number. In the analysis phase, a preliminary plan for design, implementation, test and delivery is devised based on the collected information of the previous phase along with the project's documentation. In the design phase, it is required to identify the affected module, modify software module documentation like data and control flow diagrams, schematics, etc., and create test cases for the design, select metrics and regression tests for the implemented features. After implementing and testing the modified software, the acceptance test is carried by the customer followed by the delivery activities like backup, installation and training at the costumer facility (Yongchang et al., 2011).

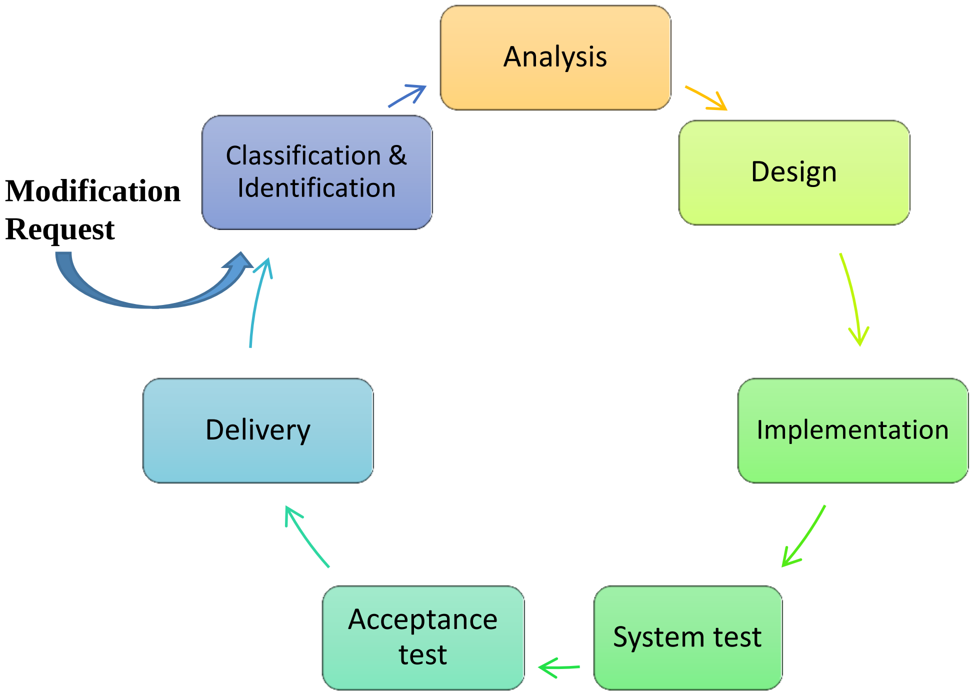

Figure 2.8 IEEE Maintenance process activities - IEEE 1219 (Yongchang et al., 2011)

4.8 ISO/IEC Model

International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) have proposed software maintenance activities (ISO/IEC 14764) similar to that of the IEEE 1219 but organized little differently (Bourque et al., 2001). Each activity is divided into six tasks. The starting point of this model is the process implementation which is a pre-delivery task that creates plans for post-delivery maintenance activities. These latter are performed in a continuous loop, beginning with the problem and modification analysis, then software modification, and last with maintenance reviews and acceptance. Migration activity is when the systems need to be executed on different environments. The last activity is the retirement in which the software is no more useful (**Figure 2.9**).

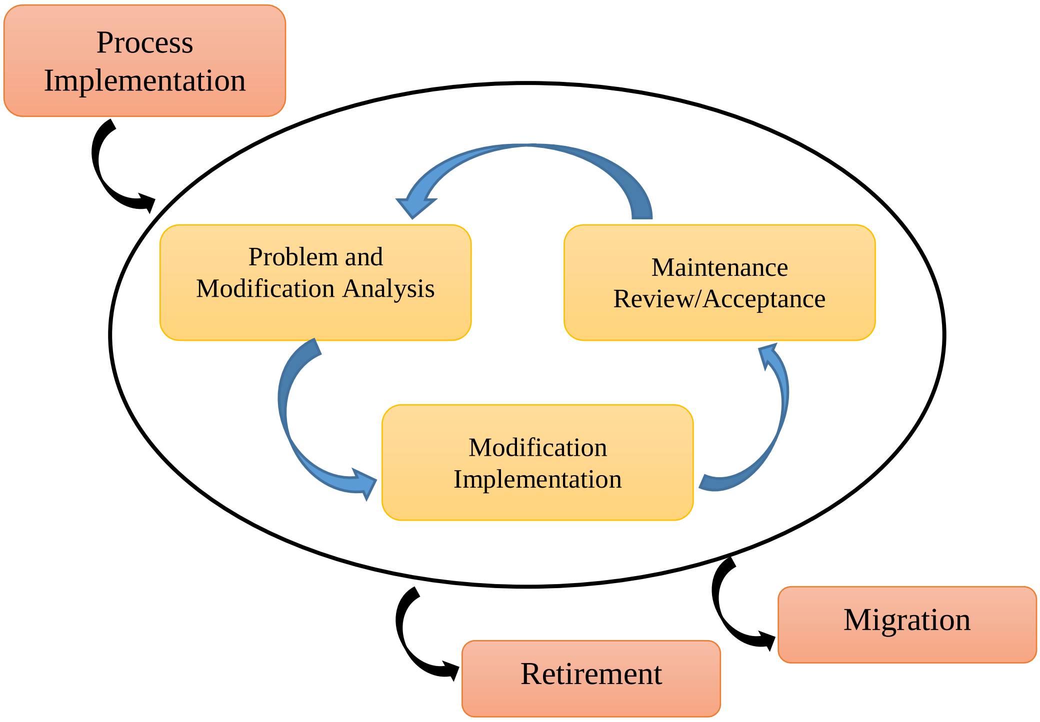

Figure 2.9 ISO/IEC 14764 Maintenance Process Activities (Bourque et al., 2001)

5 Reverse Engineering, Reengineering and Refactoring

Reverse engineering is an important concept within the research domain of software maintenance and evolution (Chikofsky and Cross, 1990). When software documentation is missing or outdated, the reserve engineering activity helps maintainers to understand the system's architecture and behavior by building higher-level, more abstract, software models using only the source code (Mens and Demeyer, 2008). Reverse engineering is a passive activity and it does not alter the software. However, it is a crucial step before starting any reengineering effort (Varga, 2017).

Reverse engineering can also be regarded as the initial phase in the process of software reengineering (Arnold, 1993). Reengineering is necessary when we are confronted with legacy systems. These are systems that are still valuable, but they are notoriously difficult to maintain (Demeyer et al., 2002). The goal of reengineering is thus to come to a new software system that is more evolvable and possibly has more functionality than the original software system.

The reengineering process is typically composed of three activities (**Figure 2.10**). First, reverse engineering may be necessary when the technological platform of the software system (language, tools, machines, operating systems) is outdated, or when the original developers are no longer available. This activity is typically followed by a phase of software restructuring or refactoring (Arnold, 1989) in which we try to improve crucial aspects of the system. Finally, in the forward engineering phase we build a new running system based on the new restructured model (Mens and Demeyer, 2008). In relation to software maintenance, the perfective maintenance is the closest type to be as a reengineering process.

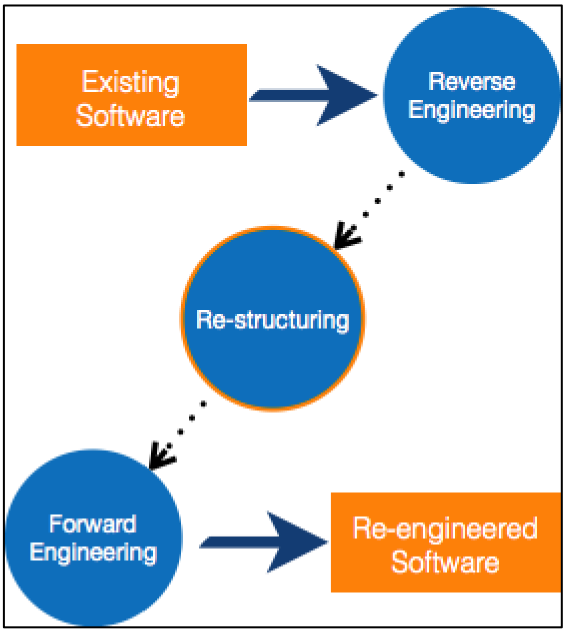

Figure 2.10 A software reengineering process ("Software Engineering," n.d.)

Another reengineering process model was proposed in (Rather and Bhatnagar, 2016) and contains other activities for software reengineering like inventory analysis, document and data restructuring (**Figure 2.11**).

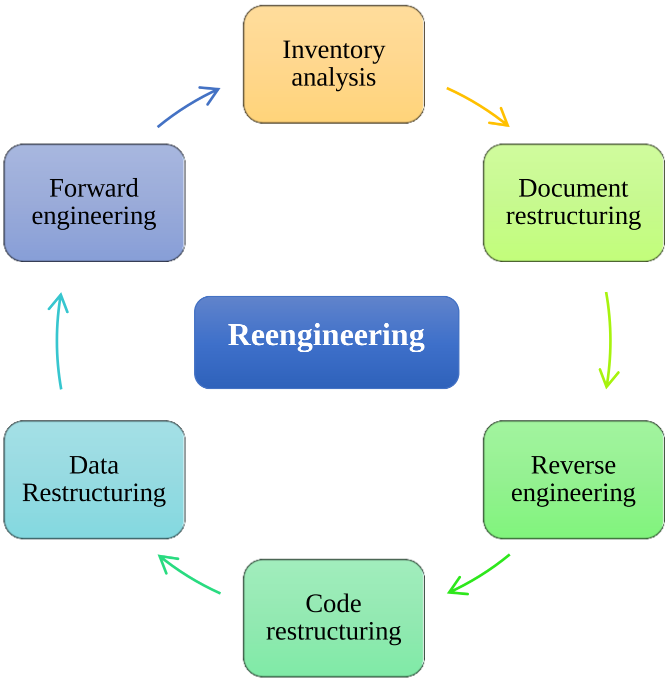

Figure 2.11 A software reengineering process by (Rather and Bhatnagar, 2016)

6 Performance and Scalability in Software Quality

Software quality is a major concern in software development. Testing, evaluating, validating and maintaining software tasks are all performed to enhance software quality. A software quality is represented by the software's characteristics, sub-characteristics and metrics which form what is called as software quality model. ISO/IEC 9126 standard has proposed six quality characteristics: functionality, reliability, usability, maintainability, portability and efficiency in which software performance and scalability are placed. ISO/IEC 25010 is an update of the previous standard which adds two more quality characteristics: compatibility and security.

6.1 Performance

Software performance becomes a common practice in the software engineer's activities. It is a non-functional concern which must be looked in software engineering. In (ISO/IEC/IEEE, 2017), a performance is *"(1) Degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage, (2) Extent to which the execution of an application in the production environment achieves its purpose in terms of speed of input, transfer, processing, storage and output (the response speed of an application observed by an end user)."* Performance evaluations could be carried statically in the early phases of software development or dynamically after code implementation. Software performance can be measured using different indicators like throughput, execution time, response time, waiting time and number of executions, etc. (Cortellessa et al., 2011). In multi-agent systems, more indicators such as the number of agents / concurrent tasks and agent states, etc. can be used for performance evaluation (Lee et al., 1998).

6.2 Scalability

Scalability has become an important concern in distributed systems. Scalability is the ability of a system to maintain its functionality and performance in an overloaded system or in a wide network. According to (Neuman, 1994), system’s scalability has three dimensions: numerical, geographical and administrative scalability. The numerical (size) scalability is related to the number of objects, services, agents or users of the system. The geographical scalability is about the distance over which the system is dispersed. The administrative dimension is about the number of organizations which manages and controls parts of the system. In our thesis, large scale systems are those with high number of users, agents, tasks, communications, data, etc. They often need more computing resources and storage capacity. Upgrading the hardware to enhance the performance and the scalability is very costly and may not offer good results if the problem is mostly related to bad software design.

A fast way to improve the performance and scalability of an existing system is to start by its evaluation. This evaluation is generally started by a dynamic analysis then a static one if needed. The dynamic analysis process consists on overloading the system to put it under stress in various conditions. Using performance metrics such as CPU usage time, methods invocation counts, etc., the analyzer can detect anomalies and congestion points while the system is running. The dynamic analysis results are then investigated along with system’s source code and the documentation to locate the cause of the problem. This is obviously a reverse engineering task, which be followed by new reengineered design to be implemented during the refactoring process.

The goal of multi-agent systems’ perfective maintenance is to enhance agent or system level properties like reactivity, autonomy, interaction, etc. Therefore, we must think differently than we do with a normal system. Object-oriented programming is known to be used for multi-agent systems development like with JADE platform, which means that multi-agent system’s problems can be raised due to object-oriented programming issues as well like duplicated code (Baker, 1993), high fan-in (Marin et al., 2004), etc. These latter must be evaluated and eliminated first before assessing multi-agent systems properties.

7 Performance and Scalability Evaluation in Multi-Agent Systems

Every system has to be evaluated to better understand its behavior. In our opinion, asking when, what and how to evaluate a system are the most important questions related to an evaluation task. An evaluation can be performed before systems' deployment for test purposes or after systems' deployment for a maintenance purpose. In both cases, this evaluation results in information which represents what we need to evaluate. To be more precise on what we need to know about a system, we use the concept of metrics, which is simply a measure of a software characteristic. Then, to obtain these measurements two tasks are available for us, profiling and monitoring.

7.1 Performance Metrics

According to ISO/IEC 9126 standard, a measure is the number or category assigned to an attribute of entities by making a measurement. A measurement is the use of a metric to assign a value from a scale to an attribute of an entity. A metric is the defined measurement method and the measurement scale. There are three kinds of metrics: internal (static), external (dynamic) and quality in use metrics. Internal metrics (ISO/IEC 9126-3) measure internal software attributes or indicate external software attributes by analyzing non-executable product elements like specifications or source codes during design and coding phases. Internal measures are values that can be extracted from code statements, control graphs and data flow representations, etc. External metrics (ISO/IEC 9126-2) are measured during software execution to analyze its behavior. In our study, the external metrics are those which we are interested in. Quality in use metrics (ISO/IEC 9126-4) depends on the use of the system. It measures the extent to which a product meets the needs of the user. It represents the user's view of the quality of the software in a given context of use.

The ISO/IEC 25023 standard (ISO/IEC, 2016) which is a revision for the ISO/IEC 9126 standard, has presented three sub-characteristics for performance efficiency of software quality which are *Time behavior*, *resource utilization* and *capacity*. Each sub-characteristic has several metrics presented in **Table 2.2.**

Table 2.2 Performance metrics according to ISO/IEC 25023:2016 (ISO/IEC, 2016)

Performance efficiency measures	Metrics ID	Metric name	Description
Time behaviour measures	PTb-1-G	Mean response time	How long is the mean time taken by the system to respond to a user task or system task?
	PTb-2-G	Response time adequacy	How well does the system response time meet the specified target?
	PTb-3-G	Mean turnaround time	What is the mean time taken for completion of a job or an asynchronous process?
	PTb-4-G	Turnaround time adequacy	How well does the turnaround time meet the specified targets?
	PTb-5-G	Mean throughput	What is the mean number of jobs completed per unit time?
Resource utilization measures	PRu-1-G	Mean processor utilization	How much processor time is used to execute a given set of tasks compared to the operation time?
	PRu-2-G	Mean memory utilization	How much of memory is used to execute a given set of tasks compared to the available memory?
	PRu-3-G	Mean I/O devices utilization	How much of I/O device busy time is used to perform a given set of tasks compared to the I/O operation time?
	PRu-4-S	Bandwidth utilization	What proportion of the available bandwidth is utilized to perform a given set of tasks?
Capacity measures	PCa-1-G	Transaction processing capacity	How many transactions can be processed per unit time?
	PCa-2-G	User access capacity	How many users can access the system simultaneously at a certain time?
	PCa-3-S	User access increase adequacy	How many users can be added successfully per unit time?

Several works have proposed metrics to measure the performance and scalability of multi-agent systems as in (Babczyński and Magott, 2006) (Lee et al., 1998) (Woodside, 2001), other works have aimed at the evaluation of communication where each with his own proposal as in (Babczyński et al., 2005) (Cortese et al., 2002) (Babczyński et al., 2004) (Doan Van Bien et al., 2010a), as well as others who have proposed to evaluate services like the directory service in (Hussain and Shabbir, 2008) and (Mengistu et al., 2008). The metrics in **Table 2.3** have been presented by (Nagwani, 2009) which were identified from general architectures and implementations of MASs (Kargupta and Stafford, 1997) (Hallenborg, 2004). A summary of some metrics used in the literature is also presented in **Table 2.4.**

Table 2.3 Performance metrics in MAS reported by (Nagwani, 2009)

Metric	Designation
Number of agents	The size of MAS is measured by the number of agents. More agents mean more size and complexity.
Computational time	It is the time spent by the agent in the active state.
Used memory	The memory space occupied by the agent and its objects.
Agents status	Number of agents' state changes can influence its performance.
Agent Coordination	Coordination is important for better timing, resource sharing and complex tasks partitioning.
Communication	Number of messages, message size and communication time are major metrics for agents' communications.
Agent Management	It traces the agents' states, their achievements and remaining tasks.
Dependencies between the agents	It is related to the data segment where agents are executing. An agent can be dependent on the output of another one. RAW (Read After Write), WAW (Write After Write) type of dependencies may exist in MAS.

Table 2.4 Few performance metrics for multi-agent systems proposed in the literature

Research	Metrics	Evaluated using
(Camacho and Aler, 2005)	Software building characteristics • Development time: is the number of hours to analyze and design, time needed to implement new role or characteristic, integration with external code which means time to modify a generic agent offered by the frameworks to build specialized agents, tests and debugging time. • Software reusability: LOC, new java classes, number of reused classes necessary to build new	JADE, JatLite, SkeletonAgent, Zeus.

	agents. Runtime Characteristics • Performance evaluation (Response or request time to finish a MAS task. Number of exchanged messages) • Scalability evaluation (Increase number of agents and calculate response time)	
(Lee et al., 1998)	• Computational time • Number of tasks • Number of agents	Zeus
(Cortese et al., 2002)	• Messaging RTT (Round Trip Time) • Number of agent couples (Sender/Receiver)	JADE
(Wilde et al., 1999)	• Communication load (Number of messages) in relation to the number of agents • Speed of agent's network in relation to the number of tasks	Zeus
(Dikaiakos and Samaras, 2000)	• Total completion time. • Average behaviour time. • Peak rate of performance capacity.	Concordia, Voyager, Aglets, Grasshopper.
(Woodside, 2001)	• Productivity. • Scalability ratio = New Productivity / Old Productivity.	N/A (Mobile agents)
(Turner and Jennings, 2001)	• Number of passed ticks (process/ communication cycles). • Number of agents. • Sum of the number of logical inferences required for each tick.	Prolog

(Nagi, 2001)	• Throughput (number of terminated actions per agent per second). • Response time of plan execution. • Ratio of compensated actions to the terminated ones. • Ratio of abandoned actions to the terminated ones. • Ratio of aborted actions to the terminated ones.	N/A (proposed middleware)
(Babczyński et al., 2005)	• Probability of receiving a response. • Mean time of receiving a response. • Mean number of bytes transmitted in network per second.	JADE
(Hussain and Shabbir, 2008)	• RTT (Round Trip Time)	JADE
(Helsinger et al., 2003)	Message transport metrics • Queue length • Message size • Message count Agent blackboard metrics (blackboard is the feature to provide the metric service in the Cougaar architecture) • Asset count • Plan element count • Task count • Object count • Custom object count	Cougaar
(Joumaa et al., 2008)	• Weight of pertinent messages (which have an impact on the agent's state) (Interaction evaluation)	N/A
(Cosio and Magariño, 2009)	• $BS(A_j)$: Ratio of the sent messages by an agent A_j on the number of all sent messages by all the agents. • $MS(A_j)$: Ratio of the received messages by an agent A_j on the total number of all the received messages by all the agents.	INGENIAS Development Kit

	• $BR(A_j)$: Ratio of the sent messages by an agent A_j on the number of all sent messages by all the agents playing the same role. • $MR(A_j)$: Ratio of the received messages by an agent A_j on the total number of all the received messages by all the agents playing the same role. • Response time.	
(Marir et al., 2016)	Autonomy metrics • *Ratio of the lack of requesting services (RLRSs)* • *Ratio of resources availability (RRA)* Reactivity Metrics • *Ratio of states changes (RSC)* • *Time to respond to changes (TRC)* Pro-activity Metrics • *Ratio of achieved purpose (RAP)* • *Time to achieve purpose (TAP)* Sociability metrics • *Ratio of interaction utility (RIU)* • *Ratio of interaction intention (RII)* • *Ratio of understand-ability (RU)* Rationality metrics • *Goal achievement acceleration (GAA)* • *Goal achievement by using resources (GAUR)* Specialization metrics • *Average number of roles played by agent (ARA)* Granularity metrics • *Behaviour granularity (BG)* • *Knowledge granularity (KG)* Organization metrics • *The number of sub-organizations* • *The average number of agents by organization* • *The relationships diversity* Environment metrics • *Accessibility* • *Determinateness*	JADE

- *Episodically*
- *Dynamicity*
- *Continuity*

Modularity metrics

- *Coupling*
- *Cohesion*

7.2 Performance Monitoring and Profiling of Multi-Agent Systems

Profiling is a performance analysis technique introduced by (Knuth, 1971). Its goal is to search for memory leaks, performance bottlenecks, and hot spots which are programmed fragments that take a long execution time (Doan Van Bien et al., 2010a). It is considered to be a software test task by running multiple execution scenarios with various generated test conditions (workload). Profiling introduces too much overhead to the analyzed system by halting its execution or its subroutines (or JVM) periodically to inspect call stacks and form call trees. During the software test phase, testers do not care about system's stability and overhead; instead they want more detailed information about the analyzed system. In the contrary, system stability and overhead are important concerns after software deployment. Profiling at this stage is inappropriate only in case of emergency system failures (**Figure 2.12**). Therefore, another analysis technique with less impact on the system's stability and overhead is needed. This technique is called monitoring which can run in parallel with the analyzed system without affecting its stability or overhead too much (Waller, 2014). Monitoring collects, processes, aggregates and displays real-time quantitative data about the system (Beyer et al., 2016), like error types and counts, execution times, waiting times, etc. Some differences between monitoring and profiling are presented in **Table 2.5**.

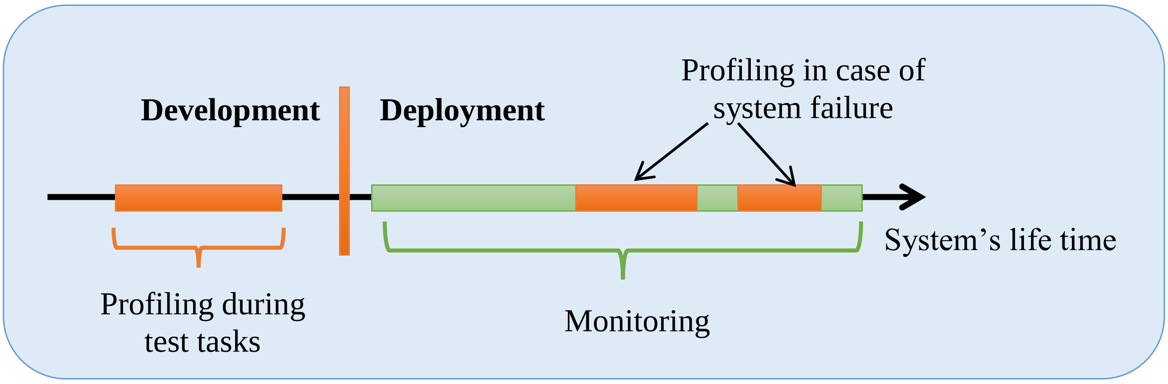

Figure 2.12 Monitoring and profiling during system's life

Table 2.5 Typical differences between profiling and monitoring (Waller, 2014)

	Profiling	**Monitoring**
Employed during	Development	Operation
Workload	Generated	Real
Data gathered	Detailed	Specific
Accepted overhead	High	Low

As explained previously, in order to perform a perfective maintenance to enhance the performance and scalability of a given system, the maintainer has to understand the details of the problem using profiling techniques. In (Nagwani, 2009), the authors have highlighted some important tasks involved in a profiling task which agrees with the performance metrics of ISO/IEC 25023 standards (ISO/IEC, 2016):

- **Memory profiling:** captures the memory space occupied by the agent or the system,
- **Processor profiling:** captures execution times which can be visualized also as CPU usage rate,
- **Remote profiling:** is useful for distributed systems where components are located in different hosts. In this case, profiling can be performed remotely,
- **Communication / Network Profiling:** is useful for network-based systems to capture communication delays.

7.2.1 *Multi-Agent Systems Profiling Techniques and Tools*

Profiling techniques and tools for MASs are very limited compared to those of other paradigms and programming languages like Gprof (Graham et al., 1982), JVMTI (Oracle, n.d.), ATOM (Srivastava and Eustace, 1994).

In (Doan Van Bien et al., 2010b), a tool called Agent spotter was proposed for the profiling of MASs. In this work, the authors presented the architecture and the space-time diagram of the tool, the latter aims to identify the events in a system that have an impact on performance. In another paper (Doan Van Bien et al., 2010a) related to the previous work, the authors presented the call graph of the tool which attempts to contextualize performance information by linking it to occasional events such as the passing of messages. Another tool proposed for profiling performance in MASs is Cougaar (Helsinger et al., 2004). It offers event detection, monitoring of ACL messages and other services. The LS / TS agent platform provides an administration tool that records certain high-level system monitoring information (Rimassa et al., 2005). The main limitation of these systems is the lack of post-processing of raw performance data in order to produce meaningful synthetic indicators like a profiler would do.

In addition to performance analysis and profiling, most agent development frameworks provide debugging tools similar to the Agent Factory Debugger (Collier, 2007). This tool provides information on mental state and communication from the point of view of individual agents. A different type of debugging tools is the AgentViewer, provided in the Brahms toolbox (Seah et al., 2005), which displays the agent's time tables so as to understand when the agents' actions are taken.

The Zeus platform (Nwana et al., 1998) contains the *Society tool* which visualizes the interaction between agents in order to help understanding the topology of social contacts within a MAS.

As part of JADE agent development, the *Sniffer* agent is a FIPA compatible agent, which monitors messages created with ACL and passed between agents. These messages are presented in a simple graphical interface (Bellifemine et al., 2000). A more sophisticated tool, called ACLAnalyser (Botía et al., 2004), provides more detailed information on agent communication (Doan Van Bien et al., 2010a).

8 General Techniques for Improving Multi-Agent Systems

Several works have been carried out to improve the performance and scalability of distributed systems by proposing new architectures and topologies (design phase) (Aguirre et al., 2000) (Othman and Schmidt, 2001) (Zhang et al., 2011) (Muscat, 2003), which requires a redesign and redevelopment of the code in the case of legacy systems. Distribution, replication and caching are the most used mechanism to obtain good scalability according to (Neuman, 1994). In this thesis, we are interested in perfective maintenance techniques for these legacy systems without going back to the preliminary phases of the development cycle. These works are very few and do not offer automatic or semi-automatic tools to maintain the system.

The following performance and scalability improvement techniques in multi-agent systems presented in (Koster, 2011) are organized into two categories: agent level and system level (**Figure 2.13**).

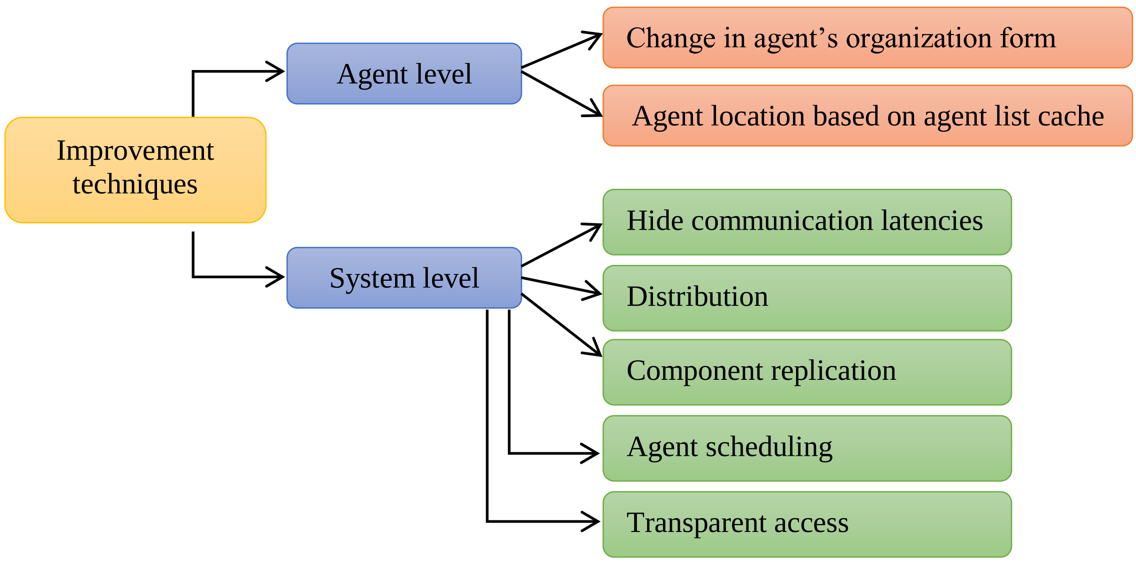

Figure 2.13 Performance and scalability techniques in Multi-agent systems (Koster, 2011)

8.1 Agent Level Improvements

Approaches of this granularity are aimed at the organization and implementation of agents.

8.1.1 *Change in Agent's Organization Form*

In multi-agent systems, agent can be organized in different forms. To investigate if a given form can increase system's performance and scalability, the authors in (Turner and Jennings, 2001) conduct a study on this matter. They believed that in large, open and dynamic multi-agent systems, the number of agents can increase and fluctuate significantly. Therefore, such systems need to be both *self-building* (able to select the best organizational structure at runtime) and *adaptive* (able to change their structure when their environment changes). The authors have defined three organizational forms for trading agents system (customers and suppliers) based on agents' interaction. In the first form, each customer can communicate with each supplier and vice versa, but they are both oblivious to other agents of the same role which means that agents of the same role cannot share information or train groups. The second form is similar to the previous one; except that it is also possible for agents of the same type to communicate with each other which mean that the agents can share task information and form groups. The third form is obtained by adding an intermediary agent responsible for collective tasks to the second form. This agent performs intermediate functions, such as matchmaking, facilitation, etc., thus relieving the other agents of this work. The authors concluded that changing the form of the organization can increase the scalability of the system, by sharing tasks between agents and reducing the workload on other agents by introducing intermediaries.

8.1.2 *Agent Location Based on the Agent List Cache*

In order to communicate, agents must be able to locate each other. Agents' addresses are usually saved in a centralized service or agent, in which all the agents in the system search for agent's locations. This process can be time consuming, therefore to increase performance and scalability; each agent can use caching lists to store the agent's addresses which it usually interacts with (Aguirre et al., 2000). If a sender agent could not find an agent's location in its cache list, then a request for this agent's address is passed to the other agent to look in their cache lists recursively (ID can be used to avoid duplicated requests) until to address is found or all the agents have checked for it.

8.2 System Level Improvements

In (Brazier et al., 2001), the authors have discussed three scaling techniques that can improve systems' performance which are: (1) hiding communication latencies, (2) distribution, and (3) replication. While (4) agent scheduling and (5) transparent access were discussed in (Koster, 2011).

8.2.1 *Hiding Communication Latencies*

The authors in (Brazier et al., 2001) proposed that agents can perform other tasks while they are waiting for messages or responses. This technique is applicable in the case of geographical scalability, when agents need to span to a wide-area network. An important condition for this technique to work is that the agent must be interruptible when the response is received.

8.2.2 ***Distribution***

Component distribution (Brazier et al., 2001) involves partitioning a large set of data into smaller parts that can be distributed on multiple servers. Furthermore, in order to reduce the load on one host, the agents can be distributed on other machines.

8.2.3 ***Component Replication***

Component replication (Brazier et al., 2001) is a known load balancing technique to minimize bottlenecks. Replicating data, service or an agent reduces the load and increases response times especially when placing these replicas close to where they are needed. Decentralizing components is more reliable and more resilient to failures.

8.2.4 ***Agent Scheduling***

Agent scheduling allows the execution of a large number of agents with better performance. By deactivating agents with no tasks to perform, calculation resources will be liberated and only memory resources are occupied. All the available resources are then at the disposal of the active agents. Multiple scheduling policies can be applied, based on rankings, heuristics or statistics (Koster, 2011).

8.2.5 ***Transparent Access***

This technique was proposed in (Deters, 2001). It is achieved by adding a resource management layer to the system which distributes agents on different machines. The CPU, memory and communication resources of every machine are divided into what the authors called *atomic cells*. The atomic cells are considered as agent containers, each one with a defined resource capacity. Scalability can be achieved only if the agent host has transparent access to all the distributed resources. Each atomic cell has one container in which only one agent can reside.

A summary of the advantages and disadvantages of the previous methods is presented in

Table 2.6.

Table 2.6 Summary of performance and scalability enhancement methods in MAS (Koster, 2011)

Method	Advantages	Disadvantages
Change agent organizational form	Can reduce communication overhead and increase efficiency.	Organizational form must fit the problem that the agents are modeling.
Locate agents based on agents caching list	No need for middle agents. Is suitable for heterogeneous MAS.	May not be suitable for unstructured MAS.
Hiding communication latencies	Agents can perform other tasks when waiting for a response.	Agents must be interrupt-able and immediate communication must not be required.
Component distribution	The components are distributed, thus spreading out the workload.	Must often be combined with hiding of communication latencies, to ensure performance increase. The components must be manually distributed. Components are bound to a single machine.
Component replication	Data is close to the agents. Bottlenecks can be prevented, by replicating heavily used components.	Possible data inconsistencies. Load balancing required.
Agent scheduling	Can increase performance, by efficient scheduling of active and inactive agents.	Useless for large numbers of proactive agents. Computationally expensive.
Transparent access	Increases location independence. Increases effectiveness of other agent-level methods.	Can increase overhead.

9 Conclusion

Perfective maintenance is the most requested and most efficient type of maintenance to improve their maintainability, performance and scalability. In this chapter, we found that few techniques and tools have been proposed to achieve these goals in multi-agent systems, and that the majority of them have been designed for the preliminary phases of development or require redevelopment. We also noticed the presence of some tools for the analysis and profiling of MASs or just for their monitoring, while a total lack of automatic or semi-automatic tools to analyze and improve the performance and scalability of a multi agent system already existing is clearly observed.

Chapter 03

Overview of Used Tools and Techniques

Chapter 3 Overview of used Tools and Techniques

1 Introduction

In this chapter, we will briefly introduce the tools and techniques used for the perfective maintenance approaches that we have proposed. First, we present the technique of Aspect-oriented programming (AOP) and its AspectJ tool. The constructs of AOP allow us to change the design of the matchmaking service without modifying its original code. AOP has been used by the agent community for different purposes (Chebout et al., 2019; Garcia et al., 2007; Robbes et al., 2004). The second technique used for enhancing agents' sociability is Machine Learning along with the Weka toolkit. Weka offers a Java library of machine learning algorithms which can be used to enrich certain agents with learning capabilities. Machine learning algorithms have been applied previously for multi-agent systems to enhance agents' properties (Kensler and Agah, 2008; Khalil et al., 2015a, 2015b; Lewenberg, 2017; Mao et al., 2020; Nguyen et al., 2018; Tuyls and Weiss, 2012).

2 Aspect-Oriented Programming

Aspect-Oriented Programming (AOP) (Kiczales et al., 1997) is a programing paradigm that offers better code modularity by separating functional concerns from non-functional ones such as logging, authentication, resource management, performance and storage management, etc. (Kiselev, 2002). This paradigm was proposed by the Xerox Company due to two major problems that exist in object-oriented applications which are: code dispersion and tangling. AOP is an extension of programming language and can be applied to Java (Gradecki and Lesiecki, 2003), C/C ++ (Spinczyk et al., 2002), C # and .NET (Schult and Polze, 2002). It is then possible to apply it on procedural programming or object-oriented languages.

2.1 Code Tangling

Code tangling (**Figure 3.1**) is caused when a module is implemented to address multiple concerns simultaneously. Developers often take into consideration multiple concerns when implementing a single module such as business code, logging, performance, synchronization, security, etc. (Laddad, 2003)

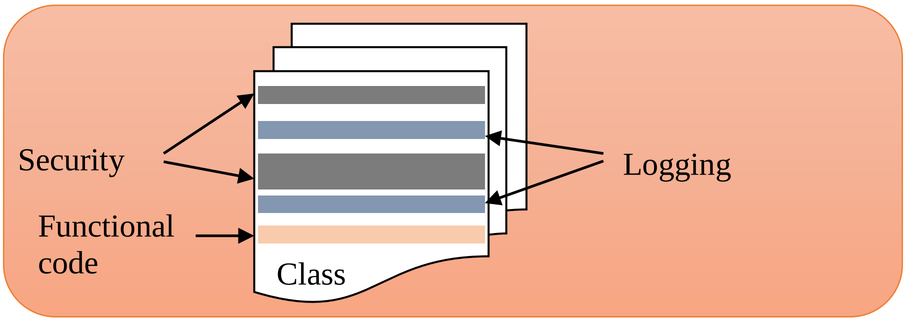

Figure 3.1 Code tangling caused by multiple simultaneous implementations of various concerns (Laddad, 2003)

2.2 Code Dispersion

In object-oriented programming, method invocation is the main mechanism for objects interactions. Since that a method can be invoked within many classes, any change in its signature requires an update in all the classes where it was invoked. This phenomenon of code dispersion or scattering (**Figure 3.2**) is an obstacle to the maintenance and development of object-oriented applications. Any change in the way of using a given method leads to numerous and costly errors.

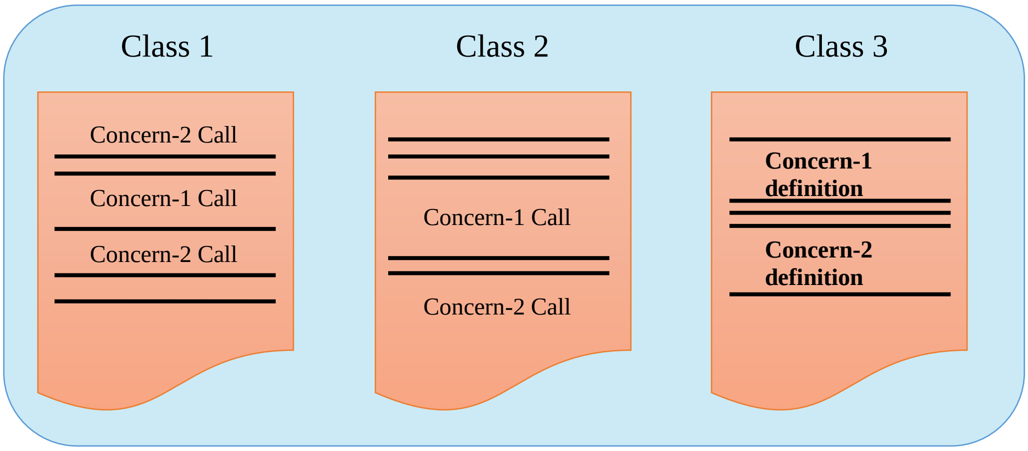

Figure 3.2 Code dispersion of Concern-1 and Concern-2

2.3 Benefits and Disadvantages of AOP

Aspect-oriented programming is considered to be the complement of object-oriented programming (Kurdi, 2013). It reduces code tangling and scattering in applications leading to (Abdul Hameed et al., 2019):

- **Better modularity and code quality**: the code is more readable and easier to understand since that each concern is in a separate module,
- **Easier maintenance and evolution**: change in a given crosscutting concern will take place only in its aspect while it affects the whole system. New concerns can be added and integrated with the original system more easily,
- **Better reuse:** a crosscutting concern can be reused without worrying about its environment and regardless of the business or field of application. An aspect can target a wide range of applications especially by using wildcards and filters in point cuts declarations,
- **Productivity gain**: developers will focus only on the concerns in hand which enhance tasks distribution and parallelization,
- **Late binding of design decisions:** since that the concerns are separated from each other design decisions can be postponed for future requirements.

Some of known disadvantages of using AOP are:

- In development groups, classes' programmer must not alter the declared join points such as methods' renaming or removing without informing the aspects' programmer (Abdul Hameed et al., 2019).
- Debugging and testing need specialized tools like *AspectJ Development Tool* (AJDT) to rapidly visualize and switch from classes to aspects and vice versa.

2.4 AspectJ

AspectJ is an aspect-oriented extension for Java language (Kiczales et al., 2001). Compared to the object-oriented languages, AspectJ provides a better way to modularize cross-cutting concerns in complex systems. New constructs are added to Java such as aspects, join points, advice, point cuts and intertype declarations (**Figure 3.3**).

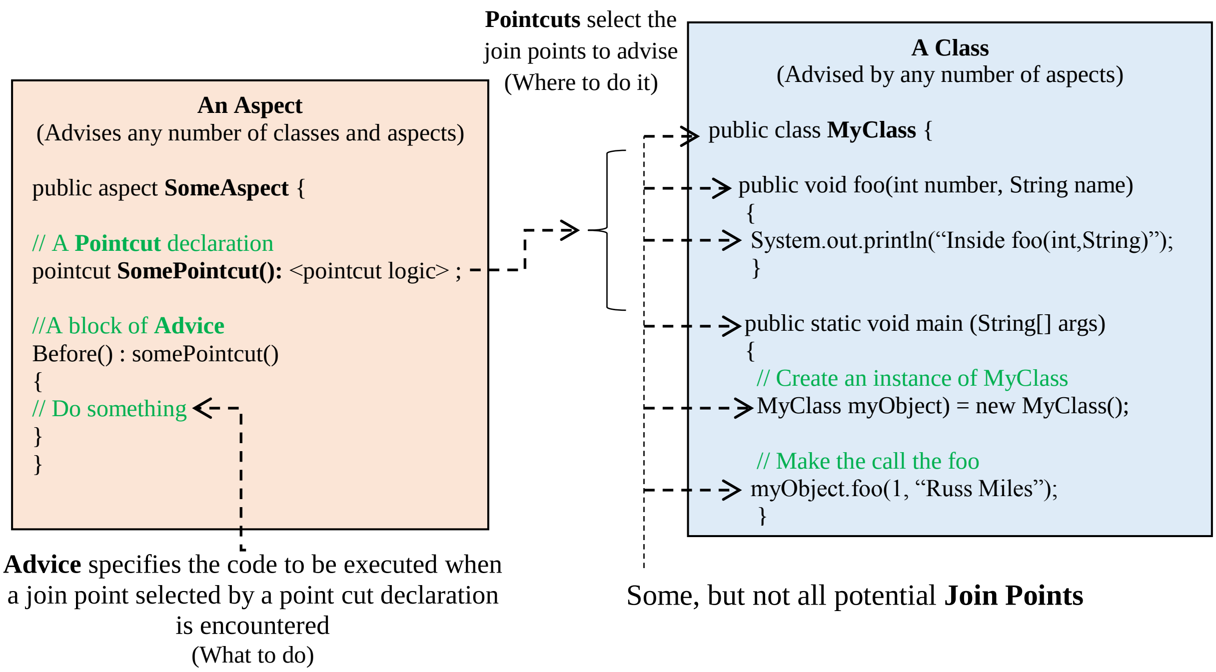

Figure 3.3 the relationships between aspects, pointcuts, and advice (Miles, 2004)

In AspectJ, an aspect is a module which contains the crosscutting implementation. A given functionality that may crosscut several classes can be encapsulated in one aspect. Several points in programs' executions called Join Points can be located by AspectJ such as method calls and executions. A collection of join points and their parameters can be referred to as Point Cuts (**Table 3.1**). An advice is a construct similar to a method which can be used to add functionalities that can be executed before, after or around a given join point. Furthermore, the execution of a given method can be replaced by the advice if it is an around type. Intertype declaration is a mechanism that allows aspects to insert new public or private fields and methods into classes (Xie et al., 2006).

Table 3.1 Join points and their point cuts designators in AspectJ (Laddad, 2003)

Join point	Join point description	Point cut syntax
Method execution	Method body	execution(MethodSignature)
Method call	Method invocation	call(MethodSignature)
Constructor execution	Execution of an object's creation logic	execution(ConstructorSignature)
Constructor call	Invocation of an object's creation logic	call(ConstructorSignature)
Class	Class loading	staticinitialization(TypeSignature)

initialization		
Field read access	Access to read an object's or a class's field	get(FieldSignature)
Field write access	Access to write an object's or a class's field	set(FieldSignature)
Exception handler execution	Catch block to handle an exception	handler(TypeSignature)
Object initialization	Object initialization in a constructor	initialization(ConstructorSignature)
Object pre-initialization	Object pre-initialization in a constructor	preinitialization(ConstructorSignature)
Advice execution	Advice execution	adviceexecution()

The aspects and classes are woven together by a static weaving which can identify all the locations of the corresponding join points statically, and a dynamic weaving which can capture only the declared join points reached by the current execution sequence (**Figure 3.4**). AspectJ Development Tools (AJDT) is the AspectJ plugin name that can be imported to projects in order to exploit its library.

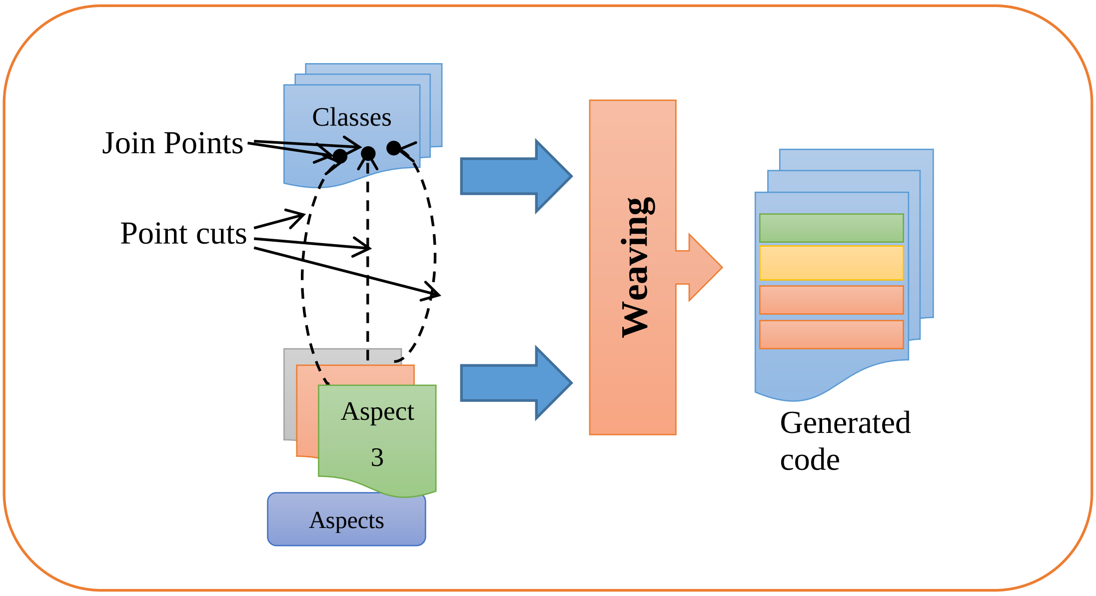

Figure 3.4 Aspect weaving (Retaillé et al., 2004)

A summary of the basic concepts of AOP are presented in **Table 3.2**.

Table 3.2 Summary of AOP basic concepts (Abdul Hameed et al., 2019)

Term	**Description**
Concern	A special goal, concept, or area of interest.
Crosscutting Concerns	Concerns tangled and scattered across many modules in a program.
Aspect	A concern modularization which cuts across several modules.
Core or functional (Base Code)	The non-aspect program section, the business program part.
Advice	Actions taken by an aspect at a special join point.
Join Point	A point through the program implementation as in the method execution or exception management.
Point cut	A predicate which declares the targeted join points.
Intertype declaration	Introducing new methods and fields into a given class from an aspect
Method signature	Contains its access restriction type, return type, full name including class/package, and arguments in their original order.
Wildcards	A set of symbols that allows us to create expressions encompassing multiple join points such as: *, +.
Filtering	Set of Keywords used in the point cut to filter join points such as: &&, ||, *within, this, target,* etc.
Scattered Code	A code regarding one concern spread in several units of the system.
Tangled Code	A code with different concerns interlacing to each other.
Coupling	Dependency between the modules.
Weaving	The procedure of interpolating aspect code into other one, it can be accomplished at build time, load time and run time.

3 Machine Learning and Weka Toolkit

Machine learning (ML) is a form of the Artificial Intelligence domain that enables a system to learn from data rather than through explicit programming (Mueller and Massaron, 2016). Machine learning studies computer algorithms for learning to perform tasks. It is about to do better in the future based on what was experienced in the past. The idea is that machine learning turns data into information (Harrington, 2012) or programs (Raschka, 2018).

3.1 Weka

Weka (Holmes et al., 1994) is a machine learning workbench proposed at Waikato University, New Zealand. Its purpose is to aid engineers to apply machine learning algorithms for real word problems. Weka offers a GUI and a Java library of machine learning algorithms that can be imported to Java projects.

3.2 Categories of Machine Learning

Different problems can be raised with different situations, inputs and outputs. In consequence, it is preferable to distinguish the possible tasks that can be handled by machine learning algorithms. Therefore, machine learning was categorized into three broad categories presented in **Figure 3.5**.

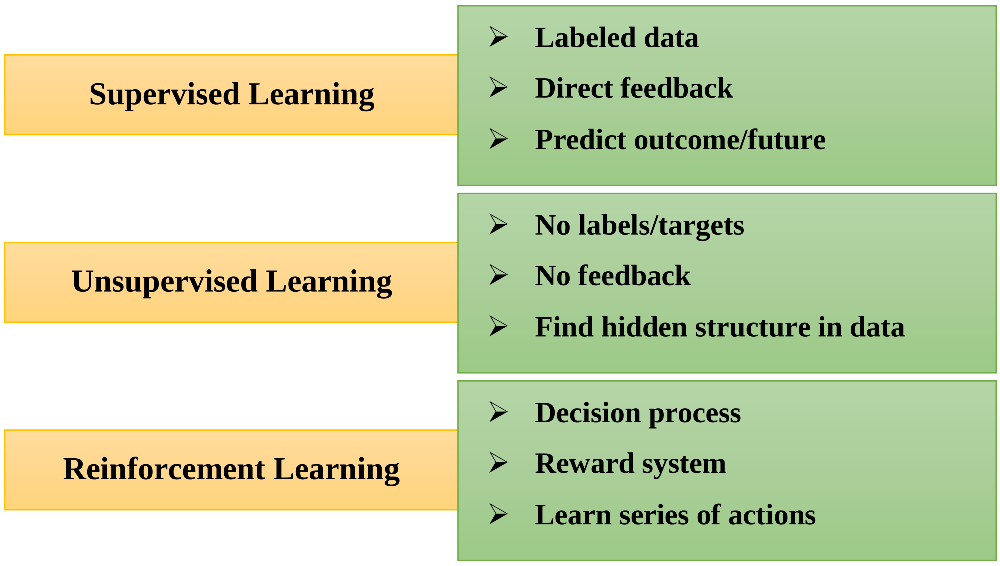

Figure 3.5 Categories of machine learning (Raschka and Mirjalili, 2017)

3.2.1 ***Supervised Learning***

Supervised learning is the subcategory of machine learning that focuses on learning a classification or regression model, which is, learning from labeled training data (i.e., inputs that also contain the desired outputs or targets; basically, “examples” of what we want to predict) (Raschka, 2018).

- ***Regression***: In machine learning, a regression is a prediction where the target’s data type is a continuous number. For example: the prediction of the temperature when it is midday.
- ***Classification***: is the task of predicting a discrete class label. Classification problems with two classes are called binary classifications, while more than two classes are called multi-class classifications. Logistic regression is for classification problems and must not be confused with regression problems.

3.2.2 ***Unsupervised Learning***

Unsupervised learning is best suited when the problem requires a massive amount of unlabeled data. The most common type of unsupervised machine learning algorithm is clustering. This means that the algorithm tries to find and label patterns and clusters in the dataset. Therefore, the supervised learning conducts an iterative process of analyzing data without human intervention (Mueller and Massaron, 2016).

- ***Clustering:*** is the organization of unlabeled data into similarity groups called clusters (Jain et al., 1999).

3.2.3 ***Reinforcement Learning***

Reinforcement learning (RL) is learning by interacting with an environment. A reinforcement learning agent learns from the consequences of its actions, rather than being explicitly taught. It selects its actions on the basis of its experiences (exploitation) and also by new choices (exploration), which is essentially trial and error learning. The reinforcement signal that the RL-agent receives is a numerical reward which encodes the success of an action's outcome, and seeks to learn how to select actions that maximize the accumulated reward over time (Woergoetter and Porr, 2008).

3.3 Basic Concepts in Machine learning

The basic concepts of machine learning that we will encounter in this chapter are:

- **Dataset**: is a set of data instances which contains important features to solve the problem,
- **Instances**: are the rows of the dataset. Every instance has values of the selected features and the target. The size of the dataset is represented by *m*,
- **Features, attributes, dimensions or variables**: important properties of the problem which can influence the outcome. These are inputs of the machine learning algorithm to help it learn. They are usually represented with the letter *X*. If the dataset has only one feature "n=1" then *X* represents a vector, if there are more than one attribute "n>1" then *X* is a matrix of features, where x_j^i is the value for feature i of instance j,
- **Target or label**: is a feature or a set of features in the dataset which is used for learning and predicting. It is also known as the ground truth or the class and represented by the letter *Y*,
- **Data types**: the selected features may have different types (Kamath and Choppella, 2017):
  - ✓ Categorical or nominal like: eye color: green, black,
  - ✓ Continuous or numeric like: temperature, weight,
  - ✓ Ordinal like: garment size: small, medium, large.
- **Model**: the model is the result of the trained algorithm. Based on this model, we can predict future outcomes,
- **Evaluation metrics**: different performance metrics can be used to evaluate the trained model such as accuracy, precision and recall (Seliya et al., 2009).

3.4 Practical problems in machine learning

It is mandatory to take into consideration the various problems that may occur during a machine learning process. Avoiding them before starting the process will help in building a good model. The best-known ones were presented in (Kamath and Choppella, 2017) as follows:

3.4.1 *Noise and Data Quality*

Some of the important issues that can be presented in datasets are duplicated, missing or incorrect values. The data quality and format must be carried carefully to build good models.

3.4.2 *Imbalanced Datasets*

An imbalanced dataset contains a large number of instances in a given class compared to the other. This must be considered because it does affect the obtained model and how its performance evaluation must be done.

3.4.3 *Data Volume, Velocity and Scalability*

Datasets can be found in large volumes as instances or as real-time streaming data at high speeds. In this case, hardware and algorithms limitations can prohibit the learning process from using the entire dataset. In such cases, sampling can be helpful to reduce the dataset size to fit the available resources.

3.4.4 *Overfitting and Underfitting*

These problems are very common when applying machine learning. Overfitting happens when the model is not generalized properly and fits the training set too well but does not perform well on new unseen data. Underfitting can occur if the model does not fit the training data well which leads to a poor performance when applied to unseen data.

3.4.5 *Curse of Dimensionality*

This problem was introduced in (Bellman, 1961). Increasing the number of features does not mean better learning. In high-dimensional data, the performance and the scalability become an issue where the amount of data we need to generalize correctly grows exponentially. If too many features are involved, then algorithms can miss out the relationship between the instances. In clustering problems for example, this can create more meaningless clusters.

3.5 Supervised Machine Learning Process

In order to apply a supervised machine learning process, there are several steps that must be performed. After recognizing the problem in hand, a dataset must be gathered then analyzed and cleaned. Afterwards, the dataset must be divided into a training set and a test set to be used for the selected algorithm. This latter has to be appropriate for the type of the classification or regression problem. The training set is used first for training the algorithm where learning parameters are tuned continuously to enhance the model. Next, the built model will be evaluated using the test set and performance metrics before its deployment (**Figure 3.6**).

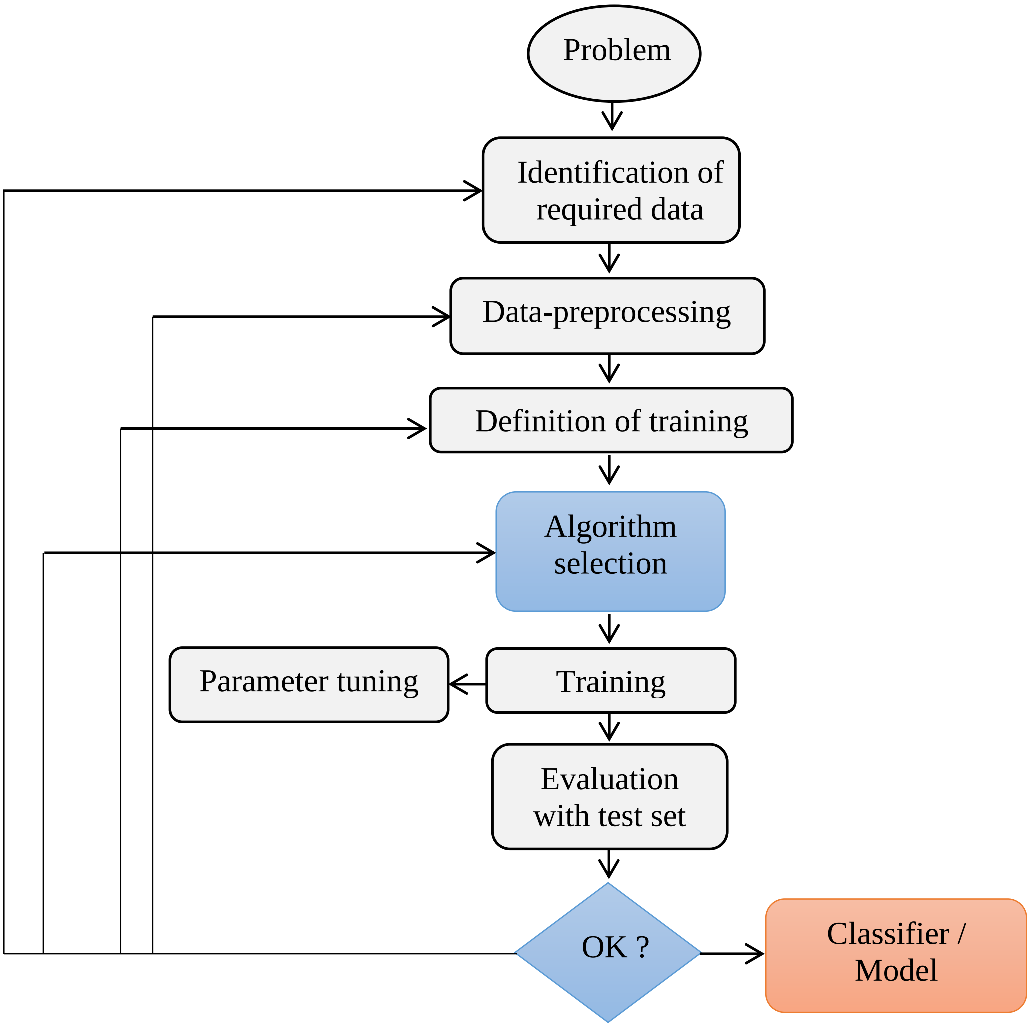

Figure 3.6 The process of supervised machine learning (Akinsola, 2017)

3.5.1 *Data Collection*

Data quality is very important for the ML process. It must be accurate and relevant to the problem we are about to solve. Data usually represents the system's history gathered from sensors, logs, results, etc. The data must be sufficient in terms of size with a good quality (avoid duplicated and data that follows a particular trend, etc.). For a supervised learning, the data must be labeled correctly.

3.5.2 *Features Selection*

This step requires a statistical data analysis to identify the features' combination that controls the outcome of the model. A feature must be relevant to the problem in hand, but this does not mean that the feature must be correlated with the class. If a feature cannot determine the outcome then it does not mean that it is useless, because it can be very efficient for the model when it is used with other features. To reduce the size of the dataset, some features can be represented by one feature, if they are correlated to each other.

3.5.3 *Preparing Training and Test Sets*

In order to build machine learning models, the dataset must be divided into two subsets called the *Training Set* and the *Test Set*. The training set is used to train the algorithm and obtain a Machine Learning (ML) model, whereas the test set is used to evaluate its performance. An important condition here is that the model must be evaluated using new data (test set) which was not seen during the training phase (Mishra, 2020).

The training set is usually 70% of the dataset, while the test set is the remaining 30%. The training set can be also split into two subsets: the training set and the validation set which is only used to tune the model. It is important to ensure that their items are well distributed and none of them follows a particular trend. Therefore, splitting the dataset into two subsets using percentages may not be the best way to create training and test sets. (Mishra, 2020)

In machine learning, underfitting and overfitting as explained previously are two important issues that must be avoided. To avoid such problems, *K fold Cross-validation* method (Stone, 1974) was proposed to create training and test sets from one dataset. The idea behind it is to shuffle the dataset randomly and divide it into a number K of smaller sets called *Folds*. The K fold cross-validation technique is used to obtain an estimate of the model's performance, by exploiting the entire dataset. This is achieved by doing several test iterations on different training and test sets, then averaging the results. The validation consists of randomly segmenting the initial dataset into K disjoint subsets numbered from 1 to K. At every test's iteration, only one subset is used as a test set while the other K-1 subsets are used as training set (**Figure 3.7**).

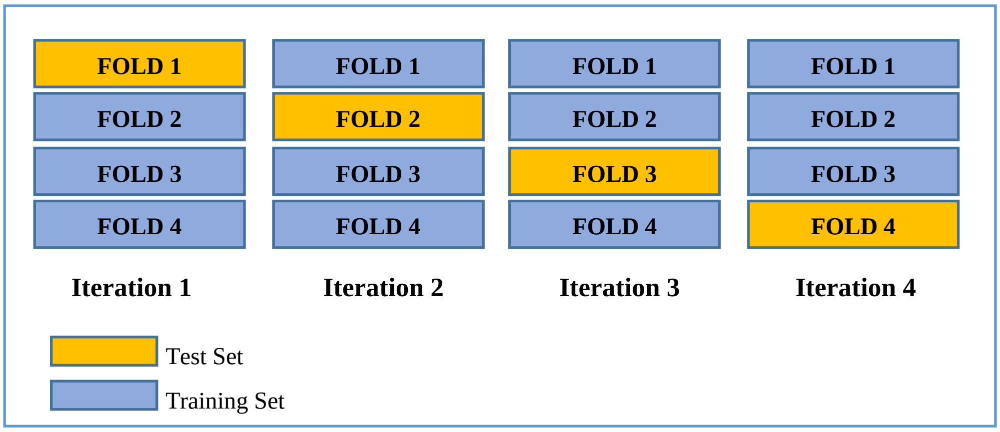

Figure 3.7 Cross validation using 4 folds (Mishra, 2020)

3.5.4 *Selecting a Machine Learning Algorithm for Classification*

Several machine learning algorithms are available for different classification problems. The appropriate algorithm also called as *classifier* can be selected depending on the classification types (binary, multi-class or multi-label classification) (**Table 3.3**), availability of sufficient training data, training time, etc. (Mishra, 2020). The binary classification which we have encountered in this thesis has only two possible outcomes while the multi-class classification has more than two possible outcomes. The multi-label classification has two or more class labels.

Presenting machine learning algorithms goes beyond the subject and the interest of this thesis. We are only interested on the effect of such discipline to enhance multi-agent systems. Therefore, we are not going into their details, and keep it simple by deploying them in the way they were offered by Weka toolkit (default parameters).

Table 3.3 Classification types with some of their corresponding algorithms

Classification Type	Example	Specialized algorithms
Binary Classification	Email spam detection (spam or not).	Logistic Regression k-Nearest Neighbors Decision Trees Support Vector Machine Naive Bayes
Multi-class Classification	Face classification.	k-Nearest Neighbors Decision Trees Naive Bayes Random Forest Gradient Boosting
Multi-label Classification	A picture with multiple objects (Apple, person, car, etc.)	Multi-label Decision Trees Multi-label Random Forests Multi-label Gradient Boosting

3.5.5 *Model's Performance Evaluation*

The built model is not yet ready to be used for classification tasks. We must ensure that this version which is called the *baseline model* performs correctly on new unseen data. In order to do so, the results of the k folds cross-validation method are used to evaluate its performance.

Different metrics for models' performance evaluation can be calculated using the confusion matrix (**Table 3.4**). This latter contains four *primary metrics* (TP, FP, TN, FN) which help visualize the distribution of the correctly and misplaced instances. The rows of a confusion matrix represent the instances in the actual class, while the columns represent the instances in the predicted class. Since this is a binary classification, only two classes are involved which are the positives class (a=1) and the negatives class (b=0).

- ✓ TP (true positives): The number of times the model predicted a positive outcome and the prediction was correct,
- ✓ FP (false positives): The number of times the model predicted a positive outcome and the prediction was incorrect,
- ✓ TN (true negatives): The number of times the model predicted a negative outcome and the prediction was correct,
- ✓ FN (false negatives): The number of times the model predicted a negative outcome and the prediction was incorrect.

Table 3.4 The Confusion Matrix where actual elements in rows and predicted elements in columns

		Predicted	
		Positives (a)	Negatives (b)
Actual	Positives (a)	TP	FN
	Negatives (b)	FP	TN

Furthermore, *secondary metrics* can be computed using the primary metrics to better evaluate the performance of the model. The best-known ones are: *accuracy, precision, recall and F-measure*. More performance metrics such as *ROC* and *AUC* will be encountered later in **Chapter 6**.

Accuracy (Eq. 3.1) is the ratio of the correct predictions on the total number of all predictions. *Recall (Eq. 3.2)* is the number of the correct predictions in a given class divided by the total elements of that class. *Precision (Eq. 3.3)* is the number of the correct predictions of a class divided by the total of predictions of the same class. The *F-Measure (Eq. 3.4)* is the harmonic mean of precision and recall which gives an overall score that depicts how well our model is performing.

$$Accuracy = \frac{(TP + TN)}{(TP + FP + TN + FN)} \qquad \textit{(Eq. 3.1)}$$

$$Recall = \frac{TP}{(TP + FN)} \qquad \textit{(Eq. 3.2)}$$

$$Precision = \frac{TP}{(TP + FP)} \qquad \textit{(Eq. 3.3)}$$

$$F - Measure = 2 \times (\frac{Recall \times Precision}{Recall + Precision}) \qquad \textit{(Eq. 3.4)}$$

4 Conclusion

In this chapter, we have presented a brief overview on the basic concepts of the tools and techniques used in our contributions. The aspect-oriented programming was the first to introduce. We showed its benefits in software development and how AspectJ can be used to capture, change and replace executions in a program. Afterwards, a brief introduction to machine learning was presented. We presented the concepts necessary to perform a machine learning process and create good prediction models.

Chapter 04

State of the Art

Chapter 4 State of the Art

1 Introduction

In this chapter, we will present the most related studies for enhancing Multi-agent systems. In these studies, authors did not use the term perfective maintenance. Instead, they have used terms like improvement and enhancement even that all of them have the same purpose. Performance, scalability and software qualities in general get decreased in certain situations due to hardware or software reasons in which we are interested.

Multi-agent systems evaluation and enhancement has been the topic of many researches. Very recent works have proposed methodologies and approaches to evaluate them (Ali et al., 2017; Nguyen et al., 2011a; van Lon and Holvoet, 2017), while others have proposed approaches to enhance some of their properties like deliberation (Rens and Moodley, 2017) (Yao and Logan, 2016) (Faccin and Nunes, 2015), cooperation (Wang et al., 2016), communication (Kadera and Novák, 2017), reactivity (Dam et al., 2013), and other various properties (Al-Yaseen et al., 2016; Du et al., 2017; Hsieh, 2017; Kadera, 2014; Li et al., 2018, 2016; Trejo-Sánchez et al., 2018).

This chapter is divided into two sections; the first one is about the state of the art related to the perfective maintenance approaches addressed to the Matchmaking service (Cao et al., 2001). The second section is dedicated to the state of the art related to agent's sociability and interaction protocols (Contract-Net protocol) enhancements.

2 State of the Art for the Matchmaker Service Enhancements

In MAS, agents need to interact in order to achieve their tasks. Therefore, interaction is an important property in agent societies. An agent is able to interact with another one only if it knows its location which can be found using a service called the *Yellow Pages*. The more agents search for each other using this service, the more performance decrease will result due to the overload of the requests. In this case, the MAS will not be able to respond to the search requests in a proper time which increases exponentially. These delays will propagate to other processes leading to a very slow system, to exceed timeouts of other services and even to a total system breakdown. Unfortunately, only few researches have been done to enhance this service by proposing different architectures.

In (Pascoa et al., 2017), the authors have focused on how to store and propagate information within *Cyber Physical Production Systems* (CPPS) using a new directory facilitator different than JADE's DF. They proposed an agent called *Capability Dissemination Agent* (CDA) which is a redundant, decentralized and resilient extension of the available JADE's DF.

Several CDAs can be present in the agent platform where each one contains registered local capabilities of the assembly elements near to it. Each CDA must also know the local capabilities registered in all the other CDAs.

In **Figure 4.1**, three strategies have been discussed for CDA deployment. The first strategy appeared in the left side of **Figure 4.1**, only one CDA is deployed for the whole system. This will not be different than the JADE's DF because the CDA is still a centralized agent which may be subject to bottlenecks and system failure. The second strategy in the right side of **Figure 4.1**, every module has its own CDA, which eliminates bottlenecks and single point failure but increases the communication load. Because each CDA will transfer the received search request to other CDAs in case it does not have a corresponding registered skill (service). The best strategy is presented in the middle of **Figure 4.1**, which offers a balance between the redundancy and communication load.

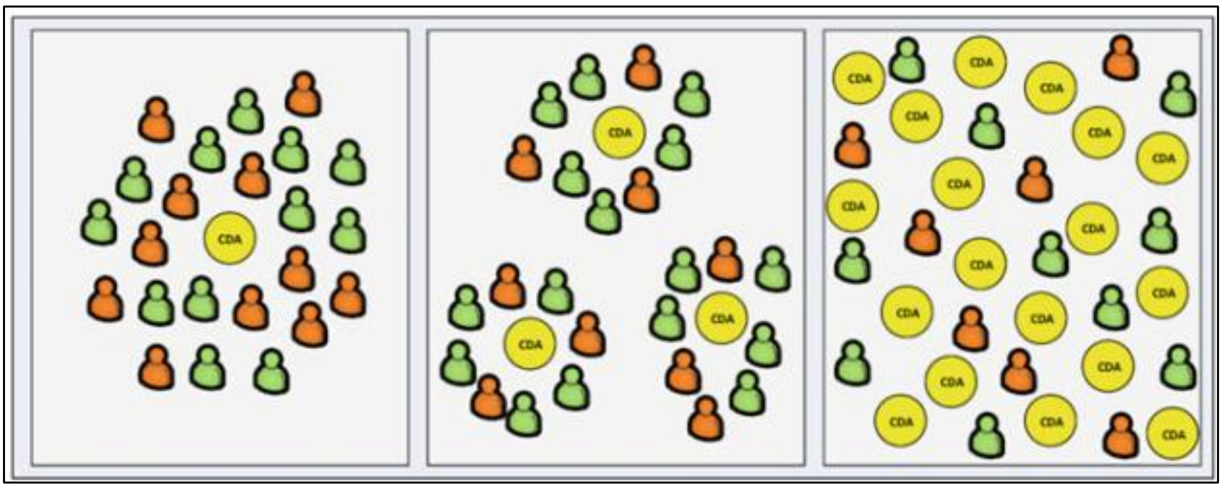

Figure 4.1 CDA use case overview (Pascoa et al., 2017)

To evaluate their proposal, the authors only addressed the ability of the CDAs to update their knowledge about the state of other CDA's if they were down. Unfortunately, they didn't evaluate the performance of the CDA which it should have been done considering the problem that they have stated about bottlenecks and performance degradation of JADE's DF.

The authors in (Del Val et al., 2014), have proposed a decentralized directory service where agents are linked together in a preferential-attachment network structure. Agents collaborate to find services when they are requested. Using the *Homophily* concept, the authors were able to propose an algorithm called *Choice Homophily Navigation* (CHN) which propagates service search queries through the network in order to find agents with the desired services. They used homophily to link agents to each other based on the similarities of their services and roles. Homophily is a social feature present in complex networks that acts as a self-organizing criterion to create decentralized and self-organized structures.

In **Figure 4.2**, agents are organized in a preferential-attachment network structure in which agents with the same roles have the same colors. In the left side (a), agent a_i has three neighbors a_k, a_j and a_n linked to them directly based on the choice of homophily function (CH) which calculates the best similar agents in terms of similar services and roles.

In the right side (b), agent a_i wishes to find an agent that can help it achieving its goals that offers a service s_6 and a role r_5. Therefore, agent a_i creates a search query q= (s_6,r_5). The query can only be forwarded for a limited number of times by a *Time To Live* (TTL) constraint. The query is forwarded to a given neighbors only if the TTL is not exceeded. Otherwise the service discovery process fails.

Agent a_i selects the most promising neighbor based on (1) the homophily between the neighbors and a hypothetical unknown target agent $a_t = (s6; r5; \phi; \phi)$ (i.e., CH (a_k, a_t)) that offers the service and plays the role specified in the query, and *(2)* the degree of connection of the neighbors. This process is repeated until the similarity between local services of an agent and the service in the query is over a certain threshold or the query exceeds the TTL. In the described scenario, the process ends when the query arrives to agent a_v that is similar to the hypothetical target agent a_t that a_i was looking for.

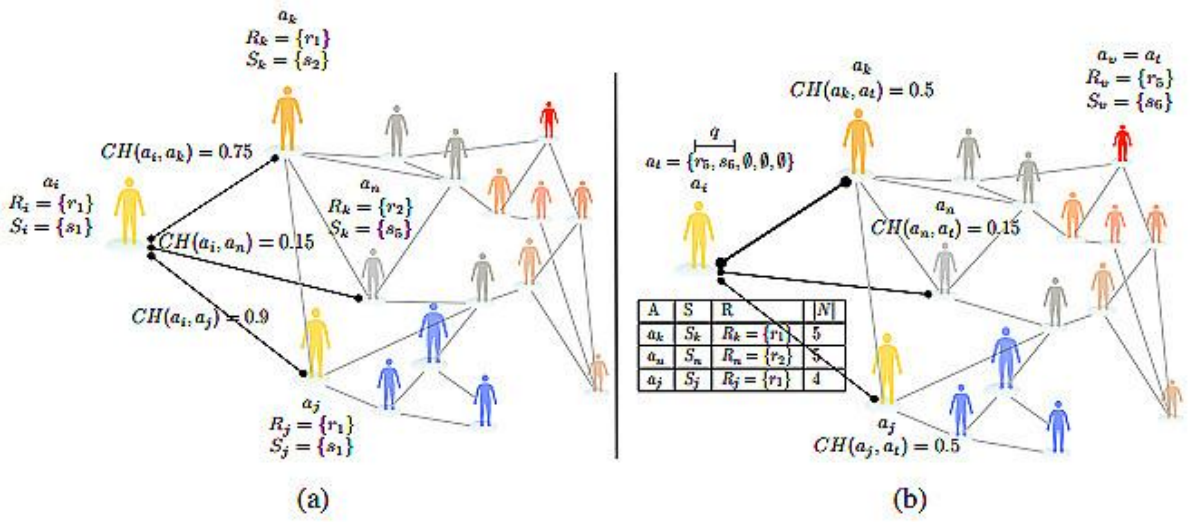

Figure 4.2 An example of a decentralized service discovery system by (Del Val et al., 2014)

To evaluate their proposals, the authors have performed a number of experiments in the context of *Semantic Web Services*. In each experiment, 5000 queries have been generated in the network. Each query is for finding only one agent with the desired service. They used success rate and short paths as performance evaluation criteria. In comparisons with four other similar approaches, their evaluation resulted in better performance and better fault tolerance.

The questions that we ask about such type of decentralized service discovery systems, which can represent their drawbacks, are:

- Does forwarding queries affect agents in performing their own tasks especially when the numbers of queries are increased?
- If the query is sent to a busy agent, does this affect the time spent to obtain the results?
- Does forwarding too many queries have an impact on the communication channel or the network?
- Is it possible for one agent to request more than one service using one single query?
- Can a given agent without a service to offer be part of such network? If it can then how may the homophily be calculated?

In (Han et al., 2012), the authors have proposed a Category-Based Directory Facilitator (CBDF) (**Figure 4.3**), which employs a category based searching method instead of a sequential one which is used by JADE’s DF. According to the authors, sequential search is very slow and overheads can grow proportionally to the number of the registered services.

In this approach, services are categorized using their types and user preferences. In consequence, the search space for a given service search request is narrowed by identifying its category first rather than parsing the whole catalogue sequentially.

Their evaluations showed, in one hand, an important improvement in memory usage every time the number of registered services was increased from 1000 to 100000 registered services compared to JADE's memory usage. In the other hand, the request response time of only one request showed that JADE's DF is better than their CBDF when the number of the registered services is fewer than 4000 services. When the registered services were greater than 4000 services, the CBDF performed slightly better by recording less searching times. The fact that in their evaluations, JADE’s DF has been searching only by service name, while the CBDF has been searching by seven attributes which minimized the search space when the registered services were increased.

Unfortunately, their evaluation did not address clearly the performance of the CBDF when the search requests are increased. Since that their proposed CBDF is still centralized in one agent, this makes it vulnerable to bottlenecks and overload.

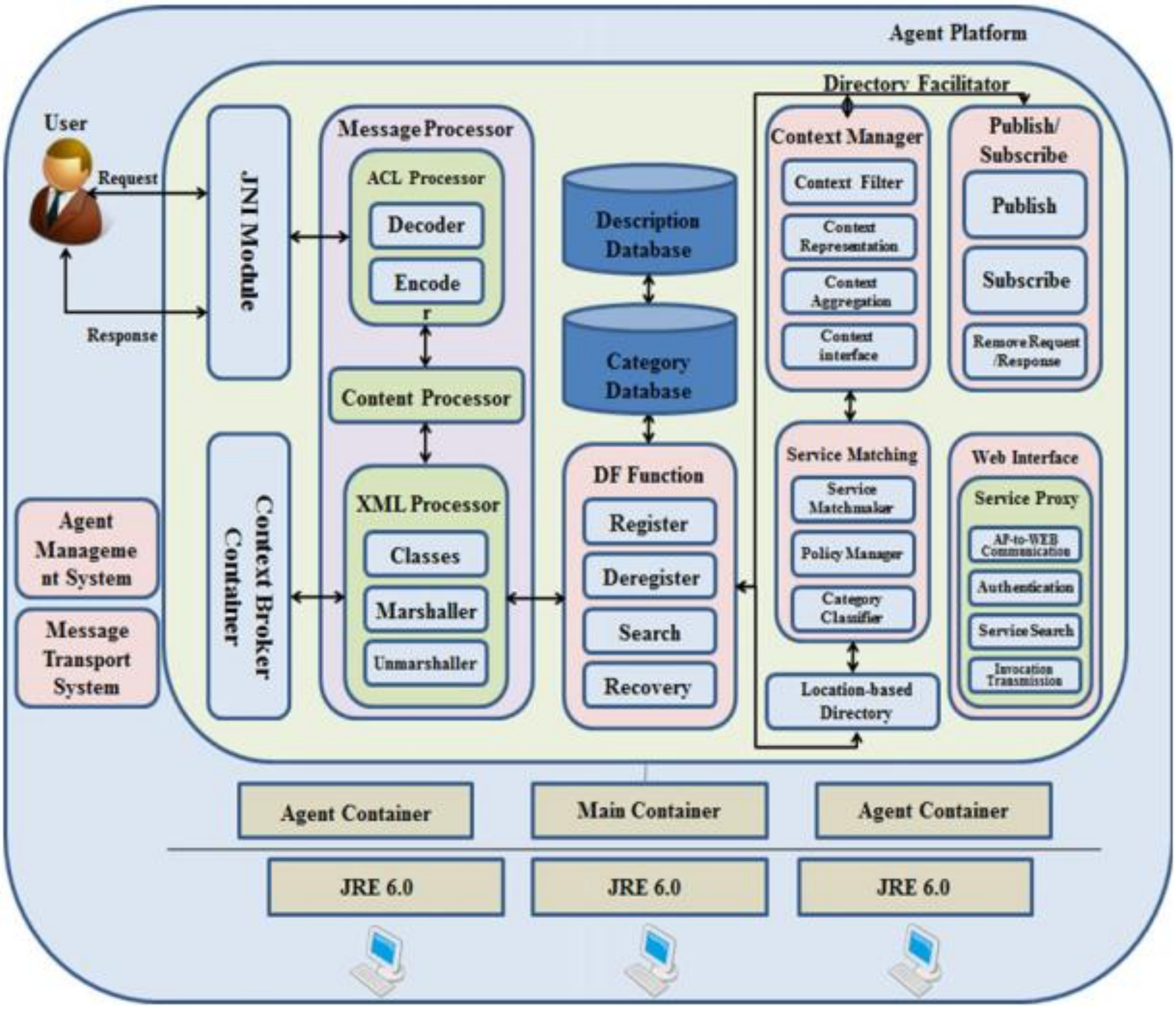

Figure 4.3 The structure of the proposed CBDF by (Han et al., 2012)

In (Kim et al., 2009), the authors have proposed a similar approach to the previous CBDF under the name of Agent-Platform Directory Facilitator (APDF). It simply consists on the use of Oracle Data Base Management System (DBMS) (**Figure 4.4**) to handle DF functions like register, deregister and search. The results were very similar to the CBDF proposal.

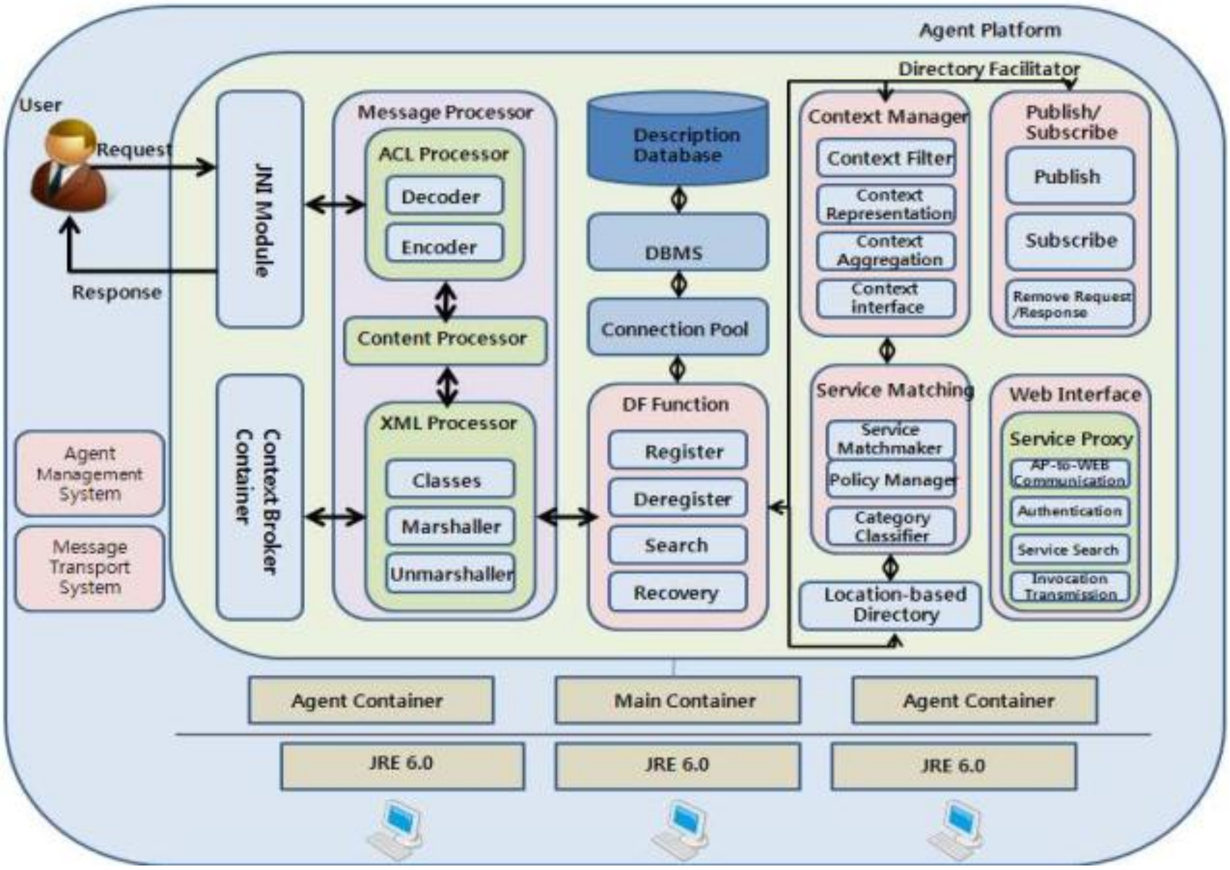

Figure 4.4 The architecture of the proposed APDF by (Kim et al., 2009)

In (Kim et al., 2007), the authors did not address the performance problem of JADE’s DF, instead they have investigated its ability to provide agents with the most appropriate services. Based on policy and context inference mechanisms they have proposed a Context-Aware Service Discovery. Their proposed DF is a part of the CALM agent platform project (Component-based Autonomic Layered Middleware) (Seungwok Han et al., 2006) (Lee et al., 2008). The proposed architecture consists on several modules presented in **Figure 4.5**, in which the Context Manager has two modules. The first one is context filtering which protects the DF from excessive information (flooding) about the service in question. This task extracts context information from register or search requests. For example, an agent provides a service of color printer with 12 pages per minute. Using the context filtering, the context stored in the context repository may be (name “printer”, type “color”, speed “12ppm”). The second module of the context manager is the context inference which creates new context from existing raw contexts based on programmed rules. For example, if a printer with a color type and 600x600dpi resolution then it is considered as picture printing service otherwise it is considered as document printing.

The service matchmaker registers and searches for services using the context provided by the context manager. Services are stored in categories with some policies related to them. For categorization, the category classifier uses language, protocol and encoding parameters. While, the policy manager indicates, for example, the availability periods of the service and the allowed number of uses by a given agent.

The performance evaluation of this architecture was not available in their studies, but only planned as future work. In our opinion, the category-based search will improve the search query time when the number of the registered services increases similarly to CBDF and APDF approaches. But given that the architecture is still centralized, bottleneck problems may still arise when the number of searches increases.

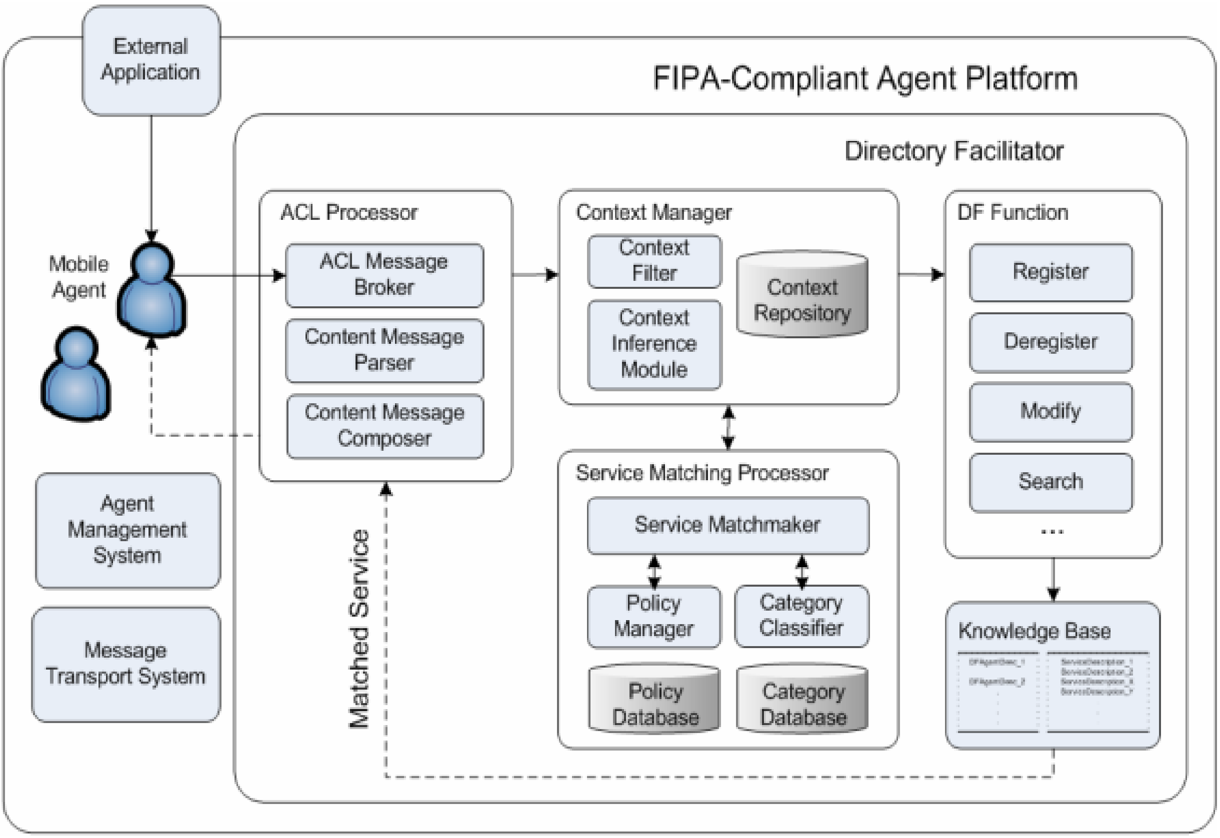

Figure 4.5 The architecture of the Context-Aware Service Discovery by (Kim et al., 2007)

The authors in (Mengistu et al., 2008) have briefly presented their proposed approach to enhance the directory facilitator in JADE platform. The fact that the DF is frequently used by other platforms' services, then any performance problem in DF will affect the other services as well. According to their study, the Lightweight Directory Access Protocol (LDAP) employed in JADE's DF is the reason of its inefficiency, since that it was proven its incapability to handle large requests (Wang et al., 2008). The authors have replicated the DF agent to other containers where each DF has its own catalogue as a cache of registered services which is updated each time any change in agents' states occurs. The authors proposed static directory cache in which the local caches are not refreshed during simulations, and a dynamic directory cache in which the local caches are updated during the simulations. Their results showed a significant increase in DF performance with a constant low response times with 100 to 1000 agents compared to the JADE's DF performance. Unfortunately, their study was not detailed enough, since they did not mention how they implemented their proposal, if the same JADE's DF is used or a different one has been implemented and if any search constraints are involved.

In (Shahid and Hassan, 2008), the authors have focused on the same performance and scalability issue of JADE's DF when the number of search requests is increased. Two approaches were proposed, the first one by replicating the DF agent in each agent container with a global shared memory of service descriptions. The second one was better in its results which consist on the integration of a shared and distributed memory with the replication of the DF in each container (**Figure 4.6**). In this approach, each agent container has its own DF which: (a) receives only requests from agents located in the same container, and (b) connects to a central replication server responsible for the registered services consistency. The results were very significant in terms of decreasing searches' response times and in increasing the number of agents in the system before it crashes due to a JVM out of memory error.

Their enhancement process requires a new package named *Replication.distributeddf* to be imported into the MAS, along with some modifications in the original source code for the fragments which uses the JADE's DF. The authors did not present how their solution was implemented (algorithms, memory type used, etc.), or where the source code can be found. Therefore, it was not possible for us to test and investigate the solution for comparing it with our approach. They validated their approach on an IBM Think Pad T43 machine with a 1.8 GHz (single core) processor and 512 MB of RAM. This configuration is very low and prohibited them to run the system with a larger number of agents or requests where they reached only 70 agents. Therefore, we cannot say that the solution is suitable for large-scale applications unless more tests with much higher number of agents have to be done.

The authors have replaced the original JADE's DF functions (register, deregister, modify and search) with new methods but with different arguments and return types (String), for example *static void Register(String name, String service)* and *static String[] Search(String service)*, instead of the original functions' arguments like *AID*, *DFAgentDescription*, *SearchContsraints*, etc. Therefore, the methods have to be changed in the original source code. Another disadvantage of their proposal is when a large number of agents are located in the same container, and because the search requests can be only sent to the local DF, it takes us back to the initial overload problem where one central DF is responsible for all the requests. Another disadvantage is that their DF agents are not fault tolerant which means that in case they crash, they must restart again and restore the services previously registered; exactly like how the JADE's DF does. Otherwise, the agents of that container cannot start an interaction process any more. Furthermore, this approach has an interoperability problem, where extern agents cannot interact with their proposed DF, but only with the JADE's DF which in this case has no registered services. Another drawback is that their proposal has no support for search constraints which is very important for more search precision. The authors planned to enhance their proposal by incorporating the shared distributed memory principle within the original JADE's DF. Therefore, we believe that the overload will still be improved than the original JADE's DF due to the distributed and the duplicated DFs. But, the search response time will be slower than their first proposal, because the searching procedure in JADE's DF is more complex and requires several actions and concepts like *ACL processing*, *matching*: *AID*, *services*, *ontologies*, *protocols*, etc.; where the services are stored in a database which is slower in reading and writing operations than using a simple memory (arrays, lists, etc.).

In our approach, the DFs are copies of the original JADE's DF; they can be located in any container and receive requests from any agent. These search requests are distributed over all the DF copies evenly. The enhancements are managed by one aspect file using aspect-oriented programming that makes it very easy to apply them in the MAS without reimplementation.

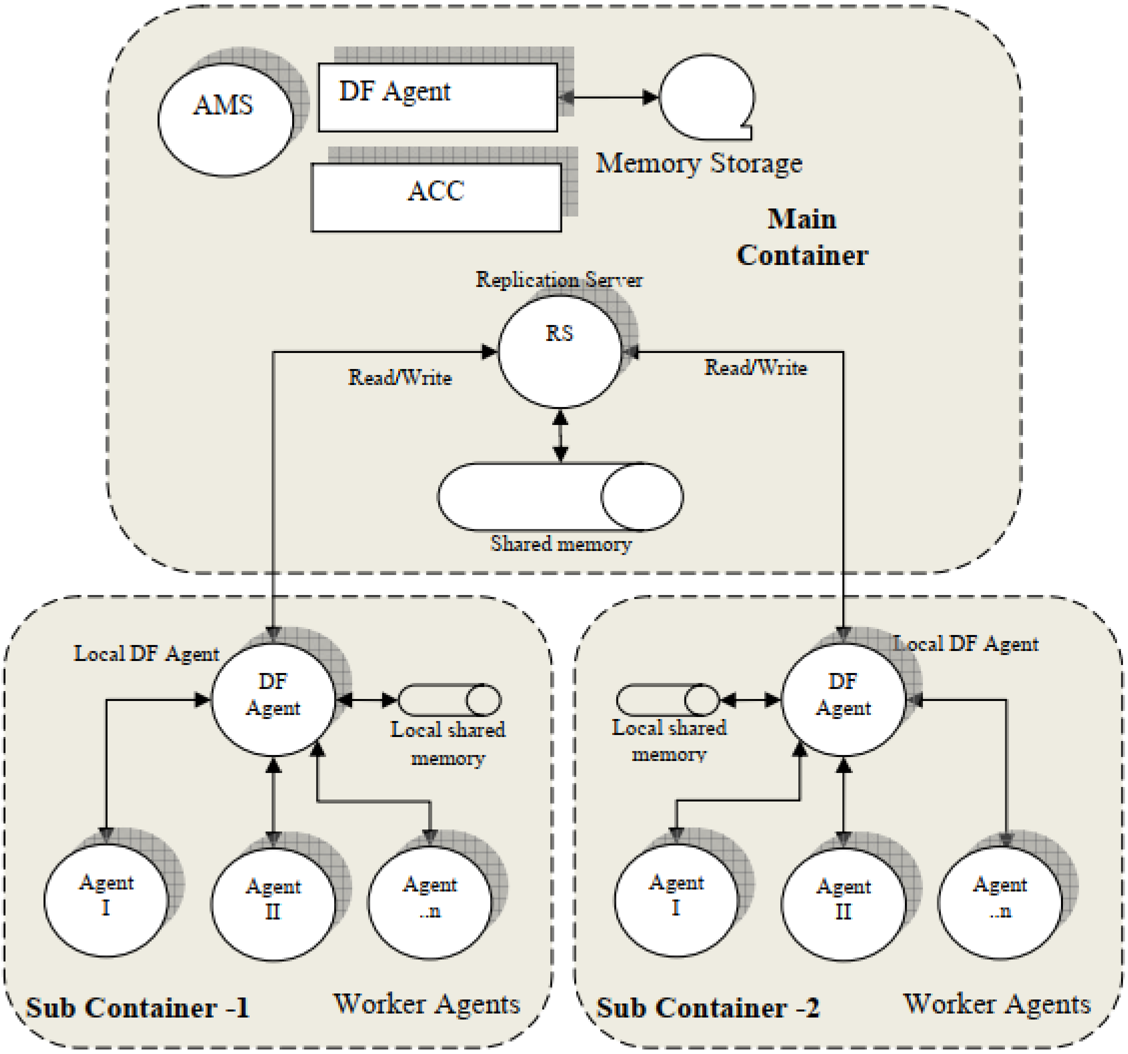

Figure 4.6 Distributed shared memory DF by (Shahid and Hassan, 2008)

A summary of this state of the art is presented in **Table 4.1**. Since that the majority of the proposed approaches were compared to the JADE's DF, in the sense that they attempted to enhance it. Therefore, we preferred to mention some of its properties without looking to its performance evaluations which varies from study to another. We have proposed eleven criteria to compare these approaches.

1. Is the proposed DF based on the default JADE's DF or a new one?
2. Is the proposed DF requires a reimplementation or a change in the original source code?
3. Is the proposed DF replicated or centralized?
4. Is the DF uses sequential or categorized search in its catalogue?
5. How many service search requests have been used to evaluate the DF performance?

6. How many registered services have been stored in the DF during performance evaluation?
7. How many services have been returned by the DF after a search request?
8. Is the DF uses a database or a memory space to store service descriptions?
9. Is multi-constraints search (parameters) possible within the DF?
10. Is there any enhancement in the performance and scalability of the proposed DF compared to others?
11. Is there any reduction in the memory space occupied by the DF or during search requests?

Table 4.1 Summary of the proposed service discovery approaches in MASs

Study	Implemented from scratch or based on JADE DF	Reimplementation required	Replicated DFs	Sequential or categorical search	Increased search requests	Increased registered services	Returned services per query	Using Database / Memory	Multi-constraints search	Performance and scalability enhancement	Memory space enhancement
JADE's DF	/	/	Possible (1 by default)	Sequential	/	/	Unlimited, 100 by Default	Database	Yes	/	/
(Pascoa et al., 2017)	Based on JADE's DF	Yes	Replicated	Sequential	Not evaluated	Not evaluated	Not mentioned (Could be 100 by JADE's default)	Database	Yes	Not evaluated	Not evaluated
(Del Val et al., 2014)	No DF	Yes	Agents' Collaboration	Sequential	5000 requests	No	Only 1 for each request	Memory	No	Enhanced	Not evaluated
Han et al., 2012) CBDF	New DF	Yes	Single DF	Categorical	Only 1 request	From 1000 to 100000	Not mentioned (Could be 100 by JADE's default)	Database	Yes	Not evaluated	Enhanced
(Kim et al., 2009)	New DF	Yes	Single DF	Categorical	Only 1 request	From 1000 to 100000	Not mentioned (Could be 100 by JADE's default)	Database	Yes	Not evaluated	Enhanced
(Kim et al., 2007)	New DF	Yes	Single DF	Categorical	Not evaluated	Not evaluated	Not mentioned	Database	Yes	Not evaluated	Not evaluated
(Mengistu et al., 2008)	Not mentioned	Yes	Replicated	Sequential	From 100 to 1000 requests	Not evaluated	Not mentioned (Could be 100 by JADE's default)	Not mentioned	Not mentioned	Enhanced	Not evaluated
(Shahid and Hassan, 2008)	New DF	Yes	Replicated	Sequential	From 10 to 100 requests	Not evaluated	Unlimited, evaluated with 100	Memory	Only service name	Enhanced	Not evaluated
The Proposed Approach (Zerrougui et al., 2017)	Based on JADE's DF	No, only Aspect file integration	Replicated	Sequential	From 2000 to 50000 requests	Not evaluated (Future work)	Unlimited, evaluated with 100	Database	Yes	Enhanced	Enhanced

3 State of the Art for the Agent's Sociability and CNP Enhancements

In this section, we present the state of the art related to the improvement of agents' sociability by enhancing interaction protocols, precisely the contract-net protocol. Agent's sociability tends to decrease in some situations during FIPA interactions, where participants may not be able to respond to certain types of messages in their proper times leading to failed interactions.

In Contract-Net protocol, each CFP has a fixed deadline for which participants must reply with proposals before it ends. Participants may be busy or blocked during CFPs' announcement which affects negatively on the remaining time available for replies. This leads to unwanted situations like blocked interactions, lost interactions, assigning tasks to less qualified agents, etc. In our proposed approach, participants have been extended with learning capabilities using machine learning concepts. Integrating machine learning algorithms in MASs to enhance agents has been proposed even recently (Kensler and Agah, 2008; Khalil et al., 2015a, 2015b; Lewenberg, 2017; Mao et al., 2020; Nguyen et al., 2018; Tuyls and Weiss, 2012). Unfortunately, none of them have addressed the agents' sociability.

This exact topic has received little attention. Therefore, to avoid talking out of subjects, only the researches related to enhancing CNP in terms of its performance and scalability will be examined.

In (Hudaib et al., 2018), authors have proposed a semi-centralized protocol, which uses a dedicated central agent with specific roles. The proposed protocol extends the well-known Contract-Net Protocol to be used for efficient and trusted negotiation task in multi-agent systems with low communication overhead. They introduced a new agent that plays the role of the system manager (**Figure 4.7**). Initiators ask the manager for a list of participants with good reputations only, to reduce the possibility of failing while implementing the assigned task. Their proposed protocol shows significant improvement in time and communication overhead under various conditions. This enhancement was not addressed for a scalability study; therefore, their proposal needs to be verified with large-scale systems, especially when there is a centralized manager agent that can be a cause for bottlenecks.

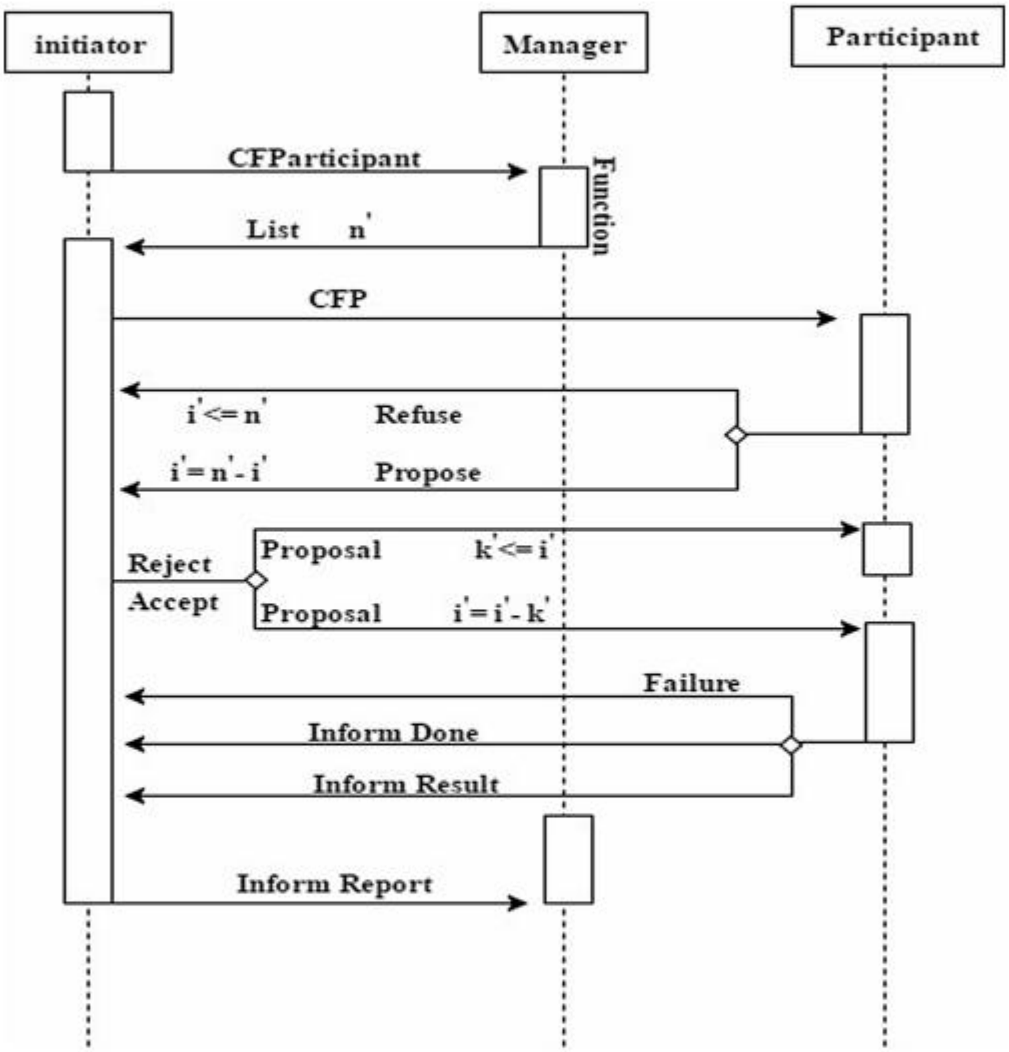

Figure 4.7 Proposed CNP by (Hudaib et al., 2018)

In (Kaur et al., 2013), the authors have proposed and implemented an updated version of CNP called updated-CNP in order to save the overhead of restarting the process of CNP execution between a set of agents willing to coordinate with each other. They have added an additional step into a conventional CNP. As long as the final report is not sent yet, any changes are allowed during the interaction. They justified their approach using a case study of predator-prey system produced with JADE and JATLite platforms. The updated-CNP results were compared with the conventional one by using three parameters, updated tasks, task repetitions and communication overhead. Performance was enhanced in terms of reaching agreements when modifications are requested during the interaction, without repeating the whole process.

In (Yang et al., 2008), the authors have proposed an improvement to Contract Net with Confirmation Protocol (CNCP) (Schillo et al., 2001). A procedure was developed for determining the participant's bidding threshold. The authors have specified a general description formula of participants' *Degree of Availability* (DoA) and its weight coefficient with an enhanced function for bids evaluation (**Figure 4.8**). Results of their simulations show that the proposed improvement can enhance the performance of CNCP in terms of simulation time, information amount, and quality of service. In the CNCP, a new phase has been added to the original CNP. The initiator requests a confirmation from the participants with the best proposals to confirm if they can actually do the job.

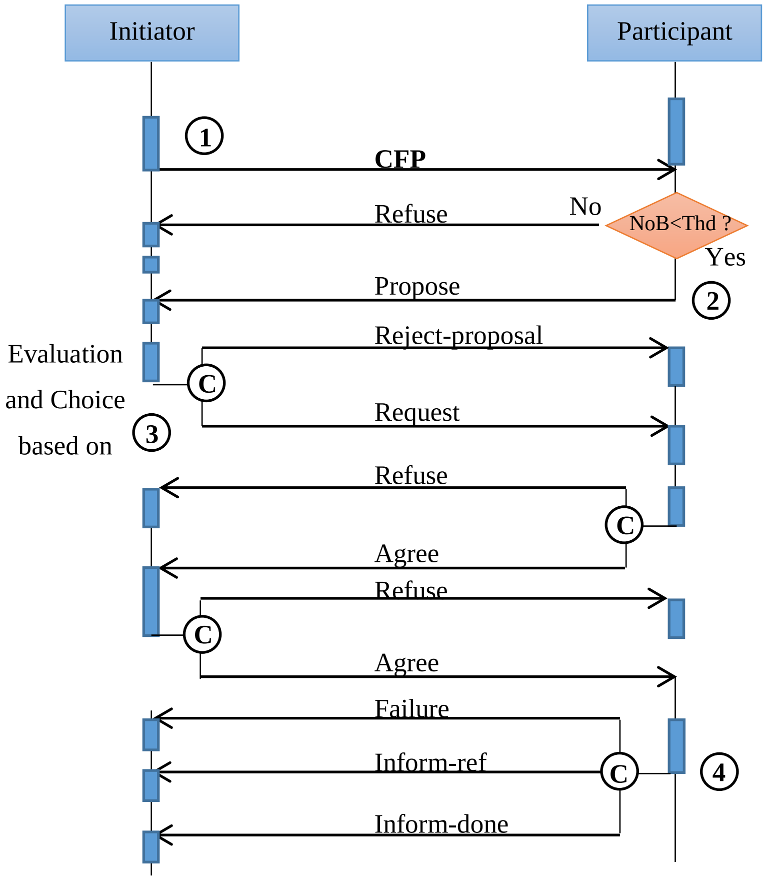

Figure 4.8 The process of extended CNP by (Yang et al., 2008)

To make Contract-net scalable in large grid resource allocation, the authors in (Mahajan and Gupta, 2008) have suggested several methods to: (1) handle increased number of resources (participants), a queue length based announcement strategy was proposed, (2) handle increased number of users (initiators), a decentralized broker system in which the brokers communicate with each other was proposed. The two strategies are combined to show that the combined strategy can handle both large number of users and resources. In their study, the authors have defined the initiators as users and the participants as resources. Furthermore, the

CFP deadlines were referred to as *the Job announcement expiration time*, and the term *Job_Deadline* has been used for expressing the time in which a resource must complete the job execution. The targeted problem is that a resource can receive multiple requests for jobs execution. The resource manager calculates the expected execution time for the received job taking into account the execution times of all the jobs waiting in its queue. The calculated time in which the job is expected to complete is compared to its deadline fixed by the user. If the job's deadline is respected then the resource manager makes a bid to the user. The problem is that the resource manager is in simultaneous interaction with other users, therefore the bid can be outdated if a new job has been accepted and added to the job's queue after the biding. The proposed queue length-based announcement strategy allows them to reduce the number of the resources to only those which can finish the job before the deadline according to their job queue length. They presented detailed simulation results to show that their approach significantly improves the scalability of the system up to 200 users.

To solve Contract Net protocol's scalability issue, the authors in (Deshpande et al., 2005) have suggested an *Instance-Based Learning* system (IBL) that would use previously stored instances to select a target agent. Its goal is to avoid the expensive bidding task. The strategy was applied in a virtual centralized healthcare network that uses the CNP for the distribution of services across hospitals. Experimental tests show that the system's performance was substantially increased with the use of the IBL. In regards to the number of tasks the system is more scalable.

The authors in (Aknine et al., 2004) have extended the Contract-Net protocol to: (1) allow parallel negotiations, (2) optimize negotiation time, (3) reduce the contractors' decommitment situations, and (4) detect failures and prevent negotiations with blocked agents. An important CNP limit that they described is when concurrent CFPs are received and answered sequentially by participants. Adding to that, the CFPs' deadlines with the participant's possible blocked and busy states will waste some contracts for them. The authors have introduced new communication primitives to the CNP, in order to create interactions' states which prevent the agent from facing faulty situations. The proposed CNP version (**Figure 4.9**) was evaluated using four experimental groups. Each one has two sub experiments with different numbers of initiators and participants. The used performance metric is the negotiation time which was indeed reduced using the proposed approach. Their work addresses the task allocation problem where participants propose their availability times to initiators. Their availability can be changed depending on the response of the initiators and the pre-scheduled tasks. In summary, the authors did not attempt to gain back some of the lost participant due to their blocked or busy situations during the task announcement, instead they just avoid to contact them to prevent failing situations.

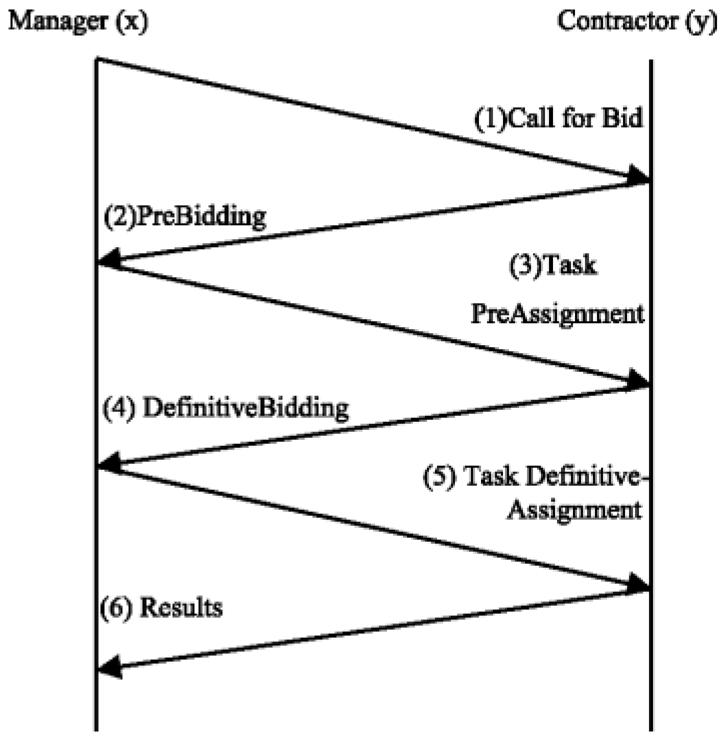

Figure 4.9 Phases of the extended Contract-Net Protocol by (Aknine et al., 2004)

In (Boukredera et al., 2012), the authors have proposed a formal model which can handle temporal aspects in a CNP interaction. In their study, CFPs' deadlines have been taken into consideration in order to prevent failures during interactions by using *Timed Colored Petri Nets* to model the CNP process. The authors have distinguished three possible situations for proposals' arrival: (1) all the proposals have been received before the deadline, (2) some of the proposals have been received after the deadline, and (3) all the proposals have been received after the deadline. According to these situations, initiators and participants can change their states by canceling and exiting the interaction to avoid blocking situations. This study is the only one that focused on the temporal aspect of the CNP along with (Aknine et al., 2004) and (Mahajan and Gupta, 2008), unfortunately, the authors did not intend to increase the number of proposals received before the deadlines but only to manage agents' states.

The presented researches have treated different problems related to the Contract-Net protocol. The most related ones to our study are those which highlighted the impact of simultaneous interactions and CFP's deadlines on the quality of the interaction. When a participant is busy or blocked for any given raison it cannot respond to all the received CFPs before their deadlines. Therefore, not only the participant's and the initiator's sociability has been decreased but also a non-optimal proposal may be selected and also announced tasks may be assigned to less qualified participants.

The presented researches have not investigated the sociability property of the agents during their interactions. In our knowledge, this work is the first investigating the problem and showing the capability of a participant agent to respond to several incoming Contract Net interaction requests (initiations) before their deadlines. Based on previous interactions, the participant will be able to prioritize the received initiations using machine learning algorithms to maximize the number of interactions which are successfully established.

A summary of this state of the art is presented in **Table 4.2**. We have proposed ten criteria to better understand and compare the different approaches.

1. Have the CFPs' deadlines been taken into consideration in the study by presenting the different situations that can arise when they are exceeded and how to prevent them?
2. Does the proposed enhancement require changes in the structure of the original Contract-Net protocol?
3. Can the participant participate to multiple interactions simultaneously in the study?
4. Does the participant receive a large number of CFPs in the study?
5. Does the proposed approach increase or decrease the number of participants for a given CFP in order to enhance the CNP?
6. Does the proposed approach increase or decrease the chance to obtain a better proposal?
7. Does the proposed approach have mechanisms to manage blocked or failure situations during the interaction?
8. Does the proposed approach allow the initiator to avoid repeating task announcement if it fails to reach an agreement?
9. Does the proposed approach reduce the CNP interaction time?
10. Does the proposed approach increase the agent's sociability by regaining the lost initiations due to CFP's deadlines or other causes?

Table 4.2 Summary of the studies related to the Contact-Net protocol enhancement in MASs

Study	Study with CFP Deadline	Change in the internal message sequences of the CNP	Simultaneous interactions by the participant	Large number of received CFPs	Increase/decrease number of participants	Increase/decrease the chance for better proposal	Blocked interaction management	Avoid repeating task announcement	Reduce CNP time	Increase replies before deadlines
(Hudaib et al., 2018)	No	Yes	No	No	Decreased	Increased	No	No	Yes	No
(Kaur et al., 2013)	No	Yes	No	No	Neither	No	No	Yes	Yes	No
(Yang et al., 2008)	No	Yes	No	No	Neither	Increased	Yes	No	No	No
(Schillo et al., 2001)	No	Yes	Yes	Yes	Neither	Increased	Yes	Yes	No	No
(Mahajan and Gupta, 2008)	Yes	Yes	Yes	Yes	Decreased	Increased	No	No	Unknown	Unclear
(Deshpande et al., 2005)	No	No	Yes	Yes	Avoided announcement	Unknown	No	No	Yes	No
(Aknine et al., 2004)	Yes	Yes	Yes	Yes	Neither	Unknown	Yes	No	Yes	No
(Boukredera et al., 2012)	Yes	No	Yes	Yes	Neither	Neither	Yes	No	No	No
The proposed approach	Yes	No	Yes	Yes	Neither	Increased	No	No	No	Yes

4 Conclusion

We presented in this section a brief description of the problems related to matchmaking and Contract-net protocol along with their state of the art.

In Matchmaking service, we have presented and discussed the most related approaches to our context. The closest ones are the studies of (Mengistu et al., 2008) and (Shahid and Hassan, 2008), they are both based on replicated and distributed DFs to balance the load when the number of search requests is increased. Their proposals still do not offer the best solutions for enhancing this service as discussed in their sections. Our proposed approach offers better solution without any change in the original source code.

For enhancing contract-net protocol several approaches were presented. Even that the most of them have highlighted the performance issue of this protocol, none of them have addressed the decreased agents' sociability or offered mechanisms to gain back interactions that were failed due to exceeded CFP's deadlines. The majority of them tends to change the internal structure of the CNP or to decrease the number of participants by excluding those with certain characteristics. Only three studies of (Aknine et al., 2004) (Mahajan and Gupta, 2008) and (Boukredera et al., 2012) have taken the CFP's deadlines into consideration in order to manage blocking and failing situations during interaction. Therefore, in our knowledge our proposal is the first of its kind that deals with failed initiations due to increased number of CFPs and exceeded deadlines.

Chapter 05

A Novel Approach for Improving the Matchmaker Service

Chapter 5 A Novel Approach for Improving the Matchmaker Service

1 Introduction

In this chapter, we will present the performance and scalability evaluation and enhancement of the JADE's DF. The problem statement was presented previously in the 4rd chapter. Therefore, we start in this one by presenting the proposed approach. Afterwards, we present an evaluation of the performance and scalability of the JADE's DF. Finally, the proposed approach will be validated, discussed and compared with the JADE's DF.

In the proposed approach, an aspect was used to play the roles of a replicator and a distributor of requests to several new Yellow Pages. The aspect module creates several DF copies in which it manages service registration, deregistration and modification requests in order to ensure data consistency. All the DFs take turn to respond to the search requests, which will be distributed by the aspect to ensure a load balancing over all the Yellow Pages of the system.

Aspect-oriented programming is used to simplify the implementation and reduce the maintenance effort to insure modularity by separating the improvements code from the original classes and to offer a good reusability of the proposed maintenance approach. Three JADE applications have been used to evaluate the proposed approach on five different execution environments. Results, which are very promising, show a significant increase in the performance and scalability of the DF service especially on Linux operating system.

2 The Proposed Approach

The proposed perfective maintenance of the Yellow Pages service is a load balancing approach which uses aspect-oriented programming to distribute the load on different DF copies. For clarity reasons, a concrete example in JADE platform is used to present the proposed approach which consists of three main activities managed by AspectJ (**Figure 5.1**). This latter was used for the following reasons:

1. It is based on Java language and easy to integrate within the project.
2. The new functionalities are separated from the original classes and located in one aspect (better separation of concerns).
3. All kinds of requests (register, deregister, search, etc.) can be intercepted, modified and forwarded during runtime.
4. Less effort and better reusability of the new functionalities without redevelopment or original code refactoring.
5. Easy documentation update.

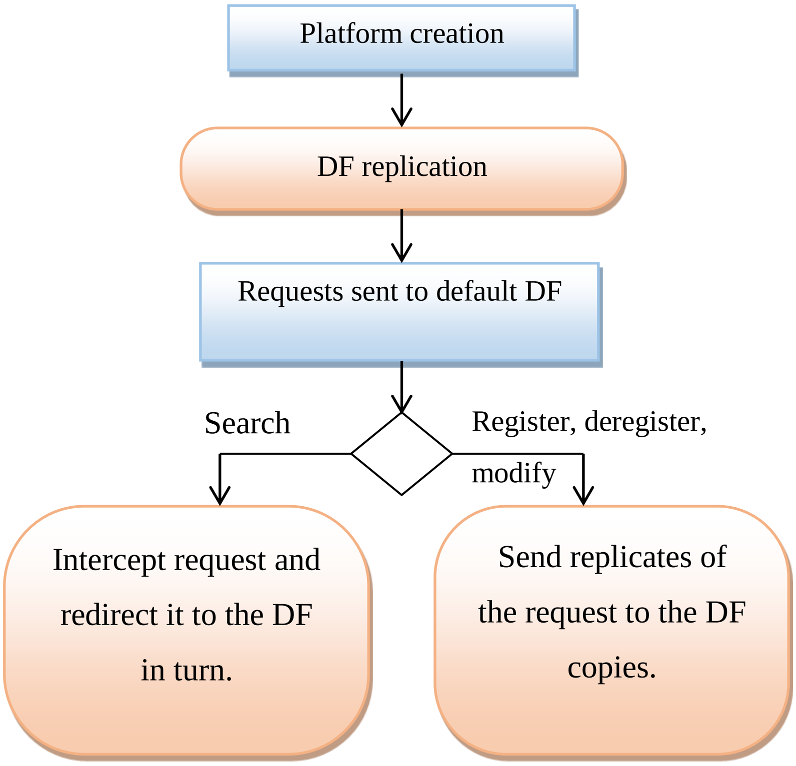

Figure 5.1 Global view of the proposed approach (Zerrougui et al., 2017)

When the agent platform is created, a number of DF copies are replicated by the aspect. Each time the default DF receives a request for service registration, deregistration or modification, the aspect send replicates of the request to the DF copies in order to ensure data consistency. The search request will be intercepted and redirected to the DF in turn (including JADE's DF).

2.1 Yellow Pages replication

The replication process is managed by the *DFenhancer* aspect (**Figure 5.2**) and starts right after containers creation and before any other agent. The DF copies must be already active to be able to have the same content of the default DF. Otherwise, all its registered services are copied to the new DF copies.

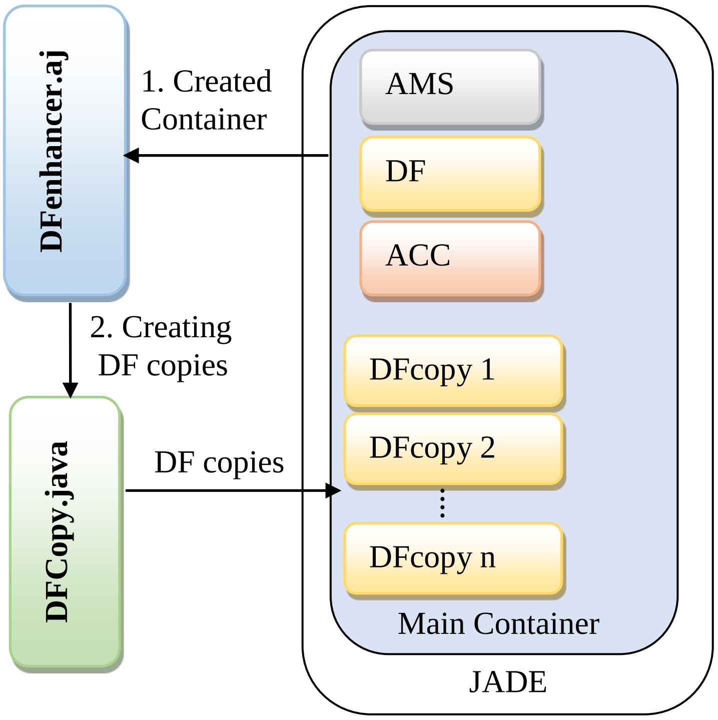

Figure 5.2 the Yellow Pages replication in the proposed approach (Zerrougui et al., 2017)

The DF copies are instances of the *DFCopy* class which extends the *jade.domain.df* class (**Snippet 5.1**). A DF copy agent can acquire the same behaviors of the default DF such as *register*, *deregister*, *modify* and *search* by calling the *super.setup()* method. The number of the instances depends on the execution environment and the complexity of the system. Therefore, the optimum number of DF copies may be obtained using some empirical evaluations.

Snippet 5.1 the *DFcopy* class

```
public class DFCopy extends jade.domain.df {
  public void setup() {

/* A method call to acquire all the behaviors of the JADE's DF */
  super.setup();

 /* An option to show the GUI of a DFcopy agent*/
  this.showGui();
 }
```

2.1.1 ***Initializations***

Two roles are played by the *DFenhancer* aspect. The first role is to replicate the JADE's DF along with three requests' types (service registration, modification and deregistration). The second role is to distribute the search requests on all the DFs. The aspect is initialized by the following variables (**Snippet 5.2**):

1. A fixed number of DF copies (*dfCopies_number*) necessary for load balancing.
2. A list of the DFs' AIDs with a counter of its size. This counter is used instead of asking for the list length in every request. The *dfCopies_number* variable can be used as well, but we prefer the use of two variables for further possible maintenance.
3. The intercepted search requests count which will be used to identify the DF in turn,
4. A Boolean used to find the default DF's AID in order to be added to the DFs' list.

Snippet 5.2 the *DFenhancer* aspect and the used variables

```
/* Section to import required packages*/
public aspect DFenhancer {
      int dfCopies_number=4; /* DF copies number in the system*/
      LinkedList<AID> dfAgents_list = new LinkedList<AID>();/*list of all DF Agents*/
      int listSize=0; /* Used to optimize code instead of calling dfAgents_list.length()*/
      int search_count=-1; /* Number of search requests*/
      boolean defaultDF_Found=false; /* To find the default df*/
```

2.1.2 *Creating DF Copies*

The focus in this thesis is to apply the proposed approach on local hosts only and not over distributed hosts through local network or internet. The reason is to avoid any influence of the network on the obtained results. The DF copies are created in the same main container even that they can be created on different containers in the same host as well.

Snippet 5.3 presents the point cut and the advice responsible for the replication of the DF agent right after the main container creation.

Snippet 5.3 the *pointcut* and the *advice* used for creating DF copies

```
/* Intercept the creation of the main container*/
pointcut makeDFcopies ():  call(* *..createMainContainer(..)); /* We can add the simple
containers here.*/
/* Start the DFcopies in the intercepted Container.*/
  AgentContainer around():makeDFcopies () {
/* Allow container creation first*/
  AgentContainer agentContainer = proceed();
/* Create and start the DFcopies agents*/
  AgentController dfCopy = null;
  try {
    for (int i = 0; i < dfCopies_number; i++) {
         dfCopy = agentContainer.createNewAgent(
         "DFCopy" + Integer.toString(i), "DFCopy", null);
         dfCopy.start();
     }
  } catch (StaleProxyException ex) {     }
/* Return the intercepted container*/
return agentContainer;
}
```

2.1.3 *Adding all DFs AID to a List*

To obtain the AIDs of the available DFs, the method *getAID()* can be only called from the intercepted join point (**Snippet 5.4**). The AID of the JADE's DF is extracted using the *getDefaultDF()* method. These AIDs are added to a list which will be used later for load distribution.

Snippet 5.4 Saving the AIDs of the DF agents to a list

```
/* Intercept The created DFcopy previously to add their AID  to the DFagents List*/
pointcut intercept_dfCopy():  execution(* DFCopy.setup(..));
Object around(): intercept_dfCopy() {
   if(!dfAgents_list.contains(((Agent)thisJoinPoint.getThis()).getAID())) {
       dfAgents_list.addFirst(((Agent)thisJoinPoint.getThis()).getAID());
       listSize++;
    } /* Get Default DF  - executed only once*/
   if (!defaultDF_Found) {
      dfAgents_list.addLast(((Agent)thisJoinPoint.getThis()).getDefaultDF());
      defaultDF_Found = true;
      listSize++;
    }
return proceed();
}
```

2.2 Service Registration, Deregistration and Modification

In order to ensure data consistency between the DF copies, the *DFenhancer* aspect intercepts service registration, deregistration and modification requests. These requests are replicated and passed to all the available DFs (**Figure 5.3**).

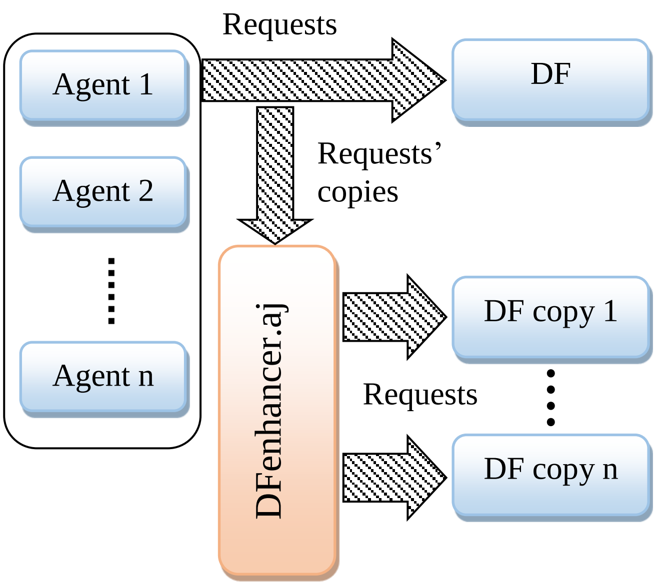

Figure 5.3 Replication of service registration, deregistration and modification requests (Zerrougui et al., 2017)

2.2.1 *Service Registration*

In **Snippet 5.5**, the *intercept_Register()* point cut declares the join point of service registration request sent by the *DFService.register()* method. When the join point is intercepted the around advice executes its code. The *DFAgentDescirption* which contains the service description is extracted from the arguments then replicated and forwarded to all the DF copies. By executing *proceed(),* the advice allows the *DFService.register()* to continue its execution to send the request to the default DF.

In this case, only two possible join points can be intercepted with different arguments (method overloading):

- register(Agent a, AID dfName, DFAgentDescription dfd),
- register(Agent a, DFAgentDescription dfd).

Snippet 5.5 Intercept and forward the *register* requests to all the DF agents

```
pointcut intercept_Register(): call(* DFService.register(..)) &&  !within(DFenhancer);
Object around(): intercept_Register() {
   try {
      for (int i = 0; i < listSize - 1; i++)
/* We add this parameter dfAgents_list.get(i) to forward the request to all the DF copies we
need the first and the last arguments ( Agent,DFAgentDescription)*/
DFService.register( (Agent) thisJoinPoint.getArgs()[0], dfAgents_list.get(i),
(DFAgentDescription) thisJoinPoint.getArgs()[thisJoinPoint.getArgs().length - 1]);
} catch (FIPAException e) {}
/* Register in the Default DF too.*/
proceed();
return null;
}
```

2.2.2 *Service Deregistration*

The implementation of the deregistration process is similar to the previous one. **Snippet 5.6** presents a point cut declaration for the *DFService.deregister()* method. Four possible calls can be intercepted with different arguments:

- deregister(Agent a),
- deregister(Agent a, AID dfName),
- deregister(Agent a, AID dfName, DFAgentDescription dfd),

- deregister(Agent a, DFAgentDescription dfd).

The arguments size can be extracted from the intercepted join point to identify which method has been intercepted.

Snippet 5.6 Intercept and forward the *deregister* requests to all the DF agents

```
pointcut intercept_Deregister():  call(* DFService.deregister(..))  && !within(DFenhancer);
Object around(): intercept_Deregister() {
      try {
          for (int i = 0; i < listSize - 1; i++)
/* We add this parameter dfagents.get(i) to forward the request to all the DFcopies*/
           DFService.deregister((Agent)thisJoinPoint.getArgs()[0], dfAgents_list.get(i));
      } catch (FIPAException e) {}
/* Deregister from the Default DF also.*/
proceed();
return null;
}
```

2.2.3 *Service Modification*

Service modification presented in **Snippet 5.7** can be intercepted using the *intercept_Modify()* point cut. The join points are the calls of the *DFService.modify()* which can be found with two different arguments as follows:

- modify(Agent a, AID dfName, DFAgentDescription dfd),
- modify(Agent a, DFAgentDescription dfd).

Snippet 5.7 Intercept and forward the *modify* requests to all the DF agents

```
pointcut intercept_Modify():call(* DFService.modify(..)) && !within(DFenhancer);
Object around(): intercept_Modify() {
   try {
       for (int i = 0; i < listSize - 1; i++)
/* We add this parameter DFagents.get(i) to forward the request to all the DFcopies*/
              DFService.modify((Agent)thisJoinPoint.getArgs()[0],dfAgents_list.get(i),(D
FAgentDescription)   thisJoinPoint.getArgs()[thisJoinPoint.getArgs().length - 1]);
       }
  catch (FIPAException e) {  }
/* Modify in the Default DF also.*/
proceed();
return null;    }
```

2.3 Service Search

The new distributed DFs are ready to receive search requests from different agents. The *DFenhancer* aspect reduces the load of the default DF by distributing the search requests on all the other DFs (**Figure 5.4**).

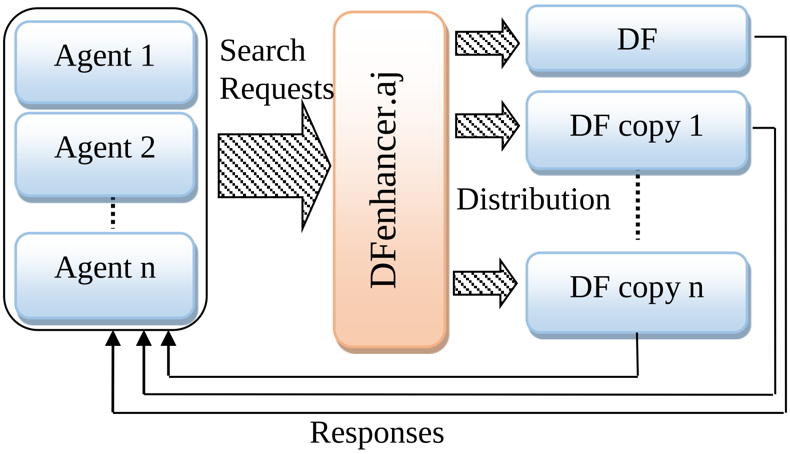

Figure 5.4 Service search by distributing requests on DF agents (Zerrougui et al., 2017)

As shown in **Figure 5.5**, all the available DFs take turns to respond to search requests. This load balancing process is managed by the *DFenhancer* aspect, which intercepts the search request and redirects it to the DF in turn ().

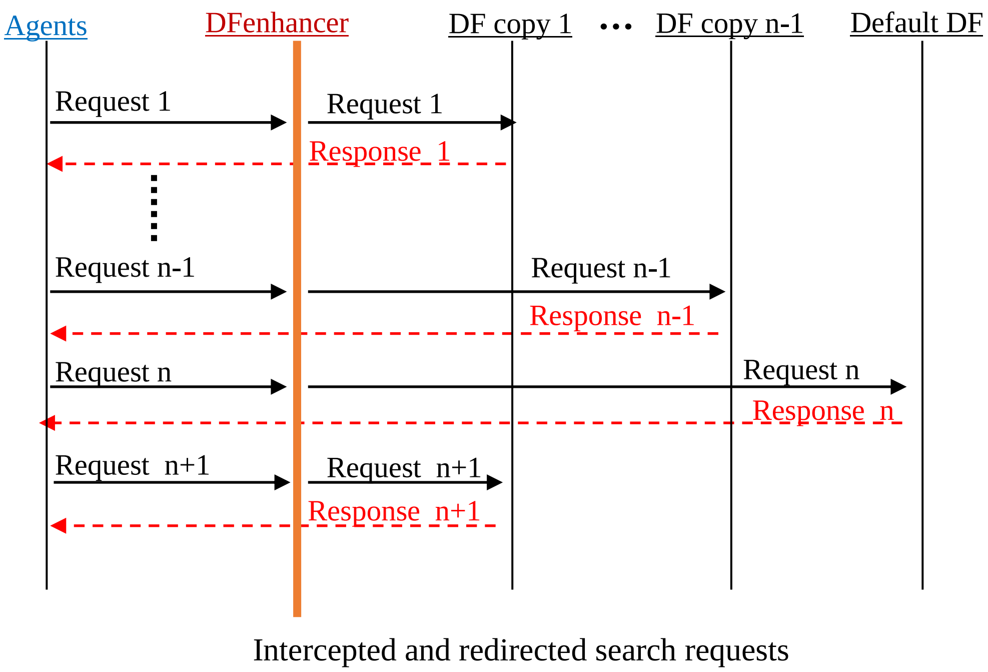

Figure 5.5 Distribution of requests using DF turns (Zerrougui et al., 2017)

The code responsible for search requests distribution is presented in **Snippet 5.8**. The *intercept_Search()* point cut contains declaration of *DFService.search(..)* method calls. When these join points are intercepted the around advice is executed. The *DFService.search(..)* method can be used with different arguments as follows:

- search(Agent a, AID dfName, DFAgentDescription dfd),
- search(Agent a, AID dfName, DFAgentDescription dfd, SearchConstraints constraints),
- search(Agent a, DFAgentDescription dfd),
- search(Agent a, DFAgentDescription dfd, SearchConstraints constraints).

In general, the default DF is always addressed for search requests, otherwise the target DF's AID must be added as argument. It is up to the application's engineer to decide whether this type of search requests can be altered or not depending on its preferences.

The search request can also be parameterized with search constraints. Therefore, in both cases, the same request must be redirected.

A circular list which holds the DFs' AIDs is used to manage their turns. It is implemented using a modulo function which returns a DFs' position *(index)* in the circular DF list. The resulted DF will reply for the current intercepted request as presented in *(Eq 5.1)*.

$$index = search\ request\ count\ \%\ number\ of\ DFs\ in\ the\ system \qquad (Eq\ 5.1)$$

Snippet 5.8 Intercept and redirect the search request to the DF agent in turn

```
pointcut intercept_Search():  call(* DFService.search(..))  && !within(DFenhancer);
DFAgentDescription[] around(): intercept_Search() {
/* If the second argument is a DF Agent ID,  the request will not be changed.*/
  if(thisJoinPoint.getArgs()[1].getClass().getSimpleName().equals("AID") )  {
      try {

/* To distribute the requests to the current DF on the list we increase the search_count then we divide  it by
listsize so that the reminder of the division is the position of DF in the list - ( this emulate a circular list where
the next element of  the last one is the first element)*/

        switch (thisJoinPoint.getArgs().length)
           { /* Depending on the search arguments, if there is a search constraints or NOT*/
            case 2:return DFService.search((Agent) thisJoinPoint.getArgs()[0],dfAgents_list.get(++search_count
% listSize),(DFAgentDescription) thisJoinPoint.getArgs()[1]);
            case 3:return DFService.search((Agent) thisJoinPoint.getArgs()[0],dfAgents_list.get(++search_count
% listSize),(DFAgentDescription) thisJoinPoint.getArgs()[1],(SearchConstraints) thisJoinPoint.getArgs()[2]);
           default:  break;
                }
        }
```

```
    catch (FIPAException e) {}
return null;
}
}
```

3 Validation and Discussion

The use of aspect-oriented paradigm has simplified the implementation and the deployment of the proposed approach. The performance and scalability enhancement concerns are separated from the original MAS classes. The MAS project must be instrumented with the following modules (**Figure 5.6**) without any code reimplementation:

1. Import *DFenhancer.aj* and *DFCopy.java* modules to the default package.
2. Import the AJDT (AspectJ) library to the BuildPath.

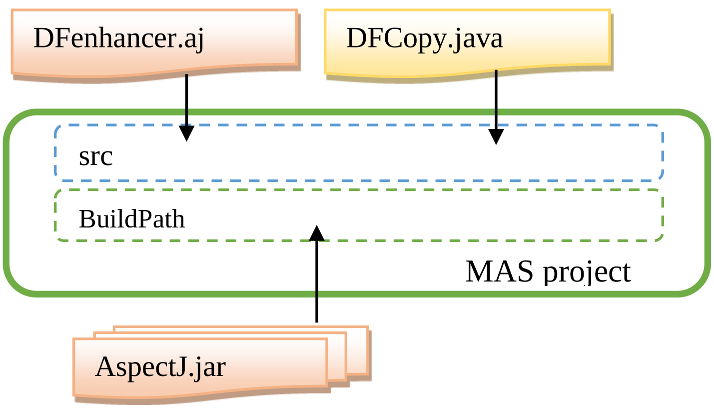

Figure 5.6 MAS project instrumentation (Zerrougui et al., 2017)

In order to validate the proposed approach, we have conducted a deep search on unified benchmarks related to the JADE platform. Unfortunately, none of them has been found in the MAS community except for few academic projects and examples. In our opinion, this is not a big issue when it comes to our research context. The Matchmaking service is a design pattern as explained before and its implementation is quite similar in all the MASs of the same agent platform. The difference may be in the complexity of the MAS and the load applied on the DF agent.

We have selected three case studies to validate the proposed approach. The first one is the *bookTrading* example provided within JADE package (Giovanni, 2003). In this example, book sellers register their services into the DF agent and book buyers search for books selling services. The second MAS named *Translators* is a simple JADE application that we developed. It consists in four agents' types: (1) a requester agent which seeks translations for English words, (2) the remaining three agents' types provide translation services where each one is specialized in a specific language. The third case study called *im2008-jade* ("im2008-jade," 2008) is a more complex MAS that we previously used in (Zerrougui et al., 2014). It's about an information system for real-time resources sharing which their availability is manifested with very short notice and with a very limited useful lifetime. Such resources can be linked to situations such as:

1. Tickets' resale (events, train, etc.) which are already purchased and no longer needed by the first buyer.
2. Availability of places on private transport.
3. Products with short-term discounts.

The first two case studies which are simple MASs that we believe are good examples to evaluate the performance of the DF agent without the interference of other functionalities. In the other hand, the *im2008-jade* MAS is more complex, which can demonstrate how the proposed approach will perform when the agents have other tasks to perform. We believe that in order to obtain correct performance results of the DF service, it is preferable to avoid complex systems to ensure that the issue is raised from the service itself and not propagated from other concerns. Therefore, for complex systems we must take into consideration that many situations may be faced during evaluation:

1. CPU resources may be shared with other functionalities which may take too much computing time. This leads to slow DF due to resource insufficiency and not to its design.
2. For a bottleneck to occur, requests must be received in a short time. This pressure can go off if agents wait too much before they send search requests.
3. The DF performance is related to the numbers of search requests and not to the number of existing agents. An agent can send more than one request using cyclic behaviors. In this case, we may control the number of requests, but unfortunately, we cannot control the requests' frequencies (interval between requests).

We have randomly fixed the number of DFs to four copies beside the original DF agent.

3.1 Execution environments

In **Table 5.1**, the technical characteristics of the used execution environments for this evaluation are presented. Five personal computers (Desktops and laptops) were used with different operating systems (Windows and Linux) and different CPU and RAM capacities.

Table 5.1 the used execution environments for the Yellow Pages service evaluation (Zerrougui et al., 2017)

	Desktop Linux	**HP Windows**	**Acer Windows**	**HP Linux**	**Acer Linux**
Model	Desktop - GigaByte H55M-S2 motherboard	HP dv6	Acer 5733	HP dv6	Acer 5733
CPU	i3-530 2.93Ghz	i5-2410M 2.30 GHz	i3-M380 2,53 GHz	i5-2410M 2.30 GHz	i3-M380 2,53 GHz
RAM	6 GB	8 GB	4 GB	8 GB	4 GB
Operating system	Linux (Ubunto 14)	Windows 7	Windows 7	Linux (Ubunto 14)	Linux (Ubunto 14)

The used tools are:

1. Eclipse Luna (4.4.1) with JDK8 Update 91
2. JADE v4.1.1 (since v4.2 a DF search timeout was set to 30 seconds, current version is v4.4)
3. AspectJ (AJDT 2.2.4 for Eclipse 4.4)

3.2 Evaluation of the DF Performance

In large-scale multi-agent systems, the DF could be highly loaded with search requests received in high frequencies, which makes it difficult for the DF agent to reply in a reasonable time. We are using only the two previously mentioned case studies *bookTrading* and *Translators* to evaluate the performance limits of the JADE's DF under several search requests frequencies.

The *Round-Trip Time* (RTT) is used as a performance metric that presents the search request's response time (Shahid and Hassan, 2008) (Mengistu et al., 2008) (Nguyen et al., 2011b) (Cortese et al., 2002). In order to obtain this measurement, we have instrumented the requester agent with two code lines (1 & 2) as shown in **Snippet 5.9**.

Snippet 5.9 Calculating the Response Time of a Search Request

```
long startTime = System.nanoTime();/* System time before request in nanoseconds (1)*/
DFAgentDescription[] result = DFService.search(myAgent,template);
System.out.println((System.nanoTime() - startTime)/1000000 ); /* RTT in milliseconds (2)*/
```

For the *bookTrading* example, sellers must be created first so they can register their services in the DF before any search requests. Since that the default search results are limited to 100 services in JADE's DF, 100 book sellers were created (**Snippet 5.10**). Afterwards, the book buyers were created starting from 2000 buyers in the first iteration to 10000 buyers in the last one by adding 2000 buyers after each iteration. It is important to terminate the active agents after each iteration before starting a new one.

In the *Translators* case study, we have repeated the same previous iterations. In which 300 services were registered in the DF (100 services for each one of the three translators).

It is important to understand that the DF service performance is related to the number of received search requests and not to the number of agents. We could create less number of agents with the same count of search requests. Unfortunately, in this method we cannot control the evaluation conditions such as search frequencies which must be reapplied for the proposed approach in order to perform a correct evaluation and comparison.

Snippet 5.10 Starting buyer agents in the *bookTrading* example

```
/*........ Here is the code for starting the JADE platform and 100 seller agent
........*/
int buyers_number=0;
int iterations=5;
    for(int iter=0; iter<iterations; iter++)
      {
        buyers_number = buyers_number + 2000;
        for (int i =0; i<buyers_number; i++)
           {
             Buyer =
ac.createNewAgent("BookBuyer"+Integer.toString(i),"examples.bookTrading.BookBuyerAg
ent",argss);                        /*creating buyer agents*/
             buyer.start();          /*starting buyer agents*/

/* Delay time in milliseconds before starting a new buyer agent, we remove the code in the no
interval case */
```

```
            try {
        Thread.sleep(1); /* No interval, 1ms, 2ms, 5ms, 6ms and 7ms were used */
           catch (InterruptedException e) { }
            }  /*  end for (i)*/
/* ...code to obtain the results and terminate the agents*/
     }  /*end for (iter)*/
```

In each execution environment, the same evaluations were performed with the same search requests frequencies. Since the results of these evaluations were similar in all the used execution environments, only the results of the desktop Linux machine using the *bookTrading* example will be presented.

The used search requests frequencies are: no interval, 1ms, 2ms, 5ms, 6ms, and 7ms. The last evaluation with 7ms interval was the state where the JADE's DF starts to perform smoothly with no performance issues. The RTTs of the search requests in the *bookTrading* case study are presented in **Table 5.2** with the median values and **Table 5.3** with the mean values (*Translators* has approximately similar results). In other words, in the first iteration with 2000 agents for example, we obtain 2000 RTTs that we measure their median and mean value.

Table 5.2 the medians of RTTs of JADE's DF performance in the Desktop Linux machine according to requests frequency (bookTrading) (Zerrougui et al., 2017)

	The median of RTTs in milliseconds (ms)					
Agents (Iteration)	**No Interval**	**1ms Interval**	**2ms Interval**	**5ms Interval**	**6ms Interval**	**7ms Interval**
2000	9173	6092	6202	2374	1360	142
4000	15297	12686	11154	3237	958	10
6000	21674	19200	16845	4747	624	9
8000	30175	26793	23989	7155	863	9
10000	37797	34593	31645	8343	1028	9

Table 5.3 the means of RTTs of JADE's DF performance in the desktop Linux machine according to requests frequency (bookTrading) (Zerrougui et al., 2017)

	The means of RTTs (ms)					
Agents	**No Interval**	**1ms Interval**	**2ms Interval**	**5ms Interval**	**6ms Interval**	**7ms Interval**
2000	8708	6046	5821	2433	1626	183
4000	14903	12343	10835	3407	959	17
6000	22060	19203	16500	5010	646	10
8000	30457	26564	23319	6939	899	10
10000	37956	34205	30476	8969	1078	10

In statistics (laerd, 2017), if the means and medians of a given measure are close to each other then it is more suitable to use the mean values. Otherwise, if they are different and the data is skewed then the use of the median values is more appropriate. Therefore, only the medians of the RTTs are presented in **Figure 5.7**. Given that the RTT necessary for the DF to return 100 services for only one search request varies about 10 to 25 milliseconds (could be less or more).

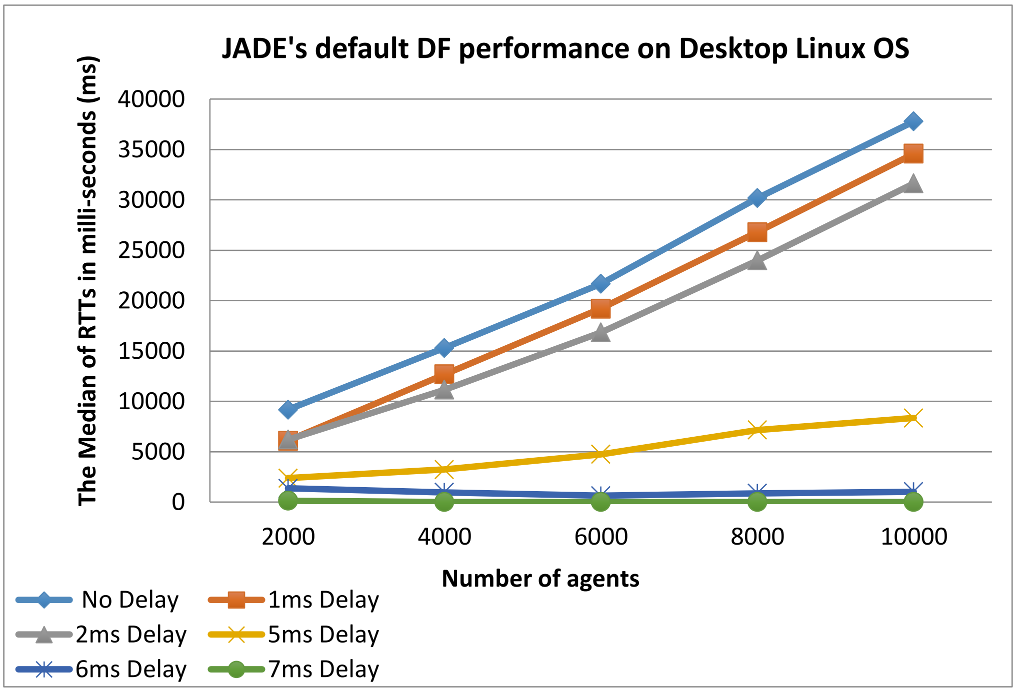

Figure 5.7 the performance of Jade's DF in a Desktop Linux machine according to requests frequency (bookTrading) (Zerrougui et al., 2017)

Six curves for the response times of search requests are presented in **Figure 5.7** where each one depends on the used frequency. For the higher frequencies' curves (no interval, 1ms, 2ms, and 5ms), we observed that the medians of RTTs are increased with the number of agents. With the 6ms frequency, the RTTs were not optimal but nicely stable during all iterations. The best performance was recorded with the 7ms interval curve, which shows that the DF agents perform smoothly at high number of agents.

We have noticed that the RTT of the 1st iteration with the 7ms evaluation is higher than the RTTs of the rest of iterations even that the number of agents was less (**Table 5.2** and **Table 5.3**).

Basically, these high RTTs values were present at the beginning of the 1st iteration of all the evaluations. The first requests had high RTTs which they supposed to be responded to in the optimal time since that the JADE's DF is not loaded yet. This observation was hidden from us during the high frequencies where the performance was decreased according to the number of requests because the RTTs keep increasing. But, for the 7ms evaluation, we observed that the RTTs drop down and stabilized after a while, which made us believe that something is influencing the response time when the evaluations start.

Therefore, further investigation was performed using JProfiler (Cho, 2006). The profiler detected that a *Garbage collector* activity was executed during the evaluations which took a considerable amount of CPU resources for a while.

The JADE support has provided other explanations to this problem in their mailing lists (JADE Support, 2011). They explained that the performance degradation at the beginning of system execution may depend on several factors such as: *"(a) ClassLoading: At the first round, the agent classes as well as the DB driver classes must be loaded but at next rounds they are already there, (b) DB connections: At the first round, all connections with the DB must be opened. But, at the next round (depending on how you access the DB) they may be already there, and (c) Just-in-time compilation."*

Since JADE v4.2, the DF search response time was limited to 30 seconds. If the time limit is exceeded during a search request then JADE will respond with a timeout error. As presented in **Table 5.4**, the maximum RTTs of the no interval requests frequencies shows that only the first two iterations could pass the evaluation in case recent JADE versions were used.

Table 5.4 the JADE's DF performance in the Desktop Linux machine according to requests frequency with No Interval (full frequency) (bookTrading) (Zerrougui et al., 2017)

Agents	Mean (ms)	Median (ms)	Min (ms)	Max (ms)
2000	8708	9173	57	15194
4000	14903	15297	17	28271
6000	22060	21674	16	43684
8000	30457	30175	13	60139
10000	37956	37797	9	74133

The proposed approach will be validated on the previously mentioned case studies in order to enhance the performance and scalability of the matchmaking service especially in high requests frequencies.

3.3 Results and discussion

3.3.1 *The First Case Study*

The first evaluation was performed using the *bookTrading* case study with four extra DF copies and no interval in requests frequency. **Table 5.5** presents the medians of the obtained RTTs. We must point that the number of services returned by the DF copies in every request was 100 services, which means a success rate (Cao et al., 2001) of 100%.

The results in **Table 5.5** show a great reduction in RTTs which means better performance and scalability of the proposed approach than with the JADE's DF especially under the Linux OS where the percentage decrease in RTTs was stable at more than 99.5%. The percentage decrease was calculated in every iteration by *(Eq 5.2)*.

$$Percentage\ decrease = \left[\left(\frac{\text{Median RTT of the default DF} - \text{Median RTT of the proposed approach}}{\text{Median RTT of the default DF}} \right) \right] \times 100 \qquad \textit{(Eq 5.2)}$$

In Windows execution environments, about 25% to 45% in percentage decrease of RTTs was obtained, where it was maintained at about 42% to 44 % with more than 4000 active agents. This is different than in Linux execution environments results where the percentage decrease in RTTs was stable at more than 99.5% in all iterations.

Table 5.5 the performance of the proposed approach with new 04 DF copies using the bookTrading application and according to requests frequencies with no interval (Zerrougui et al., 2017)

		Number of agents				
		2000	**4000**	**6000**	**8000**	**10000**
HP Windows	JADE's DF	7453	12903	19023	25620	32322
	Proposed approach	5580	7421	10731	14122	18019
	% Decrease	**25,13%**	**42,49%**	**43,59%**	**44,88%**	**44,25%**
Acer Windows	JADE's DF	13824	18815	27339	38515	48980
	Proposed approach	9647	10764	15207	22550	28399
	% Decrease	**30,22%**	**42,79%**	**44,38%**	**41,45%**	**42,02%**
Desktop Linux	JADE's DF	9173	15297	21674	30175	37797
	Proposed approach	52	35	55	50	60
	% Decrease	**99,43%**	**99,77%**	**99,75%**	**99,83%**	**99,84%**
HP Linux	JADE's DF	7283	13629	20054	26796	33794
	Proposed approach	32	27	49	40	51
	% Decrease	**99,56%**	**99,80%**	**99,76%**	**99,85%**	**99,85%**
Acer Linux	JADE's DF	10093	16490	26682	36447	47179
	Proposed approach	972	55	139	140	87
	% Decrease	**90,37%**	**99,67%**	**99,48%**	**99,62%**	**99,82%**

The results of the performance evaluation using *bookTrading* case study, with requests frequencies of 1ms interval presented in **Table 5.6**, show a significant and stable percentage decrease in RTTs of more than 99.5% even with the Windows execution environments. The medians of RTTs vary from 9ms to 15ms, while the RTTs were even more optimized in Linux environments than their previous no interval evaluations. Thus, only the no interval evaluations under Windows environments in which the proposed approach did not result as expected theoretically and practically as it did under Linux environments. Therefore, the operating system has definitely an impact on the proposed approach. Unfortunately, no other explanations are available yet, unless further investigations are carried out.

Table 5.6 the performance of the proposed approach with new 04 DF copies using the bookTrading application and according to requests frequencies with 1ms interval (Zerrougui et al., 2017)

		Number of agents				
		2000	**4000**	**6000**	**8000**	**10000**
HP Windows	JADE's DF	4005	9946	16251	22370	27672
	Proposed approach	9	10	10	10	15
	% Decrease	**99,78%**	**99,90%**	**99,94%**	**99,96%**	**99,95%**
Acer Windows	JADE's DF	3643	11874	22597	31559	42141
	Proposed approach	13	12	14	14	13
	% Decrease	**99,64%**	**99,90%**	**99,94%**	**99,96%**	**99,97%**
Desktop Linux	JADE's DF	6046	12343	19203	26564	34205
	Proposed approach	14	12	12	12	12
	% Decrease	**99,77%**	**99,90%**	**99,94%**	**99,95%**	**99,96%**
HP Linux	JADE's DF	5550	10926	17073	23410	31001
	Proposed approach	12	12	11	11	11
	% Decrease	**99,78%**	**99,89%**	**99,94%**	**99,95%**	**99,96%**
Acer Linux	JADE's DF	8840	16065	25625	35247	46449
	Proposed approach	21	18	17	17	17
	% Decrease	**99,76%**	**99,89%**	**99,93%**	**99,95%**	**99,96%**

For visual representation of the obtained results, **Figure 5.8** and **Figure 5.9** are presented. They show the evaluation of the proposed approach on the *bookTrading* example using 1ms and no interval in requests frequencies. For every used execution environment, the scalability and the performance were optimal with more than 99.5% decrease in RTTs, except for the Windows execution environments with the no interval evaluation where the RTTs where only decreased by up to 45%.

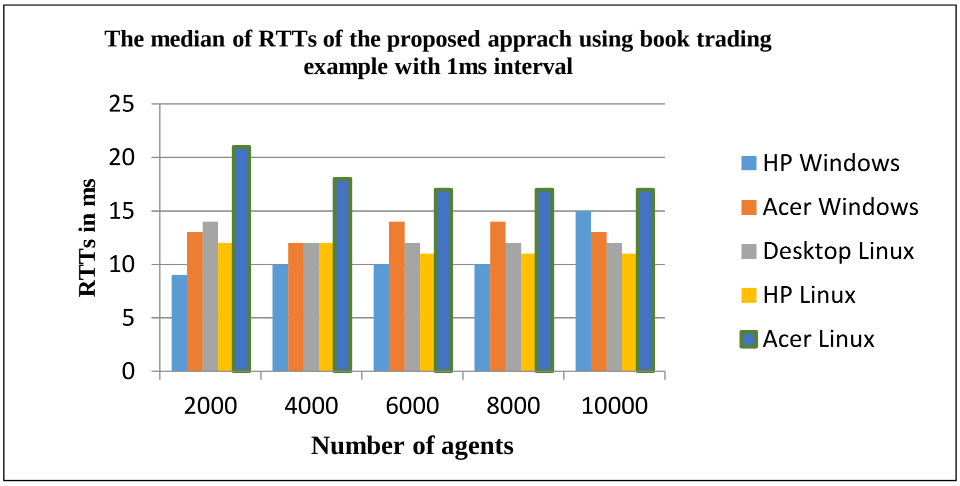

Figure 5.8 the proposed approach evaluation on bookTrading with 1ms interval (Zerrougui et al., 2017)

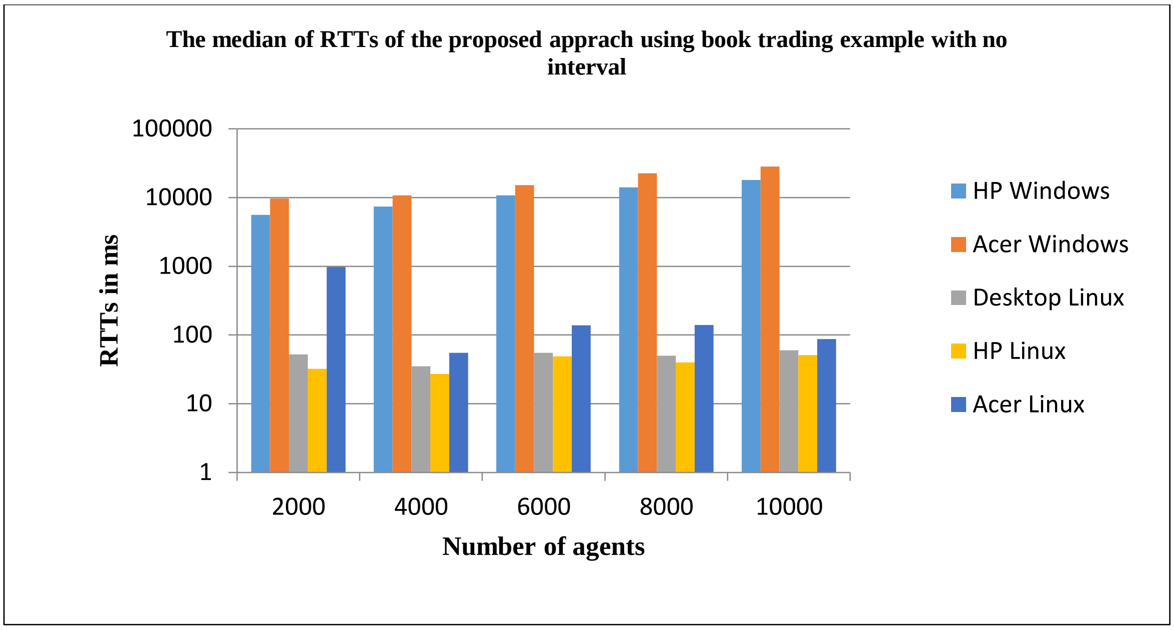

Figure 5.9 the proposed approach evaluation on bookTrading with no interval (Zerrougui et al., 2017)

A supplement evaluation was carried to investigate the performance of the proposed approach when the number of agents is increased to 50000 agents. Using the *bookTrading* case study, the results were similar to those with 10000 agents with significant performance and scalability enhancements.

3.3.2 *The Second Case Study*

The *Translators* case study that we developed to evaluate the proposed approach is a simple MAS implemented in JADE that consists on requesters which search for word translators. The translators are classified into three types; each type offers word translation service to one of these languages Dutch, French and Arabic. Therefore, the number of the registered services is 300 services (100 services for each language).

In **Table 5.7**, the evaluation results using no interval in requests frequency were presented. The obtained RTTs were approximately similar to those of the *bookTrading* case study, in which the best improvements were in the Linux execution environments.

Table 5.7 the performance of the proposed approach with new 04 DF copies in using the *Translators* application and according to requests frequencies with no interval (Zerrougui et al., 2017)

		Number of agents				
		2000	**4000**	**6000**	**8000**	**10000**
HP Windows	JADE's DF	9186	12212	17839	24161	30796
	Proposed approach	6229	6255	10169	12968	16749
	% Decrease	**32,19%**	**48,78%**	**43,00%**	**46,33%**	**45,61%**
Acer Windows	JADE's DF	12426	17461	26945	35608	47041
	Proposed approach	9654	9629	14512	20285	26857
	% Decrease	**22,31%**	**44,85%**	**46,14%**	**43,03%**	**42,91%**
Desktop Linux	JADE's DF	9246	13476	20411	28910	37425
	Proposed approach	33	46	41	49	50
	% Decrease	**99,64%**	**99,66%**	**99,80%**	**99,83%**	**99,87%**
HP Linux	JADE's DF	7838	11458	18193	24892	31331
	Proposed approach	75	31	50	83	78
	% Decrease	**99,04%**	**99,73%**	**99,73%**	**99,67%**	**99,75%**
Acer Linux	JADE's DF	8867	16625	25218	35598	45655
	Proposed approach	475	52	61	66	72
	% Decrease	**94,64%**	**99,69%**	**99,76%**	**99,81%**	**99,84%**

In **Table 5.8**, the results of the *Translators* case study using 1ms interval show a significant and stable percentage decrease in RTTs with more than 99% even for the Windows environments. While for the Linux environments, the RTTs were more optimized than the results with no interval between search requests.

We have noticed that the RTTs obtained during the 1ms interval evaluation on Windows environments were slightly higher than those of the *bookTrading* case study during the same evaluation conditions. This could be due to the increased number of the registered services.

Table 5.8 the performance of the proposed approach with new 04 DF copies in using the bookTrading application and according to requests frequencies with 1ms interval (Zerrougui et al., 2017)

		Number of agents				
		2000	**4000**	**6000**	**8000**	**10000**
HP Windows	JADE's DF	3959	9202	14107	20400	26618
	Proposed approach	36	33	45	43	43
	% Decrease	**99,09%**	**99,64%**	**99,68%**	**99,79%**	**99,84%**
Acer Windows	JADE's DF	2368	11890	20041	29781	37904
	Proposed approach	91	30	54	49	57
	% Decrease	**96,16%**	**99,75%**	**99,73%**	**99,84%**	**99,85%**
Desktop Linux	JADE's DF	6532	11941	18579	25838	33517
	Proposed approach	14	13	13	13	13
	% Decrease	**99,79%**	**99,89%**	**99,93%**	**99,95%**	**99,96%**
HP Linux	JADE's DF	6482	11201	17408	24139	31266
	Proposed approach	12	12	12	12	12
	% Decrease	**99,81%**	**99,89%**	**99,93%**	**99,95%**	**99,96%**
Acer Linux	JADE's DF	9437	14760	23115	33031	43303
	Proposed approach	20	17	16	16	16
	% Decrease	**99,79%**	**99,88%**	**99,93%**	**99,95%**	**99,96%**

3.3.3 *The Third Case Study*

To maintain the original functionalities of the *im2008-jade* application, we have only increased the numbers of requesters to 6000 agent while the number of registered services remained intact. The waiting time of 30 seconds that was set in the requester *setup()* function was removed.

In this case study, the evaluation was performed differently from the previous ones. A more complex MAS named *im2008-jade* was used. The evaluation was performed only with no interval in requests frequency and with 6000 requesters (**Table 5.9**). The number of the returned services varies from 1 to 5 services as programmed in the application.

Table 5.9 the performance of the proposed approach with new 04 DF copies in using the im2008-jade application and according to 6000 requester agents with no interval in requests frequency (Zerrougui et al., 2017)

	JADE's DF (milliseconds)				Proposed approach (milliseconds)				% Decrease (using medians)
	Mean	Median	Min	Max	Mean	Median	Min	Max	
HP Windows	11895	**12078**	15	13618	1,97	**1,42**	0,72	436,46	**99,99%**
Acer Windows	18374	**18762**	6	22081	3,30	**2,28**	0,79	244,95	**99,99%**
Desktop Linux	6826	**7498**	230	11120	3,33	**2,71**	0,84	499,97	**99,96%**
HP Linux	6433	**8003**	11	8732	2,61	**2,11**	0,70	168,34	**99,97%**
Acer Linux	9899	**12104**	10	15167	3,81	**3,08**	1,09	155,10	**99,97%**

As expected, the improvements on the Linux environments were very promising as in the previous case studies. Even more, much higher improvements were also obtained during the no interval evaluation under Windows environments than the previous case studies with more than 99.96% decrease in RTTs.

Even that this MAS is larger and more complex, the results of the proposed approach were much more significant than the first two simple case studies (*bookTrading* and *Translators*). The reasons for such results may be that:

1. Only few services have been registered which means less search space for the DF copies. And yet, a bottleneck has been detected with the default DF. Therefore, the load was nicely handled by the proposed approach.
2. Other functionalities have affected the evaluation. Therefore, simple applications are preferred for better performances evaluations where much load is needed in order to reach bottlenecks situations.

Figure 5.10 shows that the default DF has suffered from weak performance in all the executed environments. In the contrary, the proposed approach has successfully enhanced the performance of the matchmaking service.

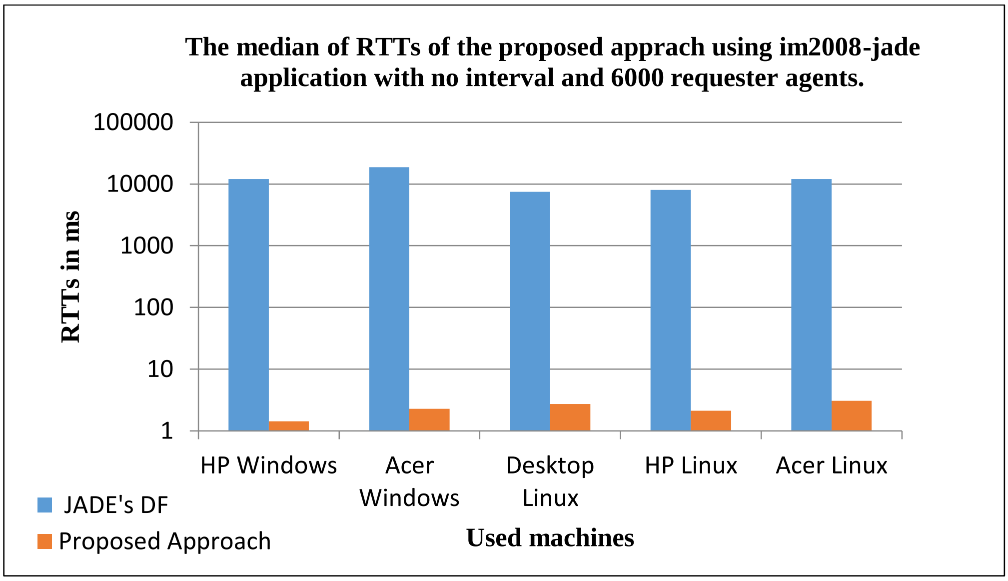

Figure 5.10 Performance comparison between JADE's DF and the proposed approach using im2008-jade application with no interval and 6000 requester agents (Zerrougui et al., 2017)

3.3.4 *The Impact of the Proposed Approach on the CPU and Memory Usage*

To investigate the impact of the proposed approach on the CPU and the memory usage, a profiling process has been performed using JProfiler (Cho, 2006). This latter was performed using the sampling method.

The presented results in **Figure 5.11** and **Figure 5.12** were obtained from the Linux desktop environment and performed on the *bookTrading* case study with 4000 buyers and 100 sellers with no interval on requests frequency. When more buyers (6000 to 10000 buyers) have been used in the profiling process, the registered CPU and memory usage behavior were very similar to each other except for the execution times which they were much longer with higher numbers of buyers.

Figure 5.11 shows the CPU load chart of the proposed approach and the default DF. When the buyers were created at the 11th second, the initial CPU usage was 4.5% in both cases.

As it appears in the height of the proposed approach curve, the CPU usage was increased with an average of 20% than the default DF CPU usage. In the other hand, the overall execution time was decreased by 63% from 118 seconds with the default DF to 44 seconds with the proposed approach.

In fact, the high CPU usage is not a negative point. A software that cannot exploit the full potentials of the execution environment or added resources (better CPU) when it is needed has a scalability problem.

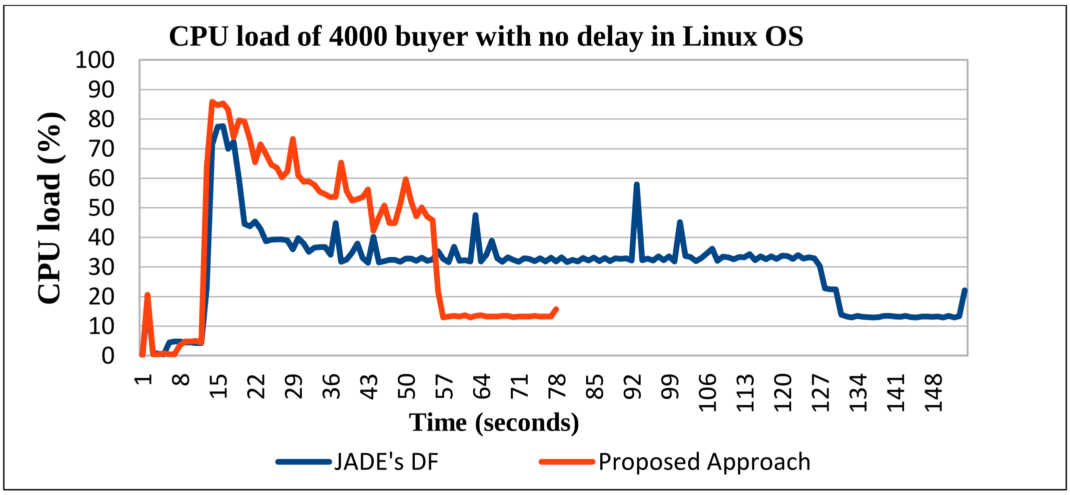

Figure 5.11 CPU load comparison with no interval in requests frequency (Zerrougui et al., 2017)

In **Figure 5.12**, the memory usage chart of the proposed approach and the default DF is presented. During 118 seconds of execution time with the default DF, the profiler has recorded an average size of 339.18 MB in memory usage. But during the 44 seconds execution time with the proposed approach the profiler has recorder 254.83 MB, during this exact time the occupied memory by the default DF was 253.45 MB. Therefore, during 44 seconds, there was a small increase in the memory usage by 0.54% but much less memory occupation time with the proposed approach. While during 118 seconds, there was a significant decrease in memory usage by 24.87% using the proposed approach.

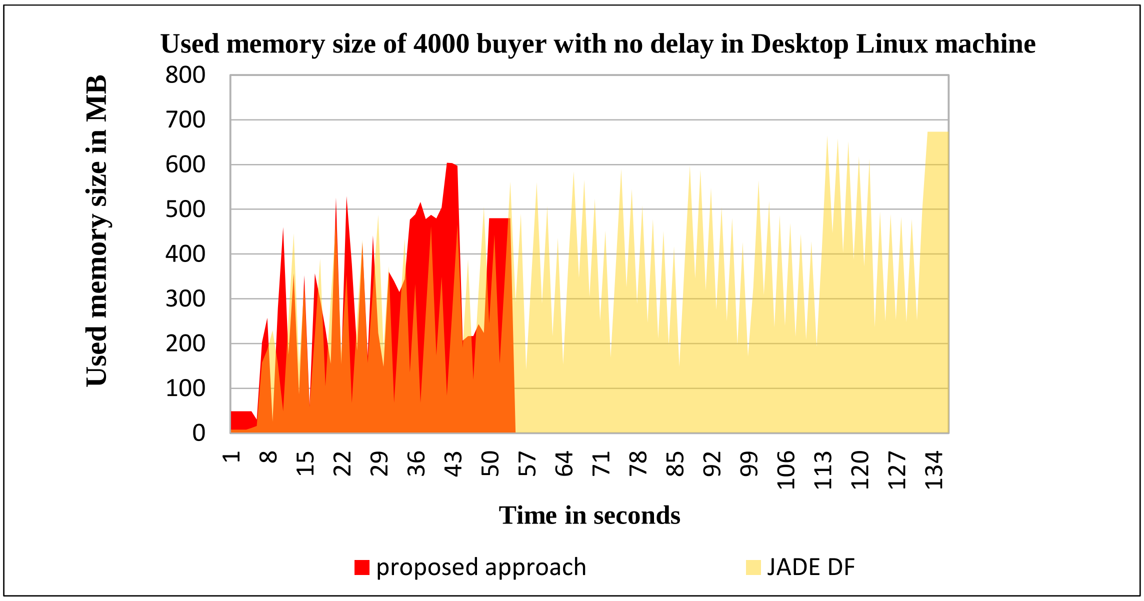

Figure 5.12 Memory usage comparison with no interval in requests frequency (Zerrougui et al., 2017)

4 Conclusion

In multi-agent systems, the Yellow Pages service is implemented in a centralized manner in which a single agent is responsible for all the search requests. In consequence, bottlenecks will occur leading the poor performance and scalability of this service. These delays will propagate through the MAS and can even lead to breakdown situations. Unfortunately, this problem did not get much attention from researchers despite its importance.

In the previous works, the proposed solutions were either: (1) new designs for DF service which cannot be easily integrated in MASs unless they were applied within the used agent platform, or (2) the proposed approaches are not fully developed and need more enhancements to be able to handle all the possible situations in a DF service.

Our proposed approach offers a Yellow Pages replication with a distribution strategy of search requests using aspect-oriented programming. Thus, no changes in the original source code are required and it is easy to implement and to reuse.

The proposed approach was evaluated on three JADE applications (*bookTrading, Translators, im2008-jade)* using five different execution environments. The obtained results are very promising in terms of reducing searches' response times and handling more requests especially on Linux environments with a percentage decrease in RTTs of more than 99% and a success rate of 100% during all the evaluations.

The proposed solution was very beneficial for reducing the CPU and memory usage. But we believe that with more registered services the memory usage may be increased.

Chapter 06

Enhancing Agent's Sociability: A Perfective Maintenance Approach

Chapter 6 Enhancing Agent's Sociability: A Perfective Maintenance Approach

1 Introduction

In this chapter, we will present an agent's sociability problem emerged during interaction protocols. In multi-agent systems, agents interact to solve problems. For better interactions, a variety of protocols are usually used. These interaction protocols have some limitations which appear under different circumstances, whether because of an increased number of agents (Ductor and Guessoum, 2018; Far et al., 2006; Karakostas, 2014; Scerri et al., 2006), slow or overloaded network, blocked or busy agent states. Our contribution consists in introducing learning capabilities to the agent using machine learning algorithms offered by Weka. The suggested perfective maintenance approach will be detailed and validated with several scenarios on two case studies under JADE. The results which are very promising show a great agent's sociability enhancement.

2 Problem Statement

Multi-agent systems offer distributed solutions to various problems, in which, agents can accomplish their goals by coordinating and interacting through interaction protocols like Contract -Net (CNP), request, etc. Conventional interaction protocols are not perfect and need to be extended to better handle some systems' situations (Alarabiat et al., 2014).

Initiators can send initiations to a number of participants, like requests or call for proposals (CFP). These initiations may be bounded to a limited time called deadline, which is the waiting time before starting responses' evaluation. Participants must reply before the deadline reaches, where sometimes, they do not have enough time to do that. Initiations can be delayed due to network problems or posted in the participant message queue for a long time if it was busy or blocked (Aknine et al., 2004).

Participants respond to initiations based on their receiving order. In this case, they can serve an initiation without deadline, an expired deadline or with a deadline that is going to be expired before the answer reaches the initiator. In all these situations, valuable responding times have been wasted that could be used to serve other initiations when they still have enough time. In this case, both initiators and participants have lost possible interactions which might give the best offers (**Figure 6.1**).

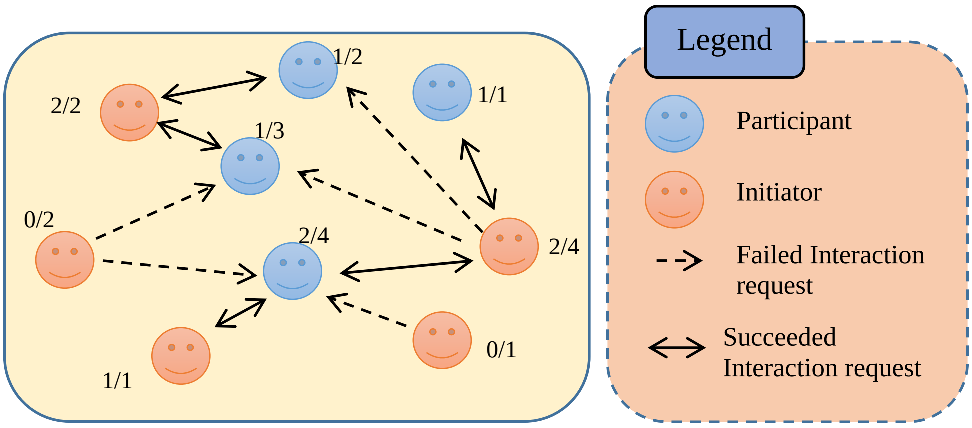

Figure 6.1 Failed and successful interactions between initiators and participants

A possible interaction scenario is presented in **Figure 6.2**. A participant agent receives four CFPs from different initiators to initiate their interactions. The participant can participate to one or multiple interactions simultaneously. In both cases, it cannot prepare proposals at the same time.

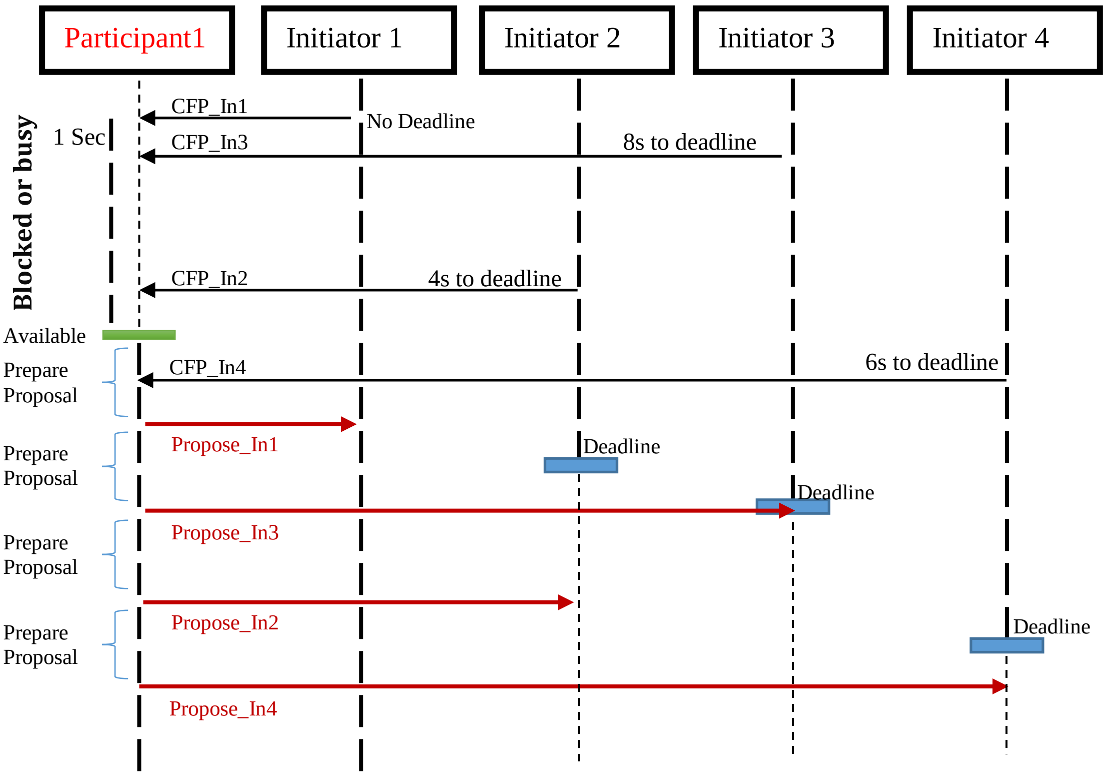

Figure 6.2 A possible interaction scenario with decreased agent's sociability

The CFPs are received at different times with different deadline constraints. The participant which has been unavailable during the reception of the first three CFPs decreases the remaining times for CFP_In2 and CFP_In3, where CFP_In1 has no time limit. The CFP_In4 arrives last during proposal creation for *Initator1*, so it waits until the participant answers all the previous messages.

In the presented scenario, we suppose that there is no network delay and the CFPs are delivered quickly to the participant which can prepare one proposal in about 2 seconds. The remaining time to the deadline for each CFP is when the participant starts to serve it until the deadline reaches. Therefore, the more the CFP waits in the queue the less are the chances to reply before the deadline and to initiate a successful interaction. Increased numbers of CFPs affect others as well because more proposing times for some CFPs are more waiting times for others. If the network is slow then it already has taken some of the deadline constraints which make it even worse.

The participant in the presented scenario was able to reply to the first CFP_In1 which does not have any deadline constraint. This increased the waiting time of the CFP_In3 from 4 seconds when the participant was unavailable to 6 seconds after preparing a proposal for the previous CFP. Adding 2 seconds to prepare its proposal makes it 8 seconds. Even if the delivery time is ignored, the deadline is still reached. So, in this case, it may or may not be succeeded. For the remaining CFPs, it was too late for the participant to respond to them, because their waiting times have exceeded the deadline.

In such scenarios, it is impossible to reply to all the CFPs in time, but managing them at first can increase interaction's initiation chance, which means more successful interactions and better agent's sociability.

3 The Proposed Approach

Many tools and methodologies have been proposed in the literature to support the development of multi-agent systems. However, maintenance of existing multi-agent systems, which is an important research area, is not explored enough (Dam et al., 2013; Gómez-Sanz et al., 2004; Gou et al., 2013).

The proposed approach enhances agent's sociability through interaction protocols and machine learning algorithms. The proposed approach allows the participant agent to avoid responding to three initiation cases while its message queue is not empty: (1) the initiation has no deadline, (2) the initiation with an expired deadline, or (3) the initiation with an expiring deadline before the response reaches the initiator.

The proposed approach's goal is to prioritize the received interactions requests based on their probability of success (**Figure 6.3**). According to the initiations types (CFPs), three priority levels have been proposed.

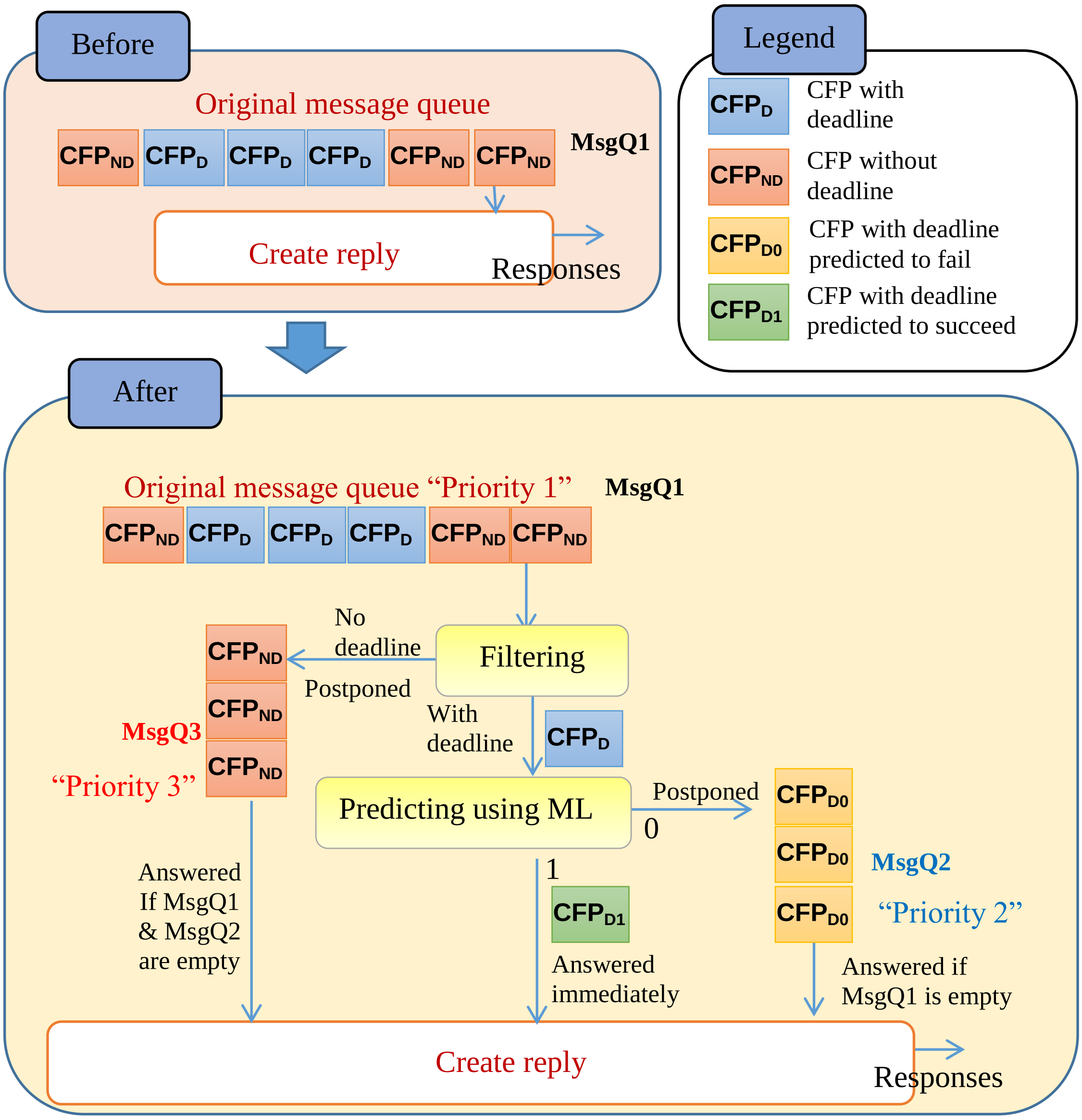

Figure 6.3 CFP message handling in the participant agent side using the proposed approach

An interaction initiation (CFP in Contract-Net) may either be bounded to a given deadline or not. If a deadline exists, this means that the initiator will evaluate the replies only when the deadline reaches or when all the replies are received even before the deadline. If there is no deadline then the initiator choses to wait for all the replies from all the contacted participants before it starts their evaluation.

At first, two types of CFPs can be categorized in the agent default message queue (MsgQ1): (1) CFPs with deadlines called **"CFP_D"** and (2) CFPs with no deadlines called **"CFP_{ND}"**. In the proposed approach, CFP_{ND} without deadlines are the last served ones in the 3rd priority, so they must be postponed in another message queue (MsgQ3) and answered when all the CFP_D with deadlines are served. The CFP_D with deadlines can be categorized into two categories as well using a prediction process, which are: (a) **"CFP_{D1}"** which their replies may reach initiators before the deadlines (successful reply) and (b) **"CFP_{D0}"** which their replies may not reach before the deadline (failed reply). The CFP_{D0} that has low chance to be replied in time is postponed for later in another message queue in the 2nd priority (MsgQ2). Therefore, the remaining CFP_{D1} types which have a higher chance to successfully initiate an interaction are in the 1st priority, so they must be answered immediately.

To predict if a CFP_D has a chance to be replied before its deadline or not, the participant agent must save the history of its interactions. The saved data are used as training set for several machine learning algorithms. Next, these algorithms are validated to be able to select the best model for the prediction process (enhanced mode). While the interaction is in the enhanced mode, the participant keeps registering its history to retrain the algorithms afterwards (**Figure 6.4**).

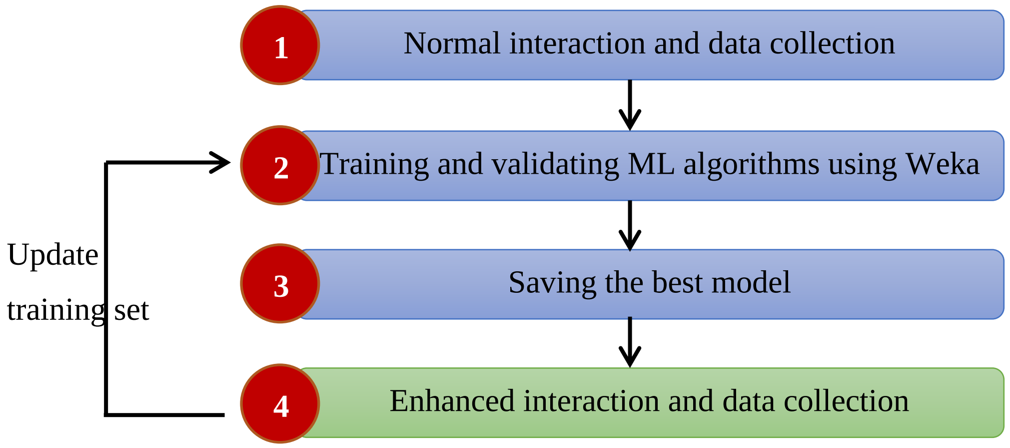

Figure 6.4 The principal steps of the classification process

3.1 Normal Interaction and Data Collection

The proposed approach consists of predicting if the current interaction initiation is going to be successful or not based on three attributes: *Remaining_time*, *Reply_time* and a class attribute called *outcome*.

The first attribute is the *Remaining_time* (**Figure 6.5**), which is the difference between the deadline time and the message's read time (when it is extracted from the message queue) as calculated in *(Eq. 6.1*. If the system under enhancement is distributed on different machines, then these machines must have a synchronized time, which can be easily done using a unique time server.

$$Remaining\_time = Deadline\_time - Read\_time \qquad \text{(Eq. 6.1)}$$

The second attribute is the *Reply_time* (**Figure 6.5**), which is the sum of two times *(Eq. 6.2*: (a) The proposing time, which is the necessary time to prepare only one proposal. It can be variable because it depends on the executed task and the used environment. Therefore, the mean of proposing times is calculated after each proposal to be used as the proposing time. (b) The second time is the delivery or the transmission time, which is also estimated. It represents the necessary time to deliver the proposal to the initiator. For distributed machines, sending a ping to the initiator can be one solution to obtain the estimated delivery time, but for slow networks this is not appropriate. Another way to obtain an estimated delivery time is by calculating the time difference between the time of a message posting in the participant message queue and its sending time. This requires a synchronized time between the hosts as mentioned before. The best delivery time estimation is when the agent development platform offers the function which records it. For this study, we are interested on local interactions only; therefore, the *Reply_time* attribute will not be as effective as the *Remaining_time*, unless the participant offers different services with different proposing times.

$$Reply\_time = Proposing\_time + Delivery\_time \qquad \text{(Eq. 6.2)}$$

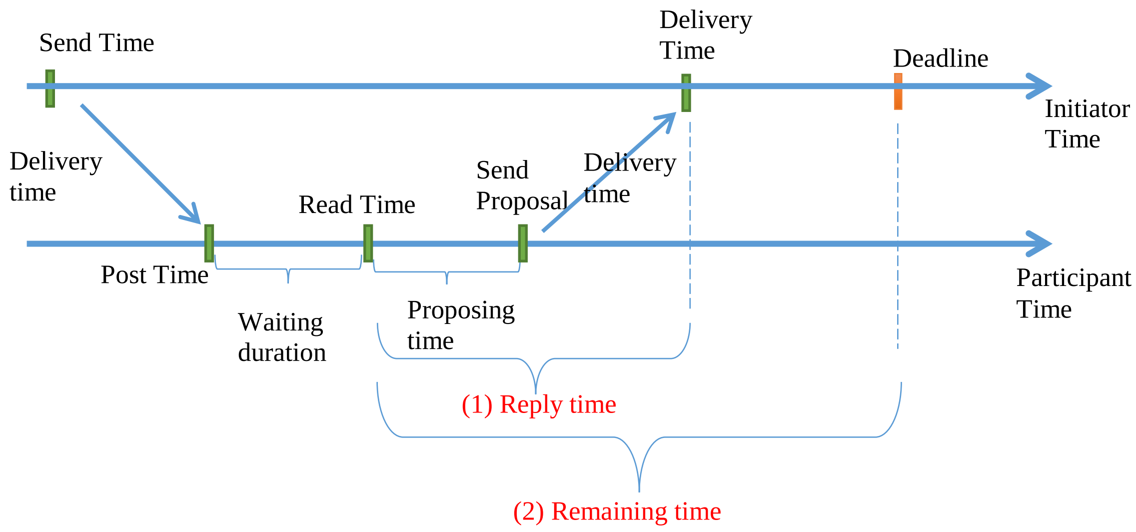

Figure 6.5 The proposed attributes

The third attribute is the class named *outcome* initialized with "0". When the initiator responds back to the participant's proposal with an accept or a reject, this means that the interaction was successful and the outcome attribute gets updated with "1".

The training set is collected during a normal interaction (**Snippet 6.1**), where only initiations with deadline constraints are committed at this stage. For every initiation, the participant creates a data instance containing the calculated remaining time, reply time and the outcome of the received initiation (**Figure 6.6**). When the desired training set size is achieved then the participant starts the training phase.

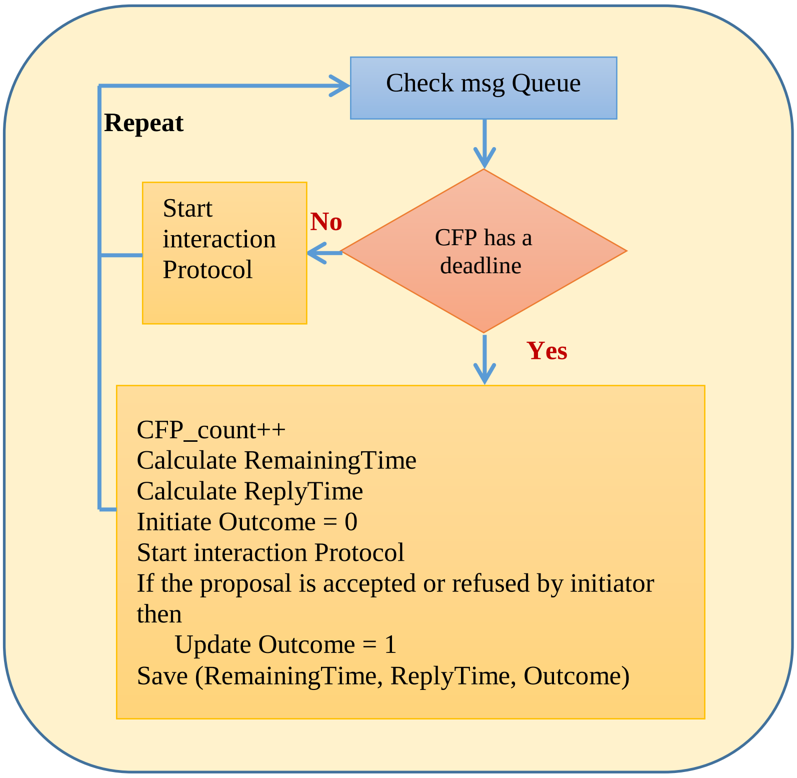

Figure 6.6 The data collection phase of the participant agent

Snippet 6.1 A Data set sample in Weka's ".arff" file format

```
@RELATION participant1
@ATTRIBUTE RemainingTime NUMERIC
@ATTRIBUTE ReplyTime NUMERIC
@ATTRIBUTE result {1, 0}
@DATA
1089, 1944,0        // Data instance
-6880, 1896,0
6541, 2500,1
.
.
```

3.2 Training and validating machine learning algorithms using Weka

The training phase is presented in **Figure 6.7**. When the number of received initiations reaches a fixed value "n" and it is allowed to start the next phase, then the gathered data is used to train and validate several machine learning algorithms (binary classifiers). The used classifiers are: voted perceptron (Freund and Schapire, 1999), logistic regression (Le Cessie and Van Houwelingen, 1992), multilayer perceptron, stochastic gradient descent (SGD) (Ruder, 2016), simple logistic (Landwehr et al., 2005; Sumner et al., 2005), random forest (Breiman, 2001), random tree (Drmota, 2009), support vector machine using sequential minimal optimization (SMO) (Keerthi et al., 2001; Platt, 1999), naïve Bayes (John and Langley, 1995) and Bayes networks (Bouckaert, 2008) are validated using the 10 fold cross-validation method. The participant selects and saves the best model to be used later for predicting interactions' outcomes.

The 10 fold cross-validation technique is used to obtain an estimate of the classifier's performance, by exploiting the entire dataset. This is achieved by doing several test iterations on different training and test sets, then averaging the results. The validation consists of randomly segmenting the initial data set into 10 disjoint subsets numbered from 1 to 10. At every test's iteration, only one subset is used as a test set while the other 9 subsets are used as a training set.

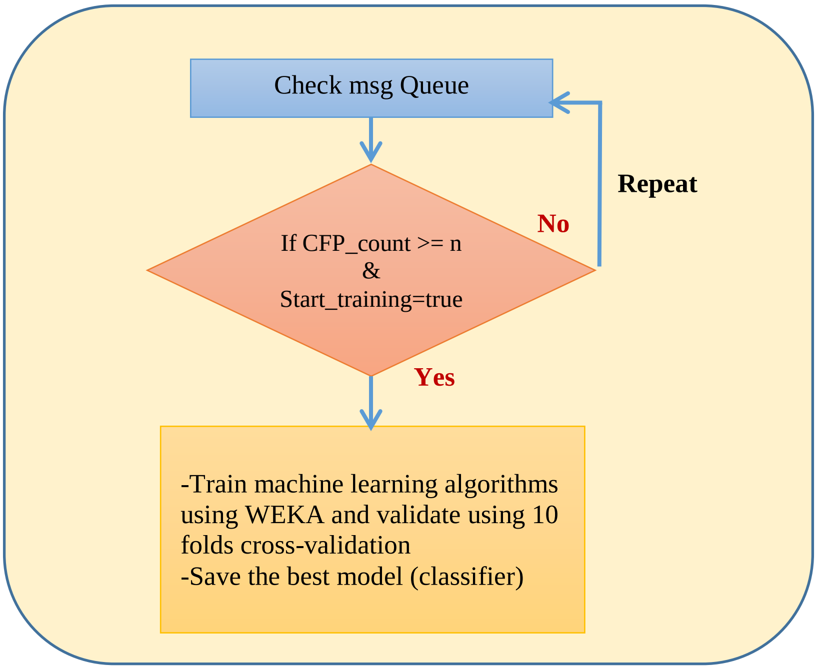

Figure 6.7 Training and validating phases

3.3 The Enhancement Phase

Snippet 6.2 presents a pseudo-code which describes the enhancement process. As mentioned before, the participant can receive two types of CFP messages: CFP_D with deadlines and CFP_{ND} without deadlines. When a CFP_D is received, the participant predicts its outcome by calculating its remaining and reply times and passing them to the selected prediction model. If the *predictedOutcome* is 1 then the participant responds to this CFP_{D1} right away, otherwise, this CFP_{D0} will be postponed in a list named *postponedWithDeadlineList* (MsgQ2). If the received CFP_{ND} has no deadline then it will be postponed in a list called *postponedNoDeadlineList* (MsgQ3). Consequently, the participant has gained a valuable time to respond to more CFP_{D1} messages before they wait too much in the message queue and become like a CFP_{D0}.

The participant starts to respond to the postponed CFP_{D0} which have deadline constraint only when the original message queue (MsgQ1) is empty. These CFP_{D0} could be postponed for a long time in MsgQ2 depending on the arrival of new CFPs. Thus, it is very possible that the deadline of a CFP_{D0} has already passed. Therefore, only the postponed CFP_{D0} in which their deadline still remains which is unlikely to be exist or with a deadline which passed a while ago (must use a threshold time) are replied to. So, by gaining these reply times, it is slightly possible to find a postponed CFP_{D0} which is still available for interaction.

Each time a CFP is treated, the participant saves a key as an interaction identifier along with its *remaining_time*, *reply_time*, the *predictedOutcome* and an extra field initialized with "0" called *realOutcome*. The *realOutcome* can be updated to "1" if the participant receives accept or reject message from the initiator. The *realOutcome* can be used along with its remaining time and reply time to enrich to training data set and also to evaluate the performance of the enhancement process. Therefore, it is possible for the participant after responding to a fixed number of messages to repeat the learning phase with the updated training set and selects the new best model as it did before.

Snippet 6.2 Pseudo-code for sociability enhancements in the participant agent

```
While agent is activated do
{
cfp = get the first cfp from the original message queue (MsgQ1)
If cfp not null then     // if cfp is null then the MsgQ1 is empty
        If cfp has a deadline then
                Calculate remaining_time
                Calculate reply_time
                Predict predictedOutcome using the saved model
                realOutcome = 0
                SaveCFP ( CFPkey, {remaining_time, reply_time, predictedOutcome,
                realOutcome} )
                If predictedOutcome == 1 then
                        Start interaction protocol
                Else
                Postpone cfp to postponedWithDeadlineList (MsgQ2)
        Else
                Postpone cfp to postponedNoDeadlineList (MsgQ3)
Else
        If postponedWithDeadlineList is not empty then
                cfp = get the first cfp from postponedWithDeadlineList
                If currentTime < cfp.deadlineTime then
```

Start interaction protocol
Else
If *postponedNoDeadlineList is not empty* **then**
cfp = get the first cfp from postponedNoDeadlineList
Start interaction protocol
Else ;
Block agent until next cfp is received
}

The following sequence diagram (**Figure 6.8**) shows the same scenario presented in the problem statement's section (**Figure 6.2**), but performed using the proposed approach. The participant agent begins to reply to the received CFPs when it is available. The participant replies to the CFP_In3 first, instead of the CFP_In1 which was filtered as a CFP_{ND} type and postponed for last. The CFP_In3 has 8 seconds before its deadline, where 4 seconds of them was wasted while the participant was blocked; this leaves only 4 seconds until the deadline as a remaining time. During this remaining time, the participant will spend approximately 2 seconds to send a reply leaving extra 2 seconds.

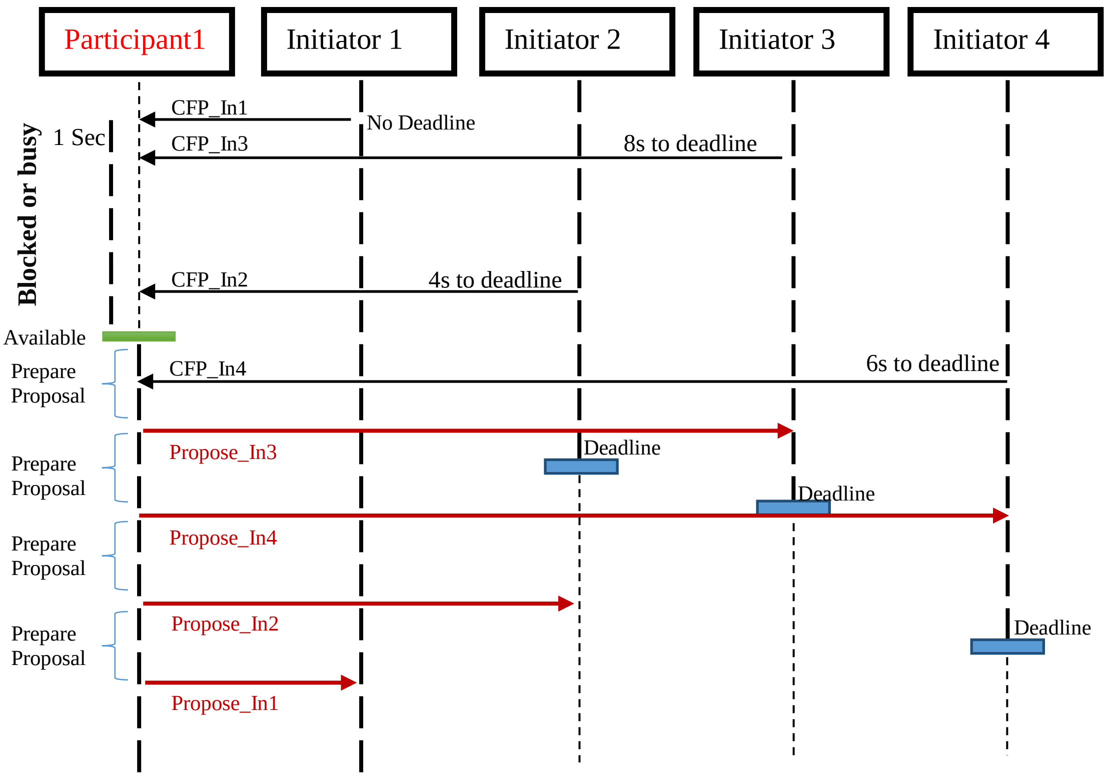

Figure 6.8 The previous interaction scenario using the proposed approach

Based on agent history and the two attributes (*Remaining_time* and *Reply_time*), the participant has predicted that the reply to this CFP_In3 has a high chance of success and replies to it immediately. Using the same process for the next CFP_In2 in message queue, gives a remaining time of 1 second and a reply time of 2 seconds. In this case, the participant has predicted that the reply of CFP_In2 has a low chance and it is most likely going to fail. Therefore, the CFP_In2 was postponed for later and the CFP_In4 takes its place with a remaining time of 5 seconds, reply time of 2 seconds and a prediction of a successful reply. When the participant has no other messages waiting in the queue, it starts to reply to the postponed CFP_{D0}. The CFP_In2 is the message to be replied at this step, which logically it should not be, because the deadline has already passed. If the postponed CFP_{D0} list was not empty, then it would be inappropriate to reply to CFP_In2, because it just wastes 2 seconds of reply time that will delay the next postponed messages ($CFP_{D0)}$. When all the CFP_D have been replied, the participant answers to the CFP_{ND} messages postponed earlier. The count of the successfully replied CFPs using the proposed approach is 03 out of 04 replies. In the other case, only 1 message out of 04 is surely successful because one of the replies will arrive to the initiator approximately at the same time when the deadline reaches, so it could be failed after counting initiators reading tasks.

4 Validation and Discussion

To validate the proposed approach, two case studies have been proposed. Both of them are implementations of the Contract-Net protocol under JADE platform. In each case study, different experiments have been performed under different conditions to create multiple scenarios. Our proposed approach is addressed to interaction protocols in which participants receive multiple initiations that must be responded to before their deadlines. Since that there are no open access Jade projects or benchmarks available in the agent community, the researchers are forced to develop their own case studies or relay on JADE examples. Fortunately, interaction protocols as presented previously are social patterns, which means that their implementations are similar in all their applications under the same agent platform. The difference between them is in the complexity and the load of the interactions.

4.1 The First Case Study

In each scenario of this implementation, multiple initiators are created to send CFPs to the same participant, where each CFP message may or may not have a deadline constraint (randomly). Furthermore, a time interval of 1 second has been used between initiators' creation to reduce the load. The participant agent takes about 2 seconds in average to only prepare the proposal message.

In the data collection phase, the participant agent interacts in a normal CNP to register its interactions' history. When the training set reaches 1000 instances, the participant trains multiple machine learning classifiers (training phase) offered by *Weka* and validates them using the 10 fold cross-validation method. The used classifiers are logistic regression, multilayer perceptron, stochastic gradient descent (SGD), simple logistic, voted perceptron, random forest, random tree, support vector machine (SVM - SMO), naïve Bayes and Bayes network.

The Weka's confusion matrix definition is presented in **Table 6.1**. If we suppose that the interaction success class "a" to be the positives and the interactions failure class "b" as the negatives, then

- TP (true positives) are the interaction correctly predicted to succeed,
- FP (false positives) are the interactions incorrectly predicted to succeed,
- TN (true negatives) are the interactions correctly predicted to fail,
- FN (false negatives) are the interactions incorrectly predicted to fail.

Table 6.1 The confusion matrix definition in Weka for interactions' predictions

		Predicted	
		To succeed (a)	To fail (b)
Actual	Interaction success (a)	TP	FN
	Interaction failure (b)	FP	TN

Table 6.2 presents the confusion matrix of the logistic regression's 10 fold cross-validation. Using this matrix, we can find the number of correctly and incorrectly predicted interactions, which leads us to the first metric called the **Accuracy**. This latter is the ratio of the number of correct predictions on the total number of all predictions. In the case of this classifier, accuracy is 99.5%, where 995 interactions were correctly classified and 5 interactions were incorrectly classified.

Table 6.2 The confusion matrix of the logistic regression classifier validated using 10 fold cross-validation

		Predicted	
		To succeed (a)	To fail (b)
Actual	Interaction success (a)	375	3
	Interaction failure (b)	2	620

Table 6.3 shows some detailed metrics for each class resulted from the validation of the logistic regression model using 10 fold cross-validation. The **TP Rate (True Positive Rate)** also called the **recall** or the **sensitivity** is the fraction of the total number of relevant instances that were actually classified. In other words, it is the number of the correct predictions in a given class divided by the total elements of that class.

Weka offers the **weighted average** of its rates which is calculated to give a single value for both classes *(Eq. 6.3)*. The weighted average is used instead of a simple average because of the imbalance that can be present in the data set. Imbalanced classes mean that a given class has more instances than the other. Therefore, the number of instances is used as weights to calculate the weighted average. In the used training set there are 378 instances labeled as class "a" ($Weight_a$) and 622 labeled as class "b" ($Weight_b$).

$$Weighted\ Avg. Recall|TP\ Rate = \frac{(Recall_a \times\ Weight_a) + (Recall_b \times\ Weight_b)}{(Weight_a +\ Weight_b)} \quad \textit{(Eq. 6.3)}$$

The **FP Rate (False Positive Rate)** is the number of the incorrect predictions in a given class divided by the total elements of the other class. If class "a" is the positives then the FP rate is the number of interactions incorrectly predicted to succeed divided by the number of all the failed interactions.

Table 6.3 The detailed results for the 10 fold cross-validation of the logistic regression classifier

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC	PRC	class
	0,992	0,003	0,995	0,992	0,993	0,999	0,999	1
	0,997	0,008	0,995	0,997	0,996	0,999	0,999	0
Weighted Avg.	0,995	0,006	0,995	0,995	0,995	0,999	0,999	

The Precision which is also called the **positive predicted value** (**PPV)** can answer the question: how many selected elements are relevant? It is the number of the correct predictions of a class divided by the total of predictions of the same class.

The **F-Measure** is the harmonic mean of precision and recall which gives an overall score that depicts how well our model is performing? **Weighted average F-Measure** is calculated using the f-measures of both classes and not with the weighted averages of precision and recall.

Weka also offers very important metrics for each class which are the ROC (**Receiver Operating Characteristic**) and PRC (**Precision- Recall**) curves. ROC curve is constructed using TPR and FPR, when the PRC is created using precision and recall. The values presented by Weka for these metrics are called the **Area Under Curve (AUC)**. These AUC values are obtained using the *getROCArea()* and *getPRCArea()* functions of *weka.classifiers.evaluation.ThresholdCurve* java class. The AUC is the probability that a classifier will predict an instance correctly.

In certain applications, one class can be more important than the other, so the classifiers must have some trade-off between metrics to be able to classify the desired class as best as possible even if the classification performance for the other class decreases to an acceptable limit (ex. healthy or diseased patient). In this study, predicting correctly in both classes is very important, because if the model classifies an actual fail interaction as a successful one then the participant will lose time in processing its CFP message, which propagates over the rest of the CFPs in the message queue. Therefore, to better describe the results of both classes in less space the weighted average of the previous metrics is presented.

Table 6.4 summaries the results of the 10 fold cross-validation method applied on the selected classifiers. The accuracy of the used classifiers was very promising starting from 93.1% with the naïve Bayes algorithm to 99.6% with simple logistic and voted perceptron. As reported in previous studies, accuracy can be misleading in imbalanced datasets. Consequently, comparing classifiers using ROC and PRC is more appropriate. The ROC is based on TP rate and FP rate, the best ROC values are found with high TP rates and low FP rates which are the cases with logistic regression, multilayer perceptron, simple logistic and random forest where the TP rate is more than 99.3% and the FP rate is less than 0.7% with AUC of 0.999. The PRC is based on precision and recall, which both of them must have high values for the PRC to be good which is the case for the same classifiers with also the best ROC values where there are more than 99.3% in precision and recall and 0.999 in PRC. Other classifiers also performed very well on this validation, noting that SMO and naïve Bayes have registered more false positives and false negatives than the other classifiers but they still have very good ROC and PRC values. Since multiple classifiers have the same performance measures, the logistic regression has been selected randomly among them to be used as the model for predicting interactions outcome in the enhancement phase.

Table 6.4 The validation’s results of the used classifiers using 10 fold cross-validation for the first case study

	Confusion Matrix a = 1 | b = 0	Accuracy	Weighted Avg. Precision	Weighted Avg. Recall / TP Rate	Weighted Avg. FP Rate	Weighted Avg. F-Measure	Weighted Avg, ROC	Weighted Avg. PRC
Logistic Regression	a: 375 | 3 b: 2 | 620	99,5%	0,995	0,995	0,006	0,995	0,999	0,999
multilayer perceptron	a: 377 | 1 b: 3 | 619	99,6%	0,996	0,996	0,003	0,996	0,999	0,999
SGD - Hinge loss	a: 363 | 15 b: 4 | 618	98,1%	0,981	0,981	0,027	0,981	0,977	0,971
Simple logistic	a: 376 | 2 b: 2 | 620	99,6%	0,996	0,996	0,005	0,996	0,999	0,999
Voted perceptron	a: 376 | 2 b: 9 | 613	98,9%	0,989	0,989	0,009	0,989	0,993	0,986
random forest	a: 375 | 3 b: 4 | 618	99,3%	0,993	0,993	0,007	0,993	0,999	0,999
Random tree	a: 375 | 3 b: 2 | 620	99,5%	0,995	0,995	0,006	0,995	0,994	0,992
SMO	a: 353 | 25 b: 12 | 610	96,3%	0,963	0,963	0,048	0,963	0,957	0,944
Naïve Bayes	a: 365 | 13 b : 56 | 566	93,1 %	0,936	0,931	0,055	0,932	0,986	0,988
Bayes Network	a: 376 | 2 b: 13 | 609	98,5%	0,985	0,985	0,011	0,985	0,996	0,995

In the enhancement phase, different interaction scenarios have been performed to validate the proposed approach. In each scenario (iteration), there was only one participant which receives one CFP from multiple initiators. The number of CFPs was increased by 20 CFPs in each new scenario, starting with 20 initiators in the first one to 500 initiators in the last one (25 iterations). The CFPs’ deadlines were randomly generated from 0 to 14 seconds then saved to be used twice in the same order during the same iteration, first for normal CNP’s interactions and secondly for the proposed approach’s interactions (each iteration has one scenario performed twice, which means 50 experiments in total). The 25 scenarios are totally random and different than each other. The participant registers its predictions and its corresponding real outcomes, to be able to evaluate the enhancement process. During these evaluations, the registered real outcomes are not used for updating the data set or the model.

The results of the normal CNP and the proposed approach are presented in **Table 6.5**. The improvement percentage (percentage increase) defined by *(Eq. 6.4)* has been used for each scenario to show the impact of the proposed approach on the sociability of the participant agent.

$$Imrpovement\ percentage = \left(\frac{Succeeded\ CFP_D\ Enhanced\ CNP - Succeeded\ CFP_D\ Normal\ CNP}{Succeeded\ CFP_D\ Normal\ CNP}\right) \times 100 \qquad (Eq.\ 6.4)$$

As expected in both approaches, all the CFP_{ND} without deadlines were replied successfully. Unfortunately, this was not the case for the CFP_D. Only 1 to 15% of the CFP_D were succeeded during the majority of the normal CNP scenarios. When the number of CFPs was increased, the number of succeeded replies in the normal CNP was very low and stable during all the scenarios. This is explained in the increased count of the failed replies, where the more CFPs were involved the more fails were obtained.

Table 6.5 Replied CFPs using normal CNP and the proposed approach for the first case study

			Normal CNP		**Proposed approach**		
Number of CFPs	CFP_{ND} without deadline	CFP_D with deadline	Succeeded CFP_D	Failed (reached after deadline)	Succeeded CFP_D	Failed (reached after deadline)	Improvement Percentage (With deadline)
20	0	20	**4**	16	**15**	5	**275%**
40	2	38	**6**	32	**23**	15	**283%**
60	4	56	**8**	48	**26**	30	**225%**
80	9	71	**8**	63	**38**	33	**375%**
100	8	92	**6**	86	**53**	39	**783%**
120	12	108	**8**	100	**59**	49	**638%**
140	16	124	**8**	116	**47**	77	**488%**
160	11	149	**5**	144	**80**	69	**1500%**
180	18	162	**6**	156	**86**	76	**1333%**
200	15	185	**4**	181	**66**	119	**1550%**
220	11	209	**10**	199	**107**	102	**970%**
240	16	224	**5**	219	**113**	111	**2160%**
260	17	243	**9**	234	**120**	123	**1233%**
280	19	261	**8**	253	**136**	125	**1600%**
300	28	272	**9**	263	**92**	180	**922%**
320	20	300	**7**	293	**91**	209	**1200%**
340	26	314	**9**	305	**165**	149	**1733%**

360	30	330	**8**	322	**173**	157	**2063%**
380	25	355	**8**	347	**180**	175	**2150%**
400	19	381	**7**	374	**116**	265	**1557%**
420	27	393	**8**	385	**117**	276	**1363%**
440	25	415	**8**	407	**114**	301	**1325%**
460	32	428	**3**	425	**115**	313	**3733%**
480	42	438	**6**	432	**230**	208	**3733%**
500	24	476	**7**	469	**233**	243	**3229%**

In the proposed approach, the results were better, where the participant was able to successfully reply to 30 up to 75% of the CFP_D. When the number of CFPs was increased, the successful replies were increased as well, even that they were sometimes decreased compared to previous iteration as shown in **Figure 6.9**. This is because each scenario has different deadlines in different order than the previous one.

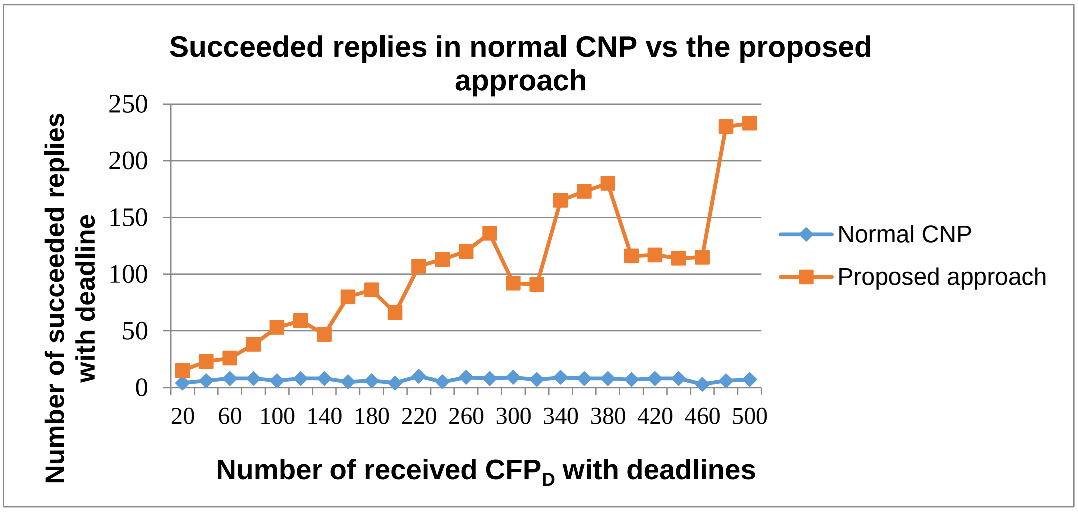

Figure 6.9 Comparison of the number of succeeded replies between the normal CNP and the proposed approach for the first case study

Compared to the normal CNP, the failed replies number has been decreased significantly as shown in **Figure 6.10**. In a percentage point of view, the participant maintained about 29% of the CFP_D as its failed replies percentage. The improvement percentages in **Table 6.5** show the effectiveness of the proposed approach in improving agent's sociability up to 3700%, especially when the received CFPs number is increased.

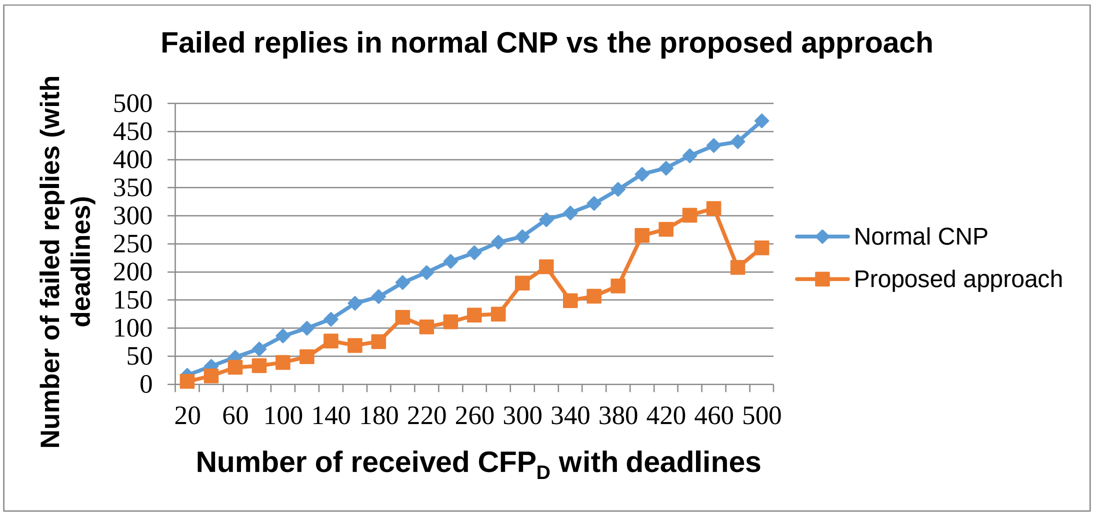

Figure 6.10 Comparison of the number of failed replies between the normal CNP and the proposed approach for the first case study

4.2 The Second Case Study

In this implementation, the participant offers 3 different services: (i) A data search service which takes 10 seconds to perform, (ii) a file download service which takes 20 seconds to perform, and (iii) a file upload service which takes 30 seconds to perform. Initiators can request only one service with 1 or 2 minutes as deadlines (CFP_D only). Furthermore, 10 seconds interval has been used between initiators' creation (**Figure 6.11**). The validation has been performed in 8 iterations designated by CFP_D number starting from 10 to 200 CFP. Three (03) scenarios have been performed for each iteration which means 24 different scenarios. Each scenario is performed twice for the normal CNP and the proposed approach (48 experiments in total).

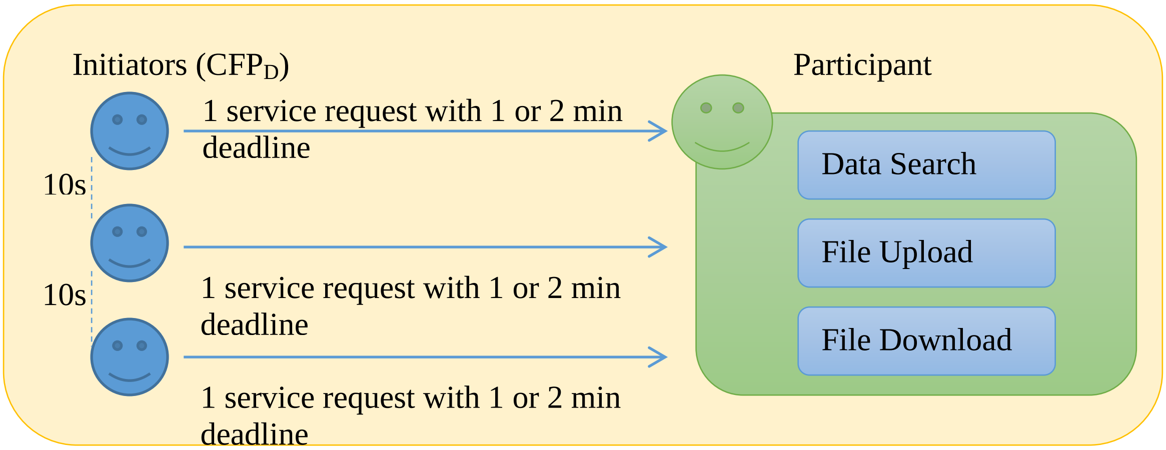

Figure 6.11 A scenario example of the second case study

The previously selected classifiers were applied on this case study as well. These classifiers were validated using the 10 fold cross-validation method. This data set has 1035 instances which contain 568 instances labeled as class 0 and 467 instances labeled as class 1. Noting that the *Remaining_time* attribute in this case cannot predict the outcome alone because there are three different proposing times depending on the selected service. Therefore, the *Reply_time* attribute has more impact on the prediction process differently than the first case study. The validation results presented in **Table 6.6** show very high-performance scores in all the classifiers especially with logistic regression which is used as the prediction model for the enhancement phase.

Table 6.6 The validation's results of the used classifiers using 10 fold cross-validation for the second case study

	Confusion Matrix a = 1 | b = 0	Accuracy	Weighted Avg. Precision	Weighted Avg. Recall / TP Rate	Weighted Avg. FP Rate	Weighted Avg. F-Measure	Weighted Avg, ROC	Weighted Avg. PRC
Logistic Regression	a: 568 | 0 b: 1 | 466	99.9%	0.999	0.999	0.001	0.999	1.000	1.000
multilayer perceptron	a: 568 | 0 b: 6 | 461	99.4%	0.994	0.994	0.007	0.994	1.000	1.000
SGD - Hinge loss	a: 549 | 19 b: 0 | 467	98.1%	0.982	0.982	0.015	0.982	0.983	0.974
Simple logistic	a: 567 | 1 b: 1 | 466	99.8%	0.998	0.998	0.002	0.998	1.000	1.000
Voted perceptron	a: 566 | 2 b: 0 | 467	99.8%	0.998	0.998	0.002	0.998	0.998	0.997
random forest	a: 567 | 1 b: 1 | 466	99.8%	0.998	0.998	0.002	0.998	0.999	0.998
Random tree	a: 567 | 1 b: 1 | 466	99.8%	0.998	0.998	0.002	0.998	0.998	0.997
SMO	a: 540 | 28 b: 0 | 467	97.2%	0.974	0.973	0.022	0.973	0.975	0.962
Naïve Bayes	a: 539 | 29 b : 0 | 467	97.1%	0.974	0.972	0.023	0.972	1.000	1.000
Bayes Network	a: 549 | 19 b: 13 | 454	96.9%	0.969	0.969	0.030	0.969	0.998	0.998

The results in **Table 6.7** show the scalability problem during the normal CNP evaluation. In this case, the participant couldn't increase its sociability when more initiators (or CFPs) were involved. The percentage of the succeeded replies has been dropped from 70% in the first iteration to 2% and 3% in the last two iterations. In the contrary with the proposed approach, the participant has successfully replied from 50% to 70% of the received initiations during all the iterations. The improvement percentage of all the performed scenarios has been presented as well. The enhancement was noticed from the 3rd scenario of the first iteration with 40%

performance increase up to 2800% performance increase in the 2nd scenario of the 7th iteration.

Table 6.7 The successfully replied CFPs during the second case study

Number of CFPs	Normal CNP			The proposed approach			Improvement percentage		
	1st	**2nd**	**3rd**	**1st**	**2nd**	**3rd**	**1st**	**2nd**	**3rd**
10	7	7	5	7	7	7	0%	0%	40%
20	6	8	11	13	12	13	117%	50%	18%
30	5	13	4	20	21	16	300%	62%	300%
40	4	7	6	23	23	25	475%	229%	317%
50	6	10	7	29	26	31	383%	160%	343%
100	8	10	9	51	53	59	538%	430%	556%
150	9	3	5	82	87	88	811%	2800%	1660%
200	11	5	8	105	122	111	855%	2340%	1288%

To visualize the agent's sociability difference between the normal CNP and the proposed approach, the means of the succeeded replies' percentages have been used (**Table 6.8**). In each iteration, the mean of the three scenarios is calculated then turned into a percentage using *(Eq. 6.5)*.

$$Mean\ percentage\ of\ succeedded\ replies = \frac{Mean(1st, 2nd, 3rd)}{Number\ of\ CFPs} \qquad \textit{(Eq. 6.5)}$$

Table 6.8 The mean percentages of the succeeded replies for the second case study

Number of CFPs	Normal CNP	The proposed approach	Improvement percentage
10	63%	70%	13%
20	42%	63%	62%
30	24%	63%	221%
40	14%	59%	340%
50	15%	57%	295%
100	9%	54%	508%
150	4%	57%	1757%
200	4%	56%	1494%

As presented in **Figure 6.12**, when the number of CFP_D was increased the normal CNP couldn't stabilize the participant's sociability. The succeeded replies mean percentage was dropped quickly and remarkably from 63% to 4%. In the other hand, the proposed approach has recorded a slow and a slight drop in the participant's sociability which was maintained above 54% during all iterations.

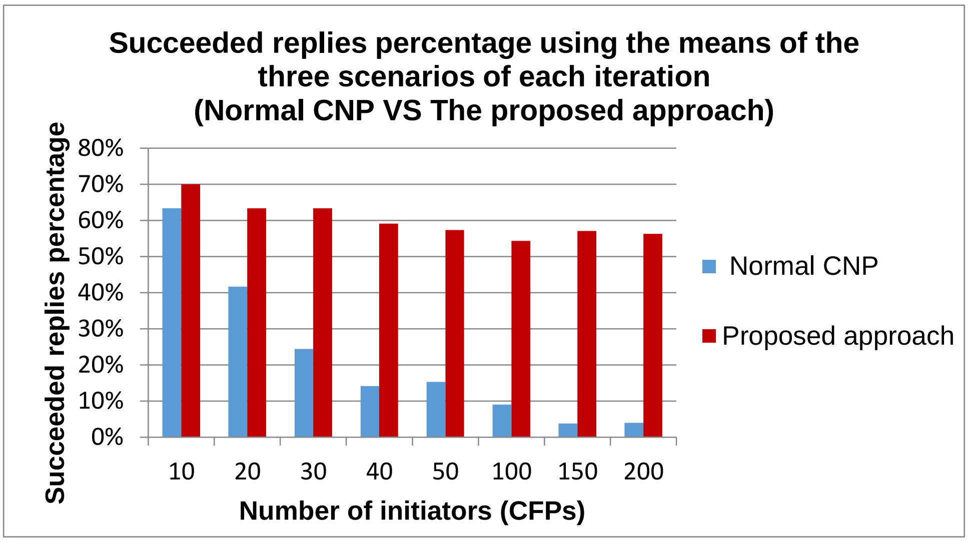

Figure 6.12 Comparing mean percentages of the succeeded replies for the normal CNP against the proposed approach for the second case study

4.3 Performance Measurements

In order to measure the performance of the proposed approach, a confusion matrix must be formed for all iterations using the *predictedOutcome* and *realOutcome* variables described before. Afterwards, recall, precision, accuracy and f-measure defined previously are used as performance metrics.

The obtained performance measures presented in **Table 6.9** shows significant results in all the used metrics. Very few false positives and false negatives have been registered in this evaluation. Since the metrics are nearly stable through all interactions from 96.6% to 100% scores then the performance of the proposed approach is not affected by the increased number of messages in terms of correct predictions and not in terms of increased agent interactions. But, to evaluate the scalability and the performance limits of the proposed approach more tests are needed with higher numbers of messages.

Table 6.9 Performance measures of the proposed approach for the first case study

	Predicted to succeed CFP_{D1}		Predicted to fail CFP_{D0}					
Number Of CFPs	**Correct (TP)**	**Incorrect (FP)**	**Correct (TN)**	**Incorrect (FN)**	**Recall**	**Precision**	**Accuracy**	**F-measure**
20	15	0	5	0	100,00%	100,00%	100,00%	100,00%
40	25	0	15	0	100,00%	100,00%	100,00%	100,00%
60	30	0	30	0	100,00%	100,00%	100,00%	100,00%
80	47	0	33	0	100,00%	100,00%	100,00%	100,00%
100	61	0	39	0	100,00%	100,00%	100,00%	100,00%
120	70	0	50	1	98,59%	100,00%	99,17%	99,29%
140	63	0	77	0	100,00%	100,00%	100,00%	100,00%
160	91	0	69	0	100,00%	100,00%	100,00%	100,00%
180	105	1	75	0	100,00%	99,06%	99,45%	99,53%
200	81	0	119	0	100,00%	100,00%	100,00%	100,00%
220	118	0	102	0	100,00%	100,00%	100,00%	100,00%
240	129	0	111	0	100,00%	100,00%	100,00%	100,00%
260	137	0	123	0	100,00%	100,00%	100,00%	100,00%
280	155	0	125	0	100,00%	100,00%	100,00%	100,00%
300	120	0	180	0	100,00%	100,00%	100,00%	100,00%
320	111	0	209	0	100,00%	100,00%	100,00%	100,00%
340	190	0	150	1	99,48%	100,00%	99,71%	99,74%
360	203	0	157	0	100,00%	100,00%	100,00%	100,00%
380	205	0	175	0	100,00%	100,00%	100,00%	100,00%
400	136	1	264	0	100,00%	99,27%	99,75%	99,63%
420	144	0	276	0	100,00%	100,00%	100,00%	100,00%
440	173	6	267	0	100,00%	96,65%	98,65%	98,30%
460	147	0	313	0	100,00%	100,00%	100,00%	100,00%
480	275	3	205	0	100,00%	98,92%	99,38%	99,46%
500	259	2	241	0	100,00%	99,23%	99,60%	99,62%

The results of the second case study were no different than the first one. **Table 6.10** contains the confusion matrix of the three scenarios of each iteration. Only two false positives have been recorded in the first scenarios of the 5th and the 6th iterations and one false negative in the 3rd scenario of the 6th iteration. All the other outcomes were predicted correctly. The performance measurements were calculated and presented in **Table 6.11** using the previous confusion matrices. The results are very promising showing high scores for the performance metrics in all the scenarios. These high scores do not mean that the predicting system is perfect or it is never wrong. But it means that this problem is very adequate for machine learning solutions with selecting the best model, the right attributes and a real training set gathered from experience.

Table 6.10 Confusion matrix for each scenario for the second case study

Number Of CFPs	(1^{st}, 2^{nd}, 3^{rd}) Scenarios of the proposed approach			
	Predicted to succeed CFP_{D1}		Predicted to fail CFP_{D0}	
	Correct (TP)	Incorrect (FP)	Correct (TN)	Incorrect (FN)
10	(7,7,7)	(0,0,0)	(3,3,3)	(0,0,0)
20	(13,12,13)	(0,0,0)	(7,8,7)	(0,0,0)
30	(20,21,16)	(0,0,0)	(10,9,14)	(0,0,0)
40	(23,23,25)	(0,0,0)	(17,17,15)	(0,0,0)
50	(30,26,31)	(1,0,0)	(20,24,19)	(0,0,0)
100	(52,53,58)	(1,0,0)	(48,47,42)	(0,0,1)
150	(82,87,88)	(0,0,0)	(68,63,62)	(0,0,0)
200	(105,122,111)	(0,0,0)	(95,78,89)	(0,0,0)

Table 6.11 Performance measures of the proposed approach for the second case study

	(1^{st}, 2^{nd}, 3^{rd}) Scenarios of the proposed approach			
Number of CFPs	**Recall**	**Precision**	**Accuracy**	**F-measure**
10	**(100%,100%,100%)**	**(100%,100%,100%)**	**(100%,100%,100%)**	**(100%,100%,100%)**
20	**(100%,100%,100%)**	**(100%,100%,100%)**	**(100%,100%,100%)**	**(100%,100%,100%)**
30	**(100%,100%,100%)**	**(100%,100%,100%)**	**(100%,100%,100%)**	**(100%,100%,100%)**
40	**(100%,100%,100%)**	**(100%,100%,100%)**	**(100%,100%,100%)**	**(100%,100%,100%)**
50	**(100%,100%,100%)**	**(96,77%,100%,100%)**	**(98,03%,100%,100%)**	**(98,36%,100%,100%)**
100	**(100%,100%,98,30%)**	**(98,11%,100%,100%)**	**(99,00%,100%,99,00%)**	**(99,04%,100%,99,14%)**
150	**(100%,100%,100%)**	**(100%,100%,100%)**	**(100%,100%,100%)**	**(100%,100%,100%)**
200	**(100%,100%,100%)**	**(100%,100%,100%)**	**(100%,100%,100%)**	**(100%,100%,100%)**

Under the definitions of overfitting and underfitting presented previously, it is clear that the selected logistic regression model is far from being under fitted because it did very well with the training data, and far from being overfitted as well because it performed very well on the unseen data like it did on the training data. The 10 fold cross-validation is one of many methods that can detect underfitting and overfitting problems. Therefore, all the used classifiers during the validation phase which have good scores are neither overfitted nor underfitted. Besides, the proposed approach allows the participant agent to update its data set overtime so that the used model is always fitted and updated.

5 Conclusion

Interaction protocols are widely used in multi-agent systems. Their standard implementations are not perfect and needs to be extended in many situations. A problem related to initiations' deadlines has been presented, which decreases agent's sociability, wastes valuable interactions and can lead to non-optimal results when the system scales up. This problem did not get any attention from researchers despite its importance.

We presented, in this chapter, a novel approach to enhance agent's sociability by extending its interactions with machine learning. The responder agent will be able to decide if an interaction initiation has to be answered immediately or postponed based on its interactions' history. Several interactions' scenarios on two case studies have been performed where different machine algorithms have been validated using the 10 fold cross-validation method. The obtained results were very promising where the number of successful interactions for the evaluated participant agent has been increased significantly from 225% up to 3733% in the first case study and from 18% to 2800% in the second one. Furthermore, the selected classification model has performed very well during all the scenarios in terms of precision, recall and accuracy.

Conclusion and Perspectives

Conclusion and Perspectives

This thesis presents an important challenge in the context of agent oriented software engineering. It is about proposing new perfective maintenance approaches of multi-agent systems. Many agent platforms and tools are available for the agent community. JADE platform is considered to be the most used one; it is well documented and supported. This platform was selected for performance and scalability evaluation. The first encountered problem in this study is that a performance issue can only be related to a given application or a case study due to bad design/code (Lee et al., 1998). This means that the perfective maintenance cannot be generalized or reused for other applications of the same agent platform. Therefore, the idea to address more applications is to target the agent platform's API, known protocols and design patterns. The API offers implementations to various agent concerns in the form of design or code patterns which are reused in many applications. The second encountered challenge is when the evaluation process detects a performance problem in a given component. In this case, the evaluations require a very long time to be performed which can be very exhaustive and time consuming. The third encountered problem is not to only propose a perfective maintenance approach to improve the miss behaving component but also an approach which is easy to implement, reuse and does not negatively affect other systems' components.

It is important to be sure that a performance or scalability problem in a given agent or MAS property is not related to unoptimized code which may contain high fan-in, duplicated code, etc. Reverse-engineering is a very important step in software perfective maintenance to correctly detect the problem cause.

As a background for this thesis, the multi-agent systems and the software maintenance concepts were presented. They are both large fields and only relevant concepts were discussed. The focus was on design patterns of the Yellow Pages service and interaction protocols of JADE platform. Through JADE's API, developers can easily implement these two concepts. In this study, we have shown that their implementations are not scalable and can cause systems' failures. JADE developers have omitted these problems, and just imposed timeout and capacity limits to alert and avoid possible system crashes.

In our proposals, we have relied on aspect-oriented programming and AspectJ tool to enhance the matchmaker service in JADE applications. AspectJ allows us to redesign the matchmaker service by replicating the JADE's DF and distributes the load on them without changing the original source code. The machine learning and Weka toolkit was used to enhance agents' sociability by extending them with learning capabilities.

We have presented the state of the art related to the matchmaking service and to the Contract-net protocol. We have discussed the proposed approaches to enhance these two patterns and how our contributions differ from them.

Our first contribution deals with the performance problems of the Yellow Pages service which is implemented as a centralized service that may suffer from bottlenecks when too much search requests are received. Furthermore, profiling this service on different execution environments showed that even when the DF agent struggles to respond to search requests, the CPU wasn't loaded to its full capacity. This means that the service was not designed to exploit the full potential of the current multi-core CPUs. The proposed approach is a load balancing approach managed by aspect-oriented programming. This programming paradigm offers the mechanisms to implement the enhancements in a separate module called aspect which can alter and replace functions during runtime. The results are very promising and show a great increase in Yellow Pages service performance.

Our second contribution deals with agents' sociability during interaction protocols. We showed that participants were implemented to respond to CFPs based on their receiving order. A good design of interaction protocols, from the initiator side, is to precise a deadline at which the proposals' evaluation begins. Therefore, in loaded systems where the participant receives multiple CFPs, it can waste proposing and responding times on three CFPs types: (1) without deadline, (2) with an expired deadline, or (3) with a deadline that is going to be expired before the answer reaches the initiator. These wasted times will propagate leading to more interactions' failures. If the agent cannot initiate interactions then its sociability will be decreased which will negatively affect its goals. Our objective is to order these CFPs by setting three priorities. The first priority of CFPs should be those which their replies have a high chance to reach the initiator before the deadline. The second priority of CFPs should be those which their replies have low chance to reach the initiator before the deadline. The Last priority of CFPs is for the ones without deadline. The proposed approach extends the participant with learning capabilities using Weka libraries to help it order CFPs in the first or the second priority. This approach increased the agent's sociability significantly and allowed it to successfully initiate more interactions.

Future Work

Further studies will be done to enhance our first contribution by:

1. Proposing a monitor to detect any degradation in the performance and scalability of the Yellow Pages service during runtime, for starting and stopping the DF enhancer automatically.
2. Proposing an approach to identify automatically the optimum number of Yellow Pages replicates needed to enhance the service, which can be changed during runtime depending on the service performance.
3. Reducing the memory space after replicating Yellow Pages, especially when the number of registered services is high, by dividing the services on the DF agents and connecting them together, so that the request gets passed to another DF if no results were found by the previous one.
4. Adapting the approach for external DF requests in multi-host systems.
5. Developing an eclipse plug-in which facilitates this maintenance process.

As future work for the second contribution, we plan to:

1. Incorporate more participants to evaluate if the learning phase has an impact on the system's performance,
2. Evaluate the proposed approach on a distributed example through local networks and internet.
3. Coding the proposed approach in one separate module using aspect-oriented programming for a better reuse and maintenance.

References

Abdul Hameed, O., Younus, A., Hassan Abbas, R., 2019. Aspect oriented programming: Concepts, characteristics and implementation 7, 2022–2033. https://doi.org/10.21533/pen.v7i4

Aguirre, C., Aguirre, C.M. de, Martínez, J.S., Huerta, R., 2008. A low cost / high performance Scalable Topology for Multi-Agent Collaboration.

Aguirre, C., Huerta, R., Noz, J.M., Huerta, R.O., 2000. A low cost/high performance Scalable Topology for Multi-Agent Collaboration, in: Autonomous Agents 2000 Workshop, Agents in Industry.

Ahmad, H.F., 2002. Multi-agent systems: overview of a new paradigm for distributed systems, in: 7th IEEE International Symposium on High Assurance Systems Engineering, 2002. Proceedings. Presented at the 7th IEEE International Symposium on High Assurance Systems Engineering, 2002. Proceedings., pp. 101–107. https://doi.org/10.1109/HASE.2002.1173110

Akinsola, J.E.T., 2017. Supervised Machine Learning Algorithms: Classification and Comparison. International Journal of Computer Trends and Technology (IJCTT) 48, 128–138. https://doi.org/10.14445/22312803/IJCTT-V48P126

Aknine, S., Pinson, S., Shakun, M.F., 2004. An Extended Multi-Agent Negotiation Protocol. Autonomous Agents and Multi-Agent Systems 8, 5–45. https://doi.org/10.1023/B:AGNT.0000009409.19387.f8

Alarabiat, Z., Alarabeyyat, A., Alheyasat, O., 2014. Development of FIPA english auction interaction protocol for multi-agent systems. ces 7, 1905–1917. https://doi.org/10.12988/ces.2014.411237

Ali, A., Aslam, M., Janjua, J.I., Chaudhry, M.U., 2017. Methodology for Performance Evaluation of Distributed Multi Agent System. The Nucleus 54, 75–82.

Alkhatib, G., 1992. The Maintenance Problem of Application Software: An Empirical Analysis. Journal of Software Maintenance 4, 83–104. https://doi.org/10.1002/smr.4360040203

Alonso, F., Fuertes, J.L., Martínez, L., Soza, H., 2008. Measuring the Social Ability of Software Agents, in: 2008 Sixth International Conference on Software Engineering Research, Management and Applications. Presented at the 2008 Sixth International Conference on Software Engineering Research, Management and Applications, pp. 3–10. https://doi.org/10.1109/SERA.2008.32

Al-Yaseen, W.L., Othman, Z.A., Nazri, M.Z.A., 2016. Exchanging large data object in multi-agent systems. p. 020018. https://doi.org/10.1063/1.4960858

Aridor, Y., Lange, D.B., 1998. Agent design patterns: elements of agent application design, in: Proceedings of the Second International Conference on Autonomous Agents, AGENTS '98. Association for Computing Machinery, New York, NY, USA, pp. 108–115. https://doi.org/10.1145/280765.280784

Arnold, R.S., 1993. Software Reengineering. IEEE Computer Society Press.

Arnold, R.S., 1989. Software restructuring. Proceedings of the IEEE 77, 607–617. https://doi.org/10.1109/5.24146

Babczyński, T., Kruczkiewicz, Z., Magott, J., 2005. Performance Comparison of Multi-agent Systems, in: Pěchouček, M., Petta, P., Varga, L.Z. (Eds.), Multi-Agent Systems and

Applications IV, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 612–615. https://doi.org/10.1007/11559221_73

Babczyński, T., Kruczkiewicz, Z., Magott, J., 2004. Performance Analysis Of Multiagent Industrial System, in: Klusch, M., Ossowski, S., Kashyap, V., Unland, R. (Eds.), Cooperative Information Agents VIII, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 242–256. https://doi.org/10.1007/978-3-540-30104-2_18

Babczyński, T., Magott, J., 2006. PERT Based Approach to Performance Analysis of Multi–Agent Systems, in: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Żurada, J.M. (Eds.), Artificial Intelligence and Soft Computing – ICAISC 2006, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 1040–1049. https://doi.org/10.1007/11785231_109

Baker, B.S., 1993. A program for identifying duplicated code. Computing Science and Statistics 49–49.

Baldoni, M., Baroglio, C., Capuzzimati, F., 2013. 2COMM: A Commitment-Based MAS Architecture, in: Cossentino, M., El Fallah Seghrouchni, A., Winikoff, M. (Eds.), Engineering Multi-Agent Systems, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 38–57. https://doi.org/10.1007/978-3-642-45343-4_3

Basili, V.R., 1990. Viewing Maintenance as Reuse-Oriented Software Development. IEEE Softw. 7, 19–25. https://doi.org/10.1109/52.43045

Bates, J., 1994. The role of emotion in believable agents. Commun. ACM 37, 122–125. https://doi.org/10.1145/176789.176803

Bates, J., Loyall, A.B., Reilly, W.S., 1992. An Architecture for Action, Emotion, and Social Behavior, in: Selected Papers from the 4th European Workshop on on Modelling Autonomous Agents in a Multi-Agent World, Artificial Social Systems, MAAMAW ’92. Springer-Verlag, Berlin, Heidelberg, pp. 55–68.

Bellifemine, F., Poggi, A., Rimassa, G., 2000. Developing multi-agent systems with JADE, in: International Workshop on Agent Theories, Architectures, and Languages. Springer, ATAL 2000 Boston, MA, USA, pp. 89–103.

Bellifemine, F., Rimassa, G., Poggi, A., 2001. Developing multi-agent systems with a FIPA-compliant agent framework. Softw. Pract. Exper. 31, 103–128.

Bellman, R., 1961. Adaptive Control Processes: A Guided Tour. Princeton University Press.

Bensaid, N., Mathieu, P., 1997. MAGIQUE: A Hybrid and Hierarchical Multi-Agent Architecture Model.

Bergenti, F., Caire, G., Monica, S., Poggi, A., 2020. The first twenty years of agent-based software development with JADE. Auton Agent Multi-Agent Syst 34, 36. https://doi.org/10.1007/s10458-020-09460-z

Beyer, B., Jones, C., Petoff, J., Murphy, N.R., 2016. Site Reliability Engineering: How Google Runs Production Systems, 1st ed. O’Reilly Media, Inc.

Boehm, B.W., 1983. The Economics of Software Maintenance [WWW Document]. URL https://www.semanticscholar.org/paper/The-Economics-of-Software-Maintenance-Boehm/eaabce709054bd6e9dd48be3440c1fc6e31d43ac (accessed 5.15.20).

Boehm, B.W., others, 1981. Software engineering economics. Prentice-hall Englewood Cliffs (NJ).

Bordini, R.H., Hübner, J.F., 2005. BDI agent programming in agentspeak using Jason, in: Proceedings of the 6th International Conference on Computational Logic in Multi-Agent Systems, CLIMA’05. Springer-Verlag, London, UK, pp. 143–164. https://doi.org/10.1007/11750734_9

Botía, J.A., Hernansáez, J.M., Skarmeta, F.G., 2004. Towards an Approach for Debugging MAS Through the Analysis of ACL Messages, in: Lindemann, G., Denzinger, J., Timm, I.J., Unland, R. (Eds.), Multiagent System Technologies, Lecture Notes in

Computer Science. Springer, Berlin, Heidelberg, pp. 301–312. https://doi.org/10.1007/978-3-540-30082-3_22
Bouckaert, R.R., 2008. Bayesian Network Classifiers in Weka for Version 3-5-7 47.
Boukredera, D., Aknine, S., Maamri, R., 2012. Modeling Temporal Aspects of Contract Net Protocol Using Timed Colored Petri Nets, in: STAIRS. https://doi.org/10.3233/978-1-61499-096-3-83
Bourque, P., Robert, F., Lavoie, J.M., Lee, A., Trudel, S., Lethbridge, T.C., 2001. Guide to the Software Engineering Body of Knowledge (SWEBOK) and the Software Engineering Education Knowledge (SEEK) - a preliminary mapping. 10th International Workshop on Software Technology and Engineering Practice. https://doi.org/10.1109/STEP.2002.1267595
Braubach, L., Pokahr, A., Lamersdorf, W., 2004. Jadex: A Short Overview.
Brazier, F.M.T., Steen, M. van, Wijngaards, N.J.E., Brazier, F., 2001. On MAS Scalability.
Breiman, L., 2001. Random Forests. Machine Learning 45, 5–32. https://doi.org/10.1023/A:1010933404324
Burch, J.G., Grupe, F.H., n.d. A Systems Approach to Software Maintenance [WWW Document]. URL http://www.ittoday.info/AIMS/Information_Management/4-05-10.pdf (accessed 9.7.20).
Camacho, D., Aler, R., 2005. Software and Performance Measures for Evaluating Multi-Agent Frameworks. Applied Artificial Intelligence 19, 645–657. https://doi.org/10.1080/08839510590962050
Cao, J., Kerbyson, D.J., Nudd, G.R., 2001. High performance service discovery in large-scale multi-agent and mobile-agent systems. International Journal of Software Engineering and Knowledge Engineering 11, 621–641.
Capretz, L.F., Capretz, M.A.M., 1996. Object-Oriented Software: Design and Maintenance. WORLD SCIENTIFIC.
Chebout, M.S., Mokhati, F., Badri, M., Babahenini, M.C., 2019. Monitoring open multi-agent systems: An aspect-oriented programming based approach. Multiagent and Grid Systems 15, 155–177. https://doi.org/10.3233/MGS-190307
Chikofsky, E.J., Cross, J.H., 1990. Reverse engineering and design recovery: a taxonomy. IEEE Software 7, 13–17. https://doi.org/10.1109/52.43044
Cho, E., 2006. Jprofiler: Code coverage analysis tool for omp project. Technical Report: CMU 17–654 & 17, Tech. Rep.
Collier, R., 2007. Debugging Agents in Agent Factory, in: Bordini, R.H., Dastani, M., Dix, J., Seghrouchni, A.E.F. (Eds.), Programming Multi-Agent Systems, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 229–248. https://doi.org/10.1007/978-3-540-71956-4_14
Cortellessa, V., Di Marco, A., Inverardi, P., 2011. What Is Software Performance?, in: Cortellessa, V., Di Marco, A., Inverardi, P. (Eds.), Model-Based Software Performance Analysis. Springer, Berlin, Heidelberg, pp. 1–7. https://doi.org/10.1007/978-3-642-13621-4_1
Cortese, E., Quarta, F., Vitaglione, G., Vrba, P., 2002. Scalability and performance of jade message transport system, in: AAMAS Workshop on Agentcities. Bologna, p. 28.
Cosio, C.G., Magariño, I.G., 2009. A metrics suite for the communication of multi-agent systems. Journal of Physical Agents 3, 7–14. https://doi.org/10.14198/JoPha.2009.3.2.03
Counet, A., 2007. Am´elioration du processus de la maintenance du logiciel par un syst`eme informatis´e d'aide `a la d´ecision 162.

Dam, H.K., Zhang, T., Ghose, A., 2013. Improving the reactivity of BDI agent programs, in: International Conference on Principles and Practice of Multi-Agent Systems. Springer, Dunedin, New Zealand, pp. 85–100.

Del Val, E., Rebollo, M., Botti, V., 2014. Enhancing decentralized service discovery in open service-oriented multi-agent systems. Autonomous agents and multi-agent systems 1–30.

DeLoach, S.A., 1999. Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems.

Demazeau, Y., 2001. Voyelles. Mémoire d'habilitation à diriger des recherches, INP Grenoble 103.

Demazeau, Y., 1995. From Interactions To Collective Behaviour In Agent-Based Systems, in: In: Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo. pp. 117–132.

Demeyer, S., Ducasse, S., Nierstrasz, O., Johnson, R.E., 2002. Object Oriented Reengineering Patterns. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Demšar, J., Curk, T., Erjavec, A., Gorup, Č., Hočevar, T., Milutinovič, M., Možina, M., Polajnar, M., Toplak, M., Starič, A., Štajdohar, M., Umek, L., Žagar, L., Žbontar, J., Žitnik, M., Zupan, B., 2013. Orange: data mining toolbox in python. J. Mach. Learn. Res. 14, 2349–2353.

Deshpande, U., Gupta, A., Basu, A., 2005. Performance Enhancement of a Contract Net Protocol Based System Through Instance-Based Learning. IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics) 35, 345–358. https://doi.org/10.1109/TSMCB.2004.842256

Deters, R., 2001. Scalable multi-agent systems, in: JGI '01. https://doi.org/10.1145/376656.376851

Dikaiakos, M.D., Samaras, G., 2000. A Performance Analysis Framework for Mobile-Agent Systems, in: Revised Papers from the International Workshop on Infrastructure for Multi-Agent Systems: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems. Springer-Verlag, Berlin, Heidelberg, pp. 180–187.

Do, T.T., Kolp, M., Faulkner, S., 2005. Introspecting Agent-Oriented Design Patterns. https://doi.org/10.1142/9789812775245_0006

Doan Van Bien, D., Lillis, D., Collier, R.W., 2010a. Call Graph Profiling for Multi Agent Systems, in: Dastani, M., El Fallah Segrouchni, A., Leite, J., Torroni, P. (Eds.), Languages, Methodologies, and Development Tools for Multi-Agent Systems, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 153–167. https://doi.org/10.1007/978-3-642-13338-1_9

Doan Van Bien, D., Lillis, D., Collier, R.W., 2010b. Space-Time Diagram Generation for Profiling Multi Agent Systems, in: Braubach, L., Briot, J.-P., Thangarajah, J. (Eds.), Programming Multi-Agent Systems, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 170–184. https://doi.org/10.1007/978-3-642-14843-9_11

Drmota, M., 2009. Random Trees: An Interplay between Combinatorics and Probability, 1st ed. Springer Publishing Company, Incorporated.

Du, M., Sassioui, R., Varisteas, G., Brorsson, M., Cherkaoui, O., 2017. Improving Real-Time Bidding Using a Constrained Markov Decision Process, in: International Conference on Advanced Data Mining and Applications. Springer, pp. 711–726.

Ductor, S., Guessoum, Z., 2018. A coordination mechanism to replicate large-scale multi-agent systems, in: Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems - SEAMS '18. Presented at the the 13th International Conference, ACM Press, Gothenburg, Sweden, pp. 130–136. https://doi.org/10.1145/3194133.3194154

Đurić, B.O., 2017. A Novel Approach to Modelling Distributed Systems: Using Large-Scale Multi-agent Systems. https://doi.org/10.1007/978-3-319-54325-3_10

Eastwood, A., 1993. Firm Fires Shots at Legacy Systems. Computing Canada, 19(2), p. 17.

Erdil, K., Finn, E., Keating, K., Meattle, J., Park, S., Yoon, D., 2003. Software Maintenance as part of the Software Life-Cycle. Tufts University: Department of computer science.

Erlikh, L., 2000. Leveraging legacy system dollars for e-business. IT Professional 2, 17–23. https://doi.org/10.1109/6294.846201

Faccin, J., Nunes, I., 2015. Bdi-agent plan selection based on prediction of plan outcomes, in: 2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT). IEEE, Singapore, pp. 166–173.

Far, B., Wanyama, T., Soueina, S.O., 2006. A Negotiation Model for Large Scale Multi-Agent Systems, in: 2006 IEEE International Conference on Information Reuse Integration. Presented at the 2006 IEEE International Conference on Information Reuse Integration, pp. 589–594. https://doi.org/10.1109/IRI.2006.252480

Ferber, J., 1999. Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, 1st ed. Addison-Wesley Longman Publishing Co., Inc., USA.

Ferber, J., 1995. Les Systèmes multi-agents: vers une intelligence collective.

Ferber, J., Gutknecht, O., 1998. A meta-model for the analysis and design of organizations in multi-agent systems, in: Proceedings International Conference on Multi Agent Systems (Cat. No.98EX160). Presented at the Proceedings International Conference on Multi Agent Systems (Cat. No.98EX160), pp. 128–135. https://doi.org/10.1109/ICMAS.1998.699041

Ferber, J., Gutknecht, O., Michel, F., 2004. From Agents to Organizations: An Organizational View of Multi-agent Systems, in: Giorgini, P., Müller, J.P., Odell, J. (Eds.), Agent-Oriented Software Engineering IV, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 214–230. https://doi.org/10.1007/978-3-540-24620-6_15

Finin, T., Fritzson, R., McKay, D., McEntire, R., 1994. KQML as an agent communication language, in: Proceedings of the Third International Conference on Information and Knowledge Management, CIKM '94. Association for Computing Machinery, Gaithersburg, Maryland, USA, pp. 456–463. https://doi.org/10.1145/191246.191322

FIPA, 2002a. FIPA Communicative Act Library Specification [WWW Document]. URL http://www.fipa.org/specs/fipa00037/SC00037J.html (accessed 7.27.20).

FIPA, 2002b. FIPA Contract Net Interaction Protocol Specification [WWW Document]. URL http://www.fipa.org/specs/fipa00029/SC00029H.html (accessed 7.29.20).

FIPA, 2001. FIPA ACL Message Structure Specification.

Florea, A., Kayser, D., Pentiuc, S., Segrounichi, A.E.F., 2002. Agents Intelligents, Online course, Department of Computer Science "Politehnica" University of Bucharest [WWW Document]. URL http://turing.cs.pub.ro/auf2/ (accessed 8.10.20).

Freund, Y., Schapire, R.E., 1999. Large Margin Classification Using the Perceptron Algorithm. Machine Learning 37, 277–296. https://doi.org/10.1023/A:1007662407062

Galland, S., Gaud, N., Rodriguez, S., Hilaire, V., 2010. Janus: Another Yet General-Purpose Multiagent Platform [WWW Document]. undefined. URL /paper/Janus%3A-Another-Yet-General-Purpose-Multiagent-Galland-Gaud/b9c8d5b6094a70be76b731d65a2bc7126f47914a (accessed 6.15.20).

Galliers, J.R., 1988. A theoretical framework for computer models of cooperative dialogue, acknowledging multiagent conflict (phd). Open University (United Kingdom).

Garcia, A., Chavez, C., Choren, R., 2007. An Aspect-Oriented Modeling Framework for Multi-Agent Systems Design, in: Padgham, L., Zambonelli, F. (Eds.), Agent-Oriented Software Engineering VII, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 35–50. https://doi.org/10.1007/978-3-540-70945-9_3

Genuchten, M.V., 1992. Towards a Software Factory. Springer Netherlands. https://doi.org/10.1007/978-94-011-2486-7

Giovanni, C., 2003. Book Trading | Jade Site, TILab. URL https://jade.tilab.com/documentation/examples/book-trading/ (accessed 5.11.17).

Gómez-Sanz, J.J., Gervais, M.-P., Weiss, G., 2004. A Survey on Agent-Oriented Oriented Software Engineering Research, in: Bergenti, F., Gleizes, M.-P., Zambonelli, F. (Eds.), Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook, Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer US, Boston, MA, pp. 33–62. https://doi.org/10.1007/1-4020-8058-1_4

Gou, Y., Dam, H.K., Ghose, A., 2013. Towards Semantic Merging of Versions of BDI Agent Systems, in: Boella, G., Elkind, E., Savarimuthu, B.T.R., Dignum, F., Purvis, M.K. (Eds.), PRIMA 2013: Principles and Practice of Multi-Agent Systems, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 437–444. https://doi.org/10.1007/978-3-642-44927-7_32

Gradecki, J.D., Lesiecki, N., 2003. Mastering AspectJ: Aspect-Oriented Programming in Java.

Graham, S.L., Kessler, P.B., Mckusick, M.K., 1982. Gprof: A call graph execution profiler. SIGPLAN Not. 17, 120–126. https://doi.org/10.1145/872726.806987

Growin, J., Weyns, D., Holvoet, T., 2014. Design Patterns for Multi-agent Systems: A Systematic Literature Review. Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks 9783642544323, 79–99. https://doi.org/10.1007/978-3-642-54432-3_5

Grubb, P., Takang, A.A., 2003. Software maintenance: concepts and practice. World Scientific, River Edge, N.J.

Gutknecht, O., Ferber, J., 2000. MadKit: a generic multi-agent platform, in: AGENTS ’00. https://doi.org/10.1145/336595.337048

Hallenborg, K., 2004. Core design pattern for efficient multi-agent architecture. International Book Series "Information Science and Computing " pp 29-36.

Han, S., Youn, H.Y., Song, O., 2012. Efficient category-based service discovery on multi-agent platform. Information Systems Frontiers 14, 601–616.

Hao, J., Leung, H., 2012. Maintaining cooperation in homogeneous multi-agent system, in: 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, Seoul, South Korea, pp. 301–306.

Harrington, P., 2012. Machine Learning in Action. Manning Publications Co., USA.

Harrison, R., 1987. Maintenance Giant Sleeps Undisturbed in Federal Data Centers. pp. 81–86.

Hayden, S.C., Carrick, C., Yang, Q., 1999. Architectural Design Patterns for Multiagent Coordination [WWW Document]. undefined. URL /paper/Architectural-Design-Patterns-for-Multiagent-Hayden-Carrick/af871cf5352cb845e62b6b8447dbf1510dfcc0b4 (accessed 8.6.20).

Helsinger, A., Lazarus, R., Wright, W., Zinky, J., 2003. Tools and techniques for performance measurement of large distributed multiagent systems, in: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS ’03. Association for Computing Machinery, Melbourne, Australia, pp. 843–850. https://doi.org/10.1145/860575.860711

Helsinger, A., Thome, M., Wright, T., 2004. Cougaar: a scalable, distributed multi-agent architecture, in: 2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583). Presented at the 2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583), pp. 1910–1917 vol.2. https://doi.org/10.1109/ICSMC.2004.1399959

Holmes, G., Donkin, A., Witten, I.H., 1994. WEKA: a machine learning workbench (Working Paper).

Hsieh, F.-S., 2017. A hybrid and scalable multi-agent approach for patient scheduling based on Petri net models. Applied Intelligence 47, 1068–1086. https://doi.org/10.1007/s10489-017-0935-y

Hudaib, A., Qasem, M.H., Obeid, N., 2018. FIPA-Based Semi-centralized Protocol for Negotiation, in: Silhavy, R., Silhavy, P., Prokopova, Z. (Eds.), Cybernetics Approaches in Intelligent Systems. Springer International Publishing, Cham, pp. 135–149. https://doi.org/10.1007/978-3-319-67618-0_13

Huff, S., 1990. Information systems maintenance. The Business Quarterly, 55, pp. 30–32.

Hussain, S., Shabbir, H., 2008. Directory scalability in multi-agent based systems.

im2008-jade [WWW Document], 2008. URL https://openhub.net/p/im2008-jade (accessed 5.11.17).

ISO, 2006. ISO/IEC 14764:2006 [WWW Document]. ISO. URL https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/90/39064.html (accessed 5.16.20).

ISO/IEC, 2016. ISO/IEC 25023:2016(E) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of system and software product quality, International Standard.

ISO/IEC/IEEE, 2017. ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary. ISO/IEC/IEEE 24765:2017(E) 1–541. https://doi.org/10.1109/IEEESTD.2017.8016712

JADE Support, 2011. R: [jade-develop] performance measurement for mobile agent and rmi [WWW Document]. URL http://jade.tilab.com/pipermail/jade-develop/2011q2/017085.html (accessed 6.26.16).

Jain, A.K., Murty, M.N., Flynn, P.J., 1999. Data clustering: a review. ACM Comput. Surv. 31, 264–323. https://doi.org/10.1145/331499.331504

Jarraya, T., 2006. Réutilisation des protocoles d'interaction et démarche orientée modèles pour le développement multi-agents.

Jarzabek, S., 2007. Effective Software Maintenance and Evolution: A Reuse-Based Approach. Auerbach Publications, Boston, MA, USA.

John, G.H., Langley, P., 1995. Estimating continuous distributions in Bayesian classifiers, in: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, UAI'95. Morgan Kaufmann Publishers Inc., Montréal, Qué, Canada, pp. 338–345.

Joumaa, H., Demazeau, Y., Vincent, J.-M., 2008. Evaluation of Multi-Agent Systems: The case of Interaction, in: 2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications. Presented at the 2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications, pp. 1–6. https://doi.org/10.1109/ICTTA.2008.4530303

Kadera, P., 2014. Performance Models Preventing Multi-Agent Systems from Overloading Computational Resources. Automation, Control and Intelligent Systems 2, 105. https://doi.org/10.11648/j.acis.20140206.12

Kadera, P., Novák, P., 2017. Performance Modeling Extension of Directory Facilitator for Enhancing Communication in FIPA-Compliant Multiagent Systems. IEEE Transactions on Industrial Informatics 13, 688–695.

Kamath, U., Choppella, K., 2017. Mastering Java Machine Learning: A Java developer's guide to implementing machine learning and big data architectures. Packt Publishing.

Kamiya, T., Ohata, F., Kondou, K., Kusumoto, S., Inoue, K., 2001. Maintenance support tools for Java programs: CCFinder and JAAT, in: Proceedings of the 23rd International Conference on Software Engineering, 2001. ICSE 2001. Presented at the Proceedings of the 23rd International Conference on Software Engineering, 2001. ICSE 2001, pp. 837–838. https://doi.org/10.1109/ICSE.2001.919197

Karakostas, B., 2014. A scalable architecture for concurrent online auctions: A SCALABLE ARCHITECTURE FOR CONCURRENT ONLINE AUCTIONS. Concurrency and Computation: Practice and Experience 26, 841–850. https://doi.org/10.1002/cpe.3192

Kargupta, I.H.H., Stafford, B., 1997. Distributed Data Mining Using An Agent Based Architecture, in: KDD 1997.

Kaur, S., Kaur, H., Sehra, S.K., 2013. Modification of Contract Net Protocol (CNP): A Rule-Updation Approach. arXiv preprint arXiv:1312.4259.

Keerthi, S.S., Shevade, S.K., Bhattacharyya, C., Murthy, K.R.K., 2001. Improvements to Platt's SMO Algorithm for SVM Classifier Design. Neural Computation 13, 637–649. https://doi.org/10.1162/089976601300014493

Kensler, J.A., Agah, A., 2008. Neural networks-based adaptive bidding with the contract net protocol in multi-robot systems. Appl Intell 31, 347. https://doi.org/10.1007/s10489-008-0131-1

Khalil, K.M., Abdel-Aziz, M., Nazmy, T.T., Salem, A.-B.M., 2015a. Machine Learning Algorithms for Multi-Agent Systems, in: Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication, IPAC '15. ACM, New York, NY, USA, p. 59:1–59:5. https://doi.org/10.1145/2816839.2816925

Khalil, K.M., Abdel-Aziz, M., Nazmy, T.T., Salem, A.-B.M., 2015b. MLIMAS: A Framework for Machine Learning in Interactive Multi-agent Systems. Procedia Computer Science, International Conference on Communications, management, and Information technology (ICCMIT'2015) 65, 827–835. https://doi.org/10.1016/j.procs.2015.09.035

Khosravifar, B., Gomrokchi, M., Bentahar, J., Thiran, P., 2009. Maintenance-based trust for multi-agent systems, in: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2, AAMAS '09. International Foundation for Autonomous Agents and Multiagent Systems, Budapest, Hungary, pp. 1017–1024.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G., 2001. An Overview of AspectJ, in: Proceedings of the 15th European Conference on Object-Oriented Programming, ECOOP '01. Springer-Verlag, London, UK, UK, pp. 327–353.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., Irwin, J., 1997. Aspect-oriented programming, in: ECOOP'97 — Object-Oriented Programming. Presented at the European Conference on Object-Oriented Programming, Springer, Berlin, Heidelberg, pp. 220–242. https://doi.org/10.1007/BFb0053381

Kim, B., Son, B., Han, S., Youn, H., 2009. Agent platform-based directory facilitator for efficient service discovery, in: International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2009. CyberC'09. IEEE, Zhangijajie, China, pp. 101–107.

Kim, D.U., Lee, G.H., Lee, K.M., Heo, S.P., Choi, K.H., Shin, D.R., 2007. Design and Implementation of Advanced Directory Facilitator for Efficient Searching, in: The 9th International Conference on Advanced Communication Technology. Presented at the The 9th International Conference on Advanced Communication Technology, pp. 403–406. https://doi.org/10.1109/ICACT.2007.358381

Kiselev, I., 2002. Aspect-Oriented Programming with AspectJ. Sams, Indianapolis, Ind.

Knuth, D.E., 1971. An empirical study of FORTRAN programs. Software: Practice and Experience 1, 105–133. https://doi.org/10.1002/spe.4380010203

Kolp, M., Giorgini, P., Mylopoulos, J., 2002. Information systems development through social structures, in: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, SEKE ’02. Association for Computing Machinery, Ischia, Italy, pp. 183–190. https://doi.org/10.1145/568760.568792

Kolp, M., Giorgini, P., Mylopoulos, J., 2001. A Goal-Based Organizational Perspective on Multi-Agent Architectures. pp. 1–3.

Kolp, M., Wautelet, Y., Faulkner, S., 2010. Sociocentric Design of Multiagent Architectures. pp. 425–455.

Koster, M., 2011. Reliable Multi-agent System for a large scale distributed energy trading network. Master thesis 71.

Kraetzschmar, G.K., 1997. Distributed Reason Maintenance for Multiagent Systems. Springer Science & Business Media.

Kuokka, D., Harada, L., 1995. Matchmaking for Information Agents. International Joint Conference on Artificial Intelligence (IJCAI).

Kurdi, H.A., 2013. Review on Aspect Oriented Programming. International Journal of Advanced Computer Science and Applications (IJACSA) 4. https://doi.org/10.14569/IJACSA.2013.040904

Laddad, R., 2003. AspectJ in Action: Practical Aspect-Oriented Programming. Manning Publications Co., USA.

laerd, 2017. Mean, Mode and Median - Measures of Central Tendency - When to use with Different Types of Variable and Skewed Distributions | Laerd Statistics [WWW Document]. URL https://statistics.laerd.com/statistical-guides/measures-central-tendency-mean-mode-median.php (accessed 5.11.17).

Landwehr, N., Hall, M., Frank, E., 2005. Logistic Model Trees. Mach Learn 59, 161–205. https://doi.org/10.1007/s10994-005-0466-3

Le Cessie, S., Van Houwelingen, J.C., 1992. Ridge Estimators in Logistic Regression. Journal of the Royal Statistical Society. Series C (Applied Statistics) 41, 191–201. https://doi.org/10.2307/2347628

Lee, G.H., Lee, S.H., Choi, K.H., Shin, D.R., 2008. Design of CALM-Based Service Discovery System, in: Fourth International Conference on Networked Computing and Advanced Information Management, 2008. NCM ’08. Presented at the Fourth International Conference on Networked Computing and Advanced Information Management, 2008. NCM ’08, pp. 280–285. https://doi.org/10.1109/NCM.2008.235

Lee, L.C., Nwana, H.S., Ndumu, D.T., De Wilde, P., 1998. The stability, scalability and performance of multi-agent systems. BT Technology Journal 16, 94–103.

Lewenberg, Y., 2017. Machine Learning Techniques for MultiAgent Systems 5185–5186.

Li, K., Zhou, T., Liu, B., Li, H., 2018. A multi-agent system for sharing distributed manufacturing resources. Expert Systems with Applications 99, 32–43. https://doi.org/10.1016/j.eswa.2018.01.027

Li, Z., Guo, G., Chen, B., Ma, L., Luo, Y., Qiu, X., 2016. Optimization for Accelerating Large Scale Agent Based Simulation, in: Zhang, L., Song, X., Wu, Y. (Eds.), Theory, Methodology, Tools and Applications for Modeling and Simulation of Complex Systems. Springer Singapore, Singapore, pp. 165–175. https://doi.org/10.1007/978-981-10-2663-8_18

Lientz, B.P., Swanson, E.B., 1981. Problems in Application Software Maintenance. Commun. ACM 24, 763–769. https://doi.org/10.1145/358790.358796

Lientz, B.P., Swanson, E.B., 1980. Software Maintenance Management. Addison-Wesley Longman Publishing Co., Inc., USA.

Maes, P., 1994. Agents that reduce work and information overload. Commun. ACM 37, 30–40. https://doi.org/10.1145/176789.176792

Mahajan, R., Gupta, A., 2008. Scalable Contract Net Based Resource Allocation Strategies for Grids. IEEE, pp. 25–32. https://doi.org/10.1109/PDCAT.2008.42

Mao, H., Zhang, Z., Xiao, Z., Gong, Z., Ni, Y., 2020. Learning multi-agent communication with double attentional deep reinforcement learning. Auton Agent Multi-Agent Syst 34, 32. https://doi.org/10.1007/s10458-020-09455-w

Marin, M., Van Deursen, A., Moonen, L., 2004. Identifying aspects using fan-in analysis, in: 11th Working Conference on Reverse Engineering. IEEE, pp. 132–141.

Marir, T., 2016. Les Systèmes Multi-Agents, course support, Departement of computer science, University of Oum El Bouaghi, Algeria.

Marir, T., 2015. UNE DEMARCHE D'ASSURANCE QUALITE POUR LES SYSTEMES MULTIAGENTS. ANNABA.

Marir, T., Mokhati, F., Bouchlaghem-Seridi, H., Acid, Y., Bouzid, M., 2016. QM4MAS: a quality model for multi-agent systems. International Journal of Computer Applications in Technology.

Martin, J., McClure, C.L., 1983. Software Maintenance: The Problem and Its Solutions. Prentice-Hall.

McKee, J., 1984. Maintenance as a function of design. Proceedings of the AFIPS National Computer Conference, Las Vegas, pp. 187–193.

Mengistu, D., Tröger, P., Lundberg, L., Davidsson, P., 2008. Scalability in distributed multi-agent based simulations: The jade case, in: Second International Conference on Future Generation Communication and Networking Symposia, 2008. FGCNS'08. IEEE, Sanya, China, pp. 93–99.

Mens, K., 2017. Software Maintenance and Evolution. https://doi.org/10.13140/RG.2.2.11985.33123

Mens, T., Demeyer, S. (Eds.), 2008. Software Evolution. Springer-Verlag, Berlin Heidelberg. https://doi.org/10.1007/978-3-540-76440-3

Mertens, K., Holvoet, T., Berbers, Y., Distrinet, 2004. Adaptation in a distributed environment.

Meyer, J.-J.Ch., 2005. Programming Cognitive Agents, in: Eymann, T., Klügl, F., Lamersdorf, W., Klusch, M., Huhns, M.N. (Eds.), Multiagent System Technologies, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 27–34. https://doi.org/10.1007/11550648_3

Mihaylov, M., Razo-Zapata, I., Rădulescu, R., Jurado, S., Avellana, N., Nowé, A., 2016. Smart Grid Demonstration Platform for Renewable Energy Exchange, in: Demazeau, Y., Ito, T., Bajo, J., Escalona, M.J. (Eds.), Advances in Practical Applications of Scalable Multi-Agent Systems. The PAAMS Collection, Lecture Notes in Computer Science. Springer International Publishing, Cham, pp. 277–280. https://doi.org/10.1007/978-3-319-39324-7_30

Miles, R., 2004. AspectJ Cookbook. O'Reilly Media, Inc.

Mishra, A., 2020. Machine Learning for iOS Developers, 1st ed. Wiley. https://doi.org/10.1002/9781119602927

Moad, J., 1990. Maintaining the Competitive Edge. Datamation, p. 61,62,64,66.

Mueller, J.P., Massaron, L., 2016. Machine Learning For Dummies, 1 edition. ed. For Dummies, Hoboken, New Jersey.

Muscat, S., 2003. Enhancing distributed systems through web services.

Nagi, K., 2001. Scalability of a Transactional Infrastructure for Multi-Agent Systems, in: Wagner, T., Rana, O.F. (Eds.), Infrastructure for Agents, Multi-Agent Systems, and

Scalable Multi-Agent Systems, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 266–278. https://doi.org/10.1007/3-540-47772-1_27
Nagwani, N.K., 2009. Performance measurement analysis for multi-agent systems, in: 2009 International Conference on Intelligent Agent Multi-Agent Systems. Presented at the 2009 International Conference on Intelligent Agent Multi-Agent Systems, pp. 1–4. https://doi.org/10.1109/IAMA.2009.5228092
Neuman, B.C., 1994. Scale in Distributed Systems, in: Readings in Distributed Computing Systems. IEEE Computer Society Press, pp. 463–489.
Nguyen, P.T., Schau, V., Rossak, W., 2011a. Performance Comparison of some Message Transport Protocol Implementations for Agent Community Communication., in: IICS. pp. 193–204.
Nguyen, P.T., Schau, V., Rossak, W., 2011b. Performance Comparison of some Message Transport Protocol Implementations for Agent Community Communication., in: IICS. pp. 193–204.
Nguyen, T.T., Nguyen, N.D., Nahavandi, S., 2018. Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications. arXiv:1812.11794 [cs, stat].
Ning, J.Q., 1996. A Component-Based Software Development Model, in: Proceedings of the 20th Conference on Computer Software and Applications, COMPSAC ’96. IEEE Computer Society, USA, p. 389.
Nwana, H., Ndumu, D.T., Lee, L.C., Heath, M., 1998. ZEUS: An Advanced Tool-Kit for Engineering Distributed Multi-Agent Systems.
O’Brien, P.D., Nicol, R.C., 1998. FIPA — Towards a Standard for Software Agents. BT Technology Journal 16, 51–59. https://doi.org/10.1023/A:1009621729979
Odell, J.J., Van Dyke Parunak, H., Fleischer, M., Brueckner, S., 2003. Modeling Agents and Their Environment, in: Giunchiglia, F., Odell, J., Weiß, G. (Eds.), Agent-Oriented Software Engineering III, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 16–31. https://doi.org/10.1007/3-540-36540-0_2
Oracle, J., n.d. JDK 14 Documentation [WWW Document]. Oracle Help Center. URL https://docs.oracle.com/en/java/javase/14/ (accessed 5.16.20).
Osborne, W.M., 1987. Building and Sustaining Software Maintainability, in: Proceedings of the International Conference on Software Maintenance. pp. 13–23.
Othman, O., Schmidt, D.C., 2001. Optimizing Distributed System Performance via Adaptive Middleware Load Balancing.
Parikh, G., 1986. Handbook of Software Maintenance. John Wiley & Sons, Inc., USA.
Pascoa, F., Pereira, I., Ferreira, P., Lohse, N., 2017. Redundant and decentralised directory facilitator for resilient plug and produce cyber physical production systems.
Platt, J.C., 1999. Fast training of support vector machines using sequential minimal optimization, in: Advances in Kernel Methods: Support Vector Learning. MIT Press, Cambridge, MA, USA, pp. 185–208.
Port, O., 1998. The Software Trap - Automate or Else. Business Week, 3051(9), pp. 142–154.
Raschka, S., 2018. STAT 479: Machine Learning Lecture Notes.
Raschka, S., Mirjalili, V., 2017. Python Machine Learning - Second Edition: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow, 2nd edition. ed. Packt Publishing, Birmingham Mumbai.
Rather, A., Bhatnagar, V., 2016. Study of Software Development Using Software Re-Engineering. IJETA 3, 52–56.
Rens, G., Moodley, D., 2017. A hybrid POMDP-BDI agent architecture with online stochastic planning and plan caching. Cognitive Systems Research 43, 1–20.

Retaillé, J.-P., Pawlak, R., Seinturier, L., 2004. La programmation orientée aspect pour Java/J2EE.

Reuss, P., Althoff, K.-D., 2014. Maintenance of Distributed Case-Based Reasoning Systems in a Multi-Agent System., in: LWA. pp. 20–30.

Rimassa, G., Calisti, M., Kernland, M.E., 2005. Living Systems® Technology Suite, in: Unland, R., Calisti, M., Klusch, M. (Eds.), Software Agent-Based Applications, Platforms and Development Kits, Whitestein Series in Software Agent Technologies. Birkhäuser, Basel, pp. 73–93. https://doi.org/10.1007/3-7643-7348-2_4

Rine, D., 1993. Software perfective maintenance: Including retrainable software in software reuse. Information Sciences 75, 109–132. https://doi.org/10.1016/0020-0255(93)90116-4

Ripeanu, M., 2001. Peer-to-peer architecture case study: Gnutella network, in: Proceedings First International Conference on Peer-to-Peer Computing. Presented at the Proceedings First International Conference on Peer-to-Peer Computing, pp. 99–100. https://doi.org/10.1109/P2P.2001.990433

Robbes, R., Bouraqadi, N., Stinckwich, S., 2004. An Aspect-based Multi-Agent System.

Rosenschein, S.J., 1985. Formal theories of knowledge in AI and robotics. NGCO 3, 345–357. https://doi.org/10.1007/BF03037076

Ruder, S., 2016. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

Russell, S., Norvig, P., 2009. Artificial Intelligence: A Modern Approach, 3rd ed. Prentice Hall Press, USA.

Savall, M.H., 2003. Une Architecture d'agents pour la simulation : le modèle YAMAM et sa plate-forme Phoenix.

Scerri, P., Vincent, R., Mailler, R. (Eds.), 2006. Coordination of Large-Scale Multiagent Systems. Springer-Verlag, New York. https://doi.org/10.1007/0-387-27972-5

Schatten, M., Ševa, J., Tomičić, I., 2016. A roadmap for scalable agent organizations in the Internet of Everything. Journal of Systems and Software 115, 31–41. https://doi.org/10.1016/j.jss.2016.01.022

Schatten, M., Tomicic, I., Đurić, B.O., 2017. A Review on Application Domains of Large-Scale Multiagent Systems.

Schatten, M., Tomičić, I., Đurić, B.O., 2015. Multi-agent modeling methods for massivley Multi-Player On-Line Role-Playing Games, in: 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). Presented at the 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1256–1261. https://doi.org/10.1109/MIPRO.2015.7160468

Schillo, M., Fischer, K., Knabe, T., 2001. The contract-net with confirmation protocol : an improved mechanism for task assignment. https://doi.org/10.22028/D291-25174

Schneidewind, N.F., 1987. The State of Software Maintenance. IEEE Transactions on Software Engineering SE-13, 303–310. https://doi.org/10.1109/TSE.1987.233161

Schult, W., Polze, A., 2002. Aspect-oriented programming with C# and .NET, in: Proceedings Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. ISIRC 2002. Presented at the Proceedings Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. ISIRC 2002, pp. 241–248. https://doi.org/10.1109/ISORC.2002.1003711

Seah, C., Sierhuis, M., Clancey, W.J., 2005. Multi-agent modeling and simulation approach for design and analysis of MER mission operations.

Searle, J.R., 1969. Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9781139173438

Secq, Y., 2003. Une méthodologie pour les systèmes Multi-agent ouverts, PhD Thesis. University of sciences and technologies, Lille - France.

Seliya, N., Khoshgoftaar, T.M., Hulse, J.V., 2009. A Study on the Relationships of Classifier Performance Metrics, in: 2009 21st IEEE International Conference on Tools with Artificial Intelligence. Presented at the 2009 21st IEEE International Conference on Tools with Artificial Intelligence, pp. 59–66. https://doi.org/10.1109/ICTAI.2009.25

Seungwok Han, Sung Keun Song, Hee Yong Youn, 2006. CALM: an intelligent agent-based middleware for community computing, in: The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06). Presented at the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06), p. 6 pp.-. https://doi.org/10.1109/SEUS-WCCIA.2006.20

Shahid, H., Hassan, S., 2008. Directory scalability in multi-agent based systems (Master thesis in software engineering, N° MSE-2008-22). School of Engineering Blekinge Institute of Technoloy, Sweden.

Shen, W., Norrie, D.H., 1999. Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey. Knowl. Inf. Syst. 1, 129–156. https://doi.org/10.1007/BF03325096

Shendarkar, A., Vasudevan, K., Lee, S., Son, Y.J., 2008. Crowd simulation for emergency response using BDI agents based on immersive virtual reality. Simulation Modelling Practice and Theory 16, 1415–1429. https://doi.org/10.1016/j.simpat.2008.07.004

Shoham, Y., 1993. Agent-oriented programming. Artificial Intelligence 60, 51–92. https://doi.org/10.1016/0004-3702(93)90034-9

Smith, 1980. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. IEEE Transactions on Computers C–29, 1104–1113. https://doi.org/10.1109/TC.1980.1675516

Software Engineering [WWW Document], n.d. . Software Engineering Tutorial - Tutorialspoint. URL https://www.tutorialspoint.com/software_engineering/ (accessed 7.13.20).

Spinczyk, O., Gal, A., Schröder-Preikschat, W., 2002. AspectC++: an aspect-oriented extension to the C++ programming language, in: Proceedings of the Fortieth International Conference on Tools Pacific: Objects for Internet, Mobile and Embedded Applications, CRPIT '02. Australian Computer Society, Inc., AUS, pp. 53–60.

Srivastava, A., Eustace, A., 1994. ATOM: A System for Building Customized Program Analysis Tools. SIGPLAN Not. 29, 196–205. https://doi.org/10.1145/773473.178260

Stone, M., 1974. Cross-Validatory Choice and Assessment of Statistical Predictions. Journal of the Royal Statistical Society. Series B (Methodological) 36, 111–147.

Sumner, M., Frank, E., Hall, M., 2005. Speeding up logistic model tree induction, in: Proceedings of the 9th European Conference on European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, ECMLPKDD'05. Springer-Verlag, Porto, Portugal, pp. 675–683. https://doi.org/10.1007/11564126_72

Tahara, Y., Ohsuga, A., Honiden, S., 1999. Agent system development method based on agent patterns, in: Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002). Presented at the Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002), pp. 356–367.

Tan, W.G., Jarzabek, S., 1992. Current practices and future needs of software maintenance in Singapore. Proceedings of the SCS Silver Jubilee Conference on Software Engineering: New Technologies and Business Payoffs, Singapore, pp. 121–135.

Tranier, J., 2007. Vers une vision intégrale des systèmes multi-agents. Contribution à l'intégration des concepts d'agent, d'environnement, d'organisation et d'institution (PhD Thesis). University of Montpellier 2.

Trejo-Sánchez, J.A., López-Martínez, J.L., García, J.O.G., Ramírez-Pacheco, J.C., Fajardo-Delgado, D., 2018. A multi-agent architecture for scheduling of high performance services in a GPU cluster. International Journal of Combinatorial Optimization Problems and Informatics 9, 12–22.

Tung, T., Manuel, D., Pirotte, K., 2003. Social Patterns for Designing Multiagent Systems.

Turner, P.J., Jennings, N.R., 2001. Improving the Scalability of Multi-agent Systems, in: Wagner, T., Rana, O.F. (Eds.), Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 246–262. https://doi.org/10.1007/3-540-47772-1_25

Tuyls, K., Weiss, G., 2012. Multiagent Learning: Basics, Challenges, and Prospects. AI Magazine 33, 41–52. https://doi.org/10.1609/aimag.v33i3.2426

Uno, K., Kashiyama, K., 2008. Development of Simulation System for the Disaster Evacuation Based on Multi-Agent Model Using GIS. Tsinghua Science & Technology 13, 348–353. https://doi.org/10.1016/S1007-0214(08)70173-1

van Lon, R.R.S., Holvoet, T., 2017. When do agents outperform centralized algorithms?: A systematic empirical evaluation in logistics. Autonomous Agents and Multi-Agent Systems 31, 1578–1609. https://doi.org/10.1007/s10458-017-9371-y

Varga, E., 2017. Unraveling Software Maintenance and Evolution: Thinking Outside the Box. Springer International Publishing. https://doi.org/10.1007/978-3-319-71303-8

Waller, J., 2014. Performance Benchmarking of Application Monitoring Frameworks (PhD Thesis). University of Kiel, Germany.

Wang, T., Li, L., Zhang, S., Peng, H., Yu, L., Chen, Z., 2016. Memory Mechanism Enhances Cooperation in Mobile Multi-agent System, in: 2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC). IEEE, Hangzhou, China, pp. 476–479.

Wang, X., Schulzrinne, H., Kandlur, D., Verma, D., 2008. Measurement and Analysis of LDAP Performance. IEEE/ACM Transactions on Networking 16, 232–243. https://doi.org/10.1109/TNET.2007.911335

Weyns, D., Holvoet, T., Distrinet, A., 2003. Look, Talk and Do: A Synchronization Scheme For Situated Multi-Agent.

Weyns, D., Van Dyke Parunak, H., Michel, F., Holvoet, T., Ferber, J., 2005. Environments for Multiagent Systems State-of-the-Art and Research Challenges, in: Weyns, D., Van Dyke Parunak, H., Michel, F. (Eds.), Environments for Multi-Agent Systems, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 1–47. https://doi.org/10.1007/978-3-540-32259-7_1

White, J.K., 1994. Telescript technology: the foundation for the electronic marketplace. https://doi.org/null

Wilde, P.D., Nwana, H.S., Lee, L., 1999. Stability, fairness and scalability of multi-agent systems.

Woergoetter, F., Porr, B., 2008. Reinforcement learning. Scholarpedia 3, 1448. https://doi.org/10.4249/scholarpedia.1448

Woodside, M., 2001. Scalability Metrics and Analysis of Mobile Agent Systems, in: Wagner, T., Rana, O.F. (Eds.), Infrastructure for Agents, Multi-Agent Systems, and Scalable

Multi-Agent Systems, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 234–245. https://doi.org/10.1007/3-540-47772-1_24

Wooldridge, M., 2002. Intelligent Agents: The Key Concepts, in: Mařík, V., Štěpánková, O., Krautwurmová, H., Luck, M. (Eds.), Multi-Agent Systems and Applications II, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 3–43. https://doi.org/10.1007/3-540-45982-0_1

Wooldridge, M., Jennings, N.R., 1995. Intelligent agents: theory and practice. The Knowledge Engineering Review 10, 115–152. https://doi.org/10.1017/S0269888900008122

Wooldridge, M.J., 2009. An introduction to multiagent systems, 2nd ed. ed. John Wiley & Sons, Chichester, U.K.

Xie, T., Zhao, J., Marinov, D., Notkin, D., 2006. Detecting Redundant Unit Tests for AspectJ Programs, in: 2006 17th International Symposium on Software Reliability Engineering. Presented at the 2006 17th International Symposium on Software Reliability Engineering, pp. 179–190. https://doi.org/10.1109/ISSRE.2006.21

Yang, J., Li, W.-L., Hong, C.-Y., 2008. An Improvement to CNCP in Large-Scale Muti-Agent System, in: Innovative Computing Information and Control, 2008. ICICIC’08. 3rd International Conference On. IEEE, pp. 147–147.

Yao, Y., Logan, B., 2016. Action-level intention selection for BDI agents, in: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems, Singapore, pp. 1227–1236.

Yongchang, R., Tao, X., Zhongjing, L., Xiaoji, C., 2011. Software Maintenance Process Model and Contrastive Analysis, in: 2011 International Conference on Information Management, Innovation Management and Industrial Engineering. Presented at the 2011 International Conference on Information Management, Innovation Management and Industrial Engineering, pp. 169–172. https://doi.org/10.1109/ICIII.2011.324

Yule, G., 1996. Pragmatics. OUP Oxford.

Zambonelli, F., Jennings, N.R., Wooldridge, M., 2003. Developing Multiagent Systems: The Gaia Methodology. ACM Trans. Softw. Eng. Methodol. 12, 317–370. https://doi.org/10.1145/958961.958963

Zerrougui, S., Mokhati, F., Badri, M., 2014. Toward a new aspect-mining approach for multi-agent systems. Journal of Systems and Software 98, 9–24.

Zerrougui, S., Mokhati, F., Badri, M., Babahenini, M.C., 2017. A novel approach for scalability and performance enhancement in JADE. Multiagent and Grid Systems 13, 177–201. https://doi.org/10.3233/MGS-170267

Zhang, F., Cao, J., Liu, L., Wu, C., 2011. Performance Improvement of Distributed Systems by Autotuning of the Configuration Parameters. Tsinghua Science & Technology 16, 440–448. https://doi.org/10.1016/S1007-0214(11)70063-3