QM4MAS: a quality model for multi-agent systems

Toufik Marir, Farid Mokhati, Hassina Bouchlaghem-Seridi, Youghourta Acid, Maroua Bouzid

QM4MAS: a quality model for multi-agent systems

Toufik Marir*

Department of Mathematics and Computer Science,
University of Oum El Bouaghi,
Research Laboratory on Computer Science's
Complex Systems (RELA(CS)2) Laboratory,
B.P. 358 – 04000 Oum El Bouaghi, Algeria
and
Department of Computer Science,
University of Annaba,
BP 12 – 23000 – Annaba, Algeria
Email: marir.toufik@yahoo.fr
*Corresponding author

Farid Mokhati

Department of Mathematics and Computer Science,
University of Oum El Bouaghi,
Research Laboratory on Computer Science's
Complex Systems (RELA(CS)2) Laboratory,
B.P. 358 – 04000 Oum El Bouaghi, Algeria
Email: mokhati@yahoo.fr

Hassina Bouchlaghem-Seridi

Department of Computer Science,
LABGED Laboratory,
University of Annaba,
BP 12 – 23000 – Annaba, Algeria
Email: seridi@labged.net

Youghourta Acid

Department of Mathematics and Computer Science,
University of Oum El Bouaghi,
B.P. 358-04000 Oum El Bouaghi, Algeria
Email: acid_youghourta@hotmail.com

Maroua Bouzid

University of Caen Basse-Normandie,
Campus Côte de Nacre, Boulevard du Maréchal Juin,
BP 5186 – 14032 Caen Cedex, France
Email: maroua.bouzid-mouaddib@unicaen.fr

Abstract: Multi-agent systems (MASs) are being increasingly used in complex and distributed applications development. Such applications should satisfy the requirements of users in terms of quality. Accordingly, it is important to assess the quality of such systems. In fact, several metrics have been proposed to assess different aspects of multi-agent systems. However, the lack of comprehensive quality model for multi-agent applications that combines the software's characteristics with the proposed metrics limits the usefulness of such metrics. In this paper, we propose an overall quality model for multi-agent-based software, called QM4MAS. An overall quality model gives a global view of the quality showing the relationships between its characteristics. The use of QM4MAS has two main objectives: 1) It allows defining and assessing the MAS quality; 2) it facilitates the maintenance of software product (high quality software is easier to maintain). The proposed model has been applied to JADE applications through a set of metrics. The assessment of JADE's proposed metrics can be done automatically using a tool we developed for this purpose.

Keywords: quality assurance; quality model; multi-agent systems; MASs; ISO 9126; JADE platform; dynamic metrics.

Reference to this paper should be made as follows: Marir, T., Mokhati, F., Bouchlaghem-Seridi, H., Acid, Y. and Bouzid, M. (2016) 'QM4MAS: a quality model for multi-agent systems', *Int. J. Computer Applications in Technology*, Vol. 54, No. 4, pp.297–310.

Biographical notes: Toufik Marir is a Lecturer at the Department of Mathematics and Computer Science of The University of Oum El Bouaghi in Algeria. He holds the Magister in Computer Science (Artificial Intelligence) from the University of Khenchela (Algeria) in 2009 and the Doctorate in Science in Computer Science from the University of Annaba (Algeria) in 2015. Currently, he is a member of the team Distributed-Intelligent Systems Engineering (DISE) at the ReLa(CS)2 Laboratory. His main areas of interest include agent-oriented software engineering, software quality and formal methods.

Farid Mokhati is a Professor of Computer Science at the Department of Mathematics and Computer Science of the University of Oum El-Bouaghi in Algeria. He holds a Habilitation à Diriger des Recherches (HDR), in Computer Science (Distributed Artificial Intelligence) from the University of Annaba in Algeria. Currently, he is the Head of the team Distributed-Intelligent Systems Engineering (DISE) at the ReLa(CS)2 Laboratory. His main areas of interest include object and agent-oriented software engineering and formal methods.

Hassina Bouchlaghem-Seridi is a Professor of Computer Science at the Department of Computer Science of the University of Annaba in Algeria. She received her Master's in Computer Science and Engineering from Badji Mokhtar University in 1997 and PhD in Computer Science and Engineering in 2006. Currently, she is the Head of the team 'TEL, Data mining and Web Technology' at LabGED Laboratory. She has broad interests in information system on the web and especially service web, context, semantic, knowledge management, e-learning application, recommender systems and social web.

Youghourta Acid is a Software Development Engineer. She received her Master in Computer Science (Distributed Systems) from the University of Oum El Bouaghi in Algeria. Her main areas of interest include agent-oriented software engineering and quality models.

Maroua Bouzid received her diploma of 'Ingénieur d'état' in Computer Science from the University of Constantine (Algeria) in 1990 and her MSc degree from the University on Nancy 1 (France) in 1991 as well as her PhD degree in 1995 in Temporal Reasoning. She obtained her HDR (habilitation to supervise research) from the University of Caen (France) in 2006 in Spatio-Temporal Reasoning. From 1996 to 2002, she was an Associate Professor at The University of Artois in Lens (France), and from 2002 to 2009, she was an Associate Professor at The University of Caen. Since 2009, she is a Professor at the University of Caen in the Computer Science Department.

1 Introduction

The quality assurance of software products is one of the ultimate goals of software engineering. However, the software quality remains a complex concept. In order to understand and study the complex concepts, we often use models. In the literature, several quality models have been proposed like McCall et al. (1977) model and ISO-9126 model (ISO, 2001). The ISO-9126 standard defined the software quality model as "the set of characteristics and the relationships between them which provide the basis for specifying quality requirements and evaluating quality" (ISO, 2001). Nevertheless, it is important to distinguish the software quality models according to their purposes. So, a software quality model can be used to define, assess and/or predict quality (Deissenboeck et al., 2009). In fact, the prediction is used also for other purposes. As example, the prediction can address the cost (Kaushik et al., 2013), the fault proneness (Singh et al., 2014) or the level of severity faults (Singh et al., 2013).

Since the advent of the first models of software quality before 30 years, software engineering has undergone several evolutions. In fact, several software development paradigms have been proposed. Furthermore, specificities of each software development paradigm require the development of its own quality model. Hence, we can find in the literature several models for specific software paradigms, like object-oriented software (Alonso et al., 1998; Bansiya and Davis, 2002) or service-oriented software (Goeb and Lochmann, 2011).

Undoubtedly, multi-agent paradigm is one of the best and most applied software paradigms for complex and distributed systems development. However, few works have focused on the quality of agent-oriented software. Moreover, the proposed approaches in the quality of agent-oriented software target to develop measurements of some characteristics of agent-oriented software (Dumke et al., 2010). According to Alonso et al. (2009), up till now there is not a comprehensive quality model for agent-oriented software. In order to define and assess multi-agent applications quality, we propose an overall quality model called QM4MAS. The proposed quality model gives a global view of the quality of multi-agent systems (MASs) which allows the presentation of its

multi-dimensional nature. Moreover, applying this model to MAS applications can also facilitate their maintenance process (high quality software is easier to maintain). To validate our quality model, we developed a visual tool allowing assessing JADE applications quality through a set of proposed metrics.

The remainder of this paper is organised as follows: some related works are presented in Section 2 followed by the presentation of our proposed quality model (in Section 3). Section 4 is devoted to present the application of our quality model on JADE applications and the presentation of our developed tool. Section 5 gives some conclusions and future work directions.

2 Related work

Quality assurance is a hard task in all software projects whatever the software development paradigm is. Indeed, the quality models play a central role for understanding and evaluating the software quality. Thus, the hierarchical models, like factors-criteria-metrics (for FCM) model of McCall et al. (1977), is a well accepted technique for modelling the software quality (Lincke and Löwe, 2006).

In the multi-agent paradigm, very few works have proposed specific metrics to assess agent-oriented software quality. In fact, several approaches adapt procedural and object-oriented measures for such a purpose (Alonso et al., 2009). An overview of proposed multi-agent measurements is presented in Dumke et al. (2010).

In order to develop a quality model for agent-oriented software, Alonso et al. (2008, 2009, 2010) proposed a series of works that address different aspects of MASs. First of all, Alonso et al. (2008) identified six characteristics of MASs: social ability, autonomy, pro-activity, reactivity, mobility, intelligence, and adaptability. Then, they decomposed the social ability into three attributes: communication, cooperation and negotiation. For each attribute, the authors proposed a set of metrics. For example, the average message size (AMS) can be used to measure the communication attribute. Indeed, the social ability of MASs, especially the communication, has been targeted in other works like the metrics proposed by Gutiérrez and García-Magariño (2009) for detecting the undesirable communication patterns.

Considering the autonomy as one of the most important features of agents, Alonso et al. (2009) proposed a set of metrics to evaluate agent's autonomy. The authors consider autonomy as a characteristic composed of three attributes: self-control, functional independence and evolution capability. Each attribute is measured using a set of proposed metrics.

García-Magariño et al. (2010) adapted some object-oriented metrics to evaluate certain quality attributes of MAS architectures. Taking inspiration from McCall's et al. (1977) approach, the authors proposed three attributes for architectural design quality: extensibility, modularity and complexity. Obviously, a set of metrics has been proposed to evaluate each attribute. For example, the extensibility attribute can be evaluated using the cohesion and coupling metrics.

According to the above classification of quality models, all the cited approaches have the same purpose: assess the quality of based agent software. However, each approach addressed only a few characteristics of multi-agent paradigm. Despite the importance of using metrics to assess some characteristics of MASs, like communication (Gutiérrez and García-Magariño, 2009) and architectural design (García-Magariño et al., 2010), the lack of an overall quality model for such software limits the utility of the proposed metrics owing to the multi-dimensional nature of the quality concept.

The approaches proposed by Alonso et al. (2008, 2009, 2010) attempt to develop an overall quality model for MASs by examining each characteristic alone then the aggregation of all examined characteristics. However, this method prevents the authors to link the agent paradigm characteristics (like autonomy and social ability) with the high-level software characteristics (such as reliability and efficiency). According to Alonso et al. (1998), and Bansiya and Davis (2002), high-level software characteristics presented in well-known software quality models [like McCall et al. (1977) model and ISO-9126 model] can be reused whatever the software paradigm is. In order to bridge this gap, we present in this paper an overall quality model which links the high-level software characteristics of ISO-9126 model with agent paradigm characteristics.

Because of the diversity of implementation paradigms of MASs (such as object-oriented paradigm and knowledge-based systems, etc.) on the one hand, and their various programming languages on the other hand, we believe that the proposition of a metric must be made by specifying the paradigm or implementation language that supports the proposed metric. In our works, the proposed metrics are specified for the JADE platform.

Table 1 Summary of the main related works

Work	*Examined characteristic*	*Deficiencies*
Alonso et al. (2008)	The social ability	• There is no global view of the quality concept. • The relationships between high-level quality characteristics (like reliability) and the agent characteristics (like autonomy) are not established. • Some metrics are closely depended to the implementation paradigm without specify the specificities of each one. • The measure methods are not specified.
Alonso et al. (2009)	The autonomy	
Alonso et al. (2010)	The pro-activity	
García-Magariño et al. (2010)	The MAS architecture	

Several approaches propose different forms of complexity (structural and behavioural complexity) as metrics of

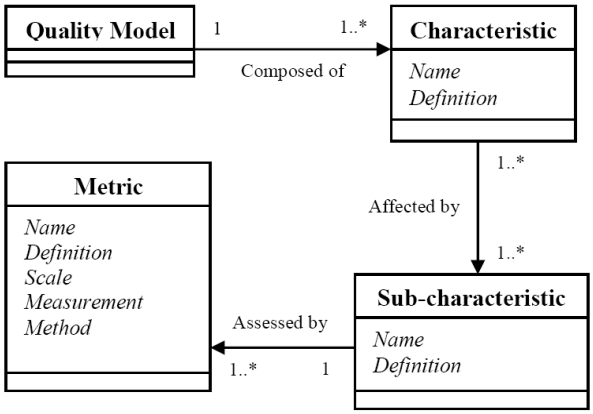
autonomy (Alonso et al., 2009). However, the relationship between autonomy and complexity is not clear. A simple agent can achieve its goal without the intervention of a third party whereas a complex agent may request the assistance of another agent to achieve its objective. Giving a global view of quality, our proposed quality model made clear the relationships between quality characteristics and the agent paradigm characteristics.

Table 1 makes clear the differences between these works and their deficiencies. These deficiencies are common to all the presented works.

3 Quality model for MASs

Hierarchical quality models represent a well-accepted means to understand, define and assess software quality. More than a standard software quality model, ISO-9126 (ISO, 2001) has many advantages compared to other quality models (Behkamal et al., 2009). In fact, several approaches customise the ISO-9126 quality model to support specific software paradigms (Bansiya and Davis, 2002; Lee and Lee, 2006; Behkamal et al., 2009). The proposed model, called QM4MAS, is an extension of ISO-9126 quality model to support MASs.

Figure 1 The meta-model of QM4MAS



As we mentioned above, a quality model is presented as a set of characteristics and the relationships between them (ISO, 2001). Before starting the development of a quality model, it is required to specify its structure in a meta-model. The quality meta-model defines precisely the model elements and their relationships in order to prevent ambiguous and simplify further refine (Deissenboeck et al., 2009). Our quality model is based on the quality meta-model presented in Figure 1. In fact, this quality meta-model is inspired from the structure of ISO-9126 quality model. Hence, our quality model is composed of several characteristics, and each characteristic is affected by several sub-characteristics. In turn, each sub-characteristic is assessed by several metrics. As it is defined by ISO-9126 quality model (ISO, 2001), a metric refers also to measurement method and measurement scale. By contrast, we have not followed the ISO-9126 quality model regarding the cardinality of the relationship characteristics-sub-characteristics. In our quality model, a sub-characteristic can affect several characteristics. Taking as an example, the *modularity* as a sub-characteristic in our quality model, it can affect the *maintainability* and *reusability* characteristics.

Hierarchical quality models suffer from the ambiguity due to the lack of precise criteria for the classification of model elements in characteristics and sub-characteristics (Deissenboeck et al., 2009). In order to prevent this drawback in the proposed quality model, we followed the IEEE Standard definition to software quality metrics methodology (IEEE, 1998). Thus, a quality characteristic [called in IEEE (1998) *quality factor*] is "a management-oriented attribute of software that contributes to its quality". In contrast, a quality sub-characteristic [called in (IEEE, 1998) a *quality sub-factor*] *is the decomposition of a quality factor to its technical components*. Each metric in the third level of our quality model is "a function whose inputs are software data and whose output is a single numerical value that can be interested as the degree to which the software possesses a given quality attribute" (IEEE, 1998).

3.1 The QM4MAS characteristics identification

ISO-9126 quality model is the result of standardisation of software quality models started by the *International Organization for Standardization* (ISO) in 1985. The model specifies six characteristics (*functionality*, *reliability*, *usability*, *efficiency*, *maintainability and portability*) applicable to every kind of software, including computer programs and data contained in firmware (ISO, 2001). Independent to any kind of software, we think that the existence of all cited characteristics in our model is very important.

Quite a long time ago, Dromey (1995) remarked that the *reusability* is omitted from the ISO-9126 quality characteristics despite its importance. In fact, reusability is "the degree to which a software module or other work product can be used in more than one computer program or software system" (IEEE, 1990). Consequently, the reusability characteristic can be viewed as an important quality characteristic for the development and maintenance team.

One reason for increasing the use of intelligent agents to develop complex software is their ability to produce flexible behaviour (Wooldridge, 2009). By "flexibility, we mean, the ability of the agent to change its behaviour according to its actual situation to satisfy its objectives" (Kiren, 2006). Despite that the IEEE Standard Glossary of Software Engineering Terminology (IEEE, 1990) defined the *adaptability* and the *flexibility* as synonyms; we choose to consider the flexibility as a quality characteristic and adaptability as a sub-characteristic. Our choice is justified by the difference between characteristics and sub-characteristics cited previously. Indeed, the flexibility, as a change in behaviour, can be done using several technical mechanisms, such as reactivity, pro-activity, interaction, adaptation or learning.

The first level of QM4MAS is composed of eight characteristics: *functionality*, *reliability*, *usability*, *efficiency*, *maintainability and portability*, *reusability* and *flexibility*. In our opinion, these minor changes of the ISO-9126 quality characteristics preserve the essence of the ISO-9126 quality model because its authors (ISO, 2001) cited that the characteristics must:

1 cover together all the software quality aspects

2 be only six to eight for reason of clarity and handling

3 describe the quality with the minimum of overlap.

Table 2 gives the definitions of our quality model characteristics.

Table 2 The QM4MAS's characteristics and their definitions

Characteristics	*Definitions*
Functionality	The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions (ISO, 2001).
Reliability	The capability of the software product to maintain a specified level of performance when used underspecified conditions (ISO, 2001).
Usability	The capability of the software product to be understood learned, used and attractive to the user, when it is used under specified conditions (ISO, 2001).
Efficiency	The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions (ISO, 2001).
Maintainability	The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications (ISO, 2001).
Portability	The capability of the software product to be transferred from one environment to another (ISO, 2001).
Reusability	The capability of a software module or other work product to be used in more than one computer program or software system (IEEE, 1990).
Flexibility	The capability of the software product to change its behaviour according to its actual situation to satisfy its objectives (Kiren, 2006).

3.2 The QM4MAS sub-characteristics identification

The ISO-9126 quality model decomposes the quality characteristics into a set of sub-characteristics. However, the list of sub-characteristics is not exhaustive. As is suggested by several authors (Radulovic, 2011), ISO-9126 sub-characteristics can be customised in order to take into account the particularities of some software products. In fact, the customising of the defined sub-characteristics can be done in several ways (Radulovic, 2011): adding some sub-characteristics, redefining some existed sub-characteristics or re-establishing the relationships between characteristics and sub-characteristics. In QM4MAS, we customised the defined ISO-9126 sub-characteristics by taking into account the agent-oriented paradigm features.

Table 3 The sub-characteristics added to ISO-9126 to support agent-oriented software

Sub-characteristic	*The definition*
Autonomy	The ability of the agent to operate without the intervention of humans or other agents (Dumke et al., 2010)
Reactivity	The ability of the agent to perceive its environment and generate instant responses to possible occurred changes (Dumke et al., 2010)
Pro-activity	The ability of the agent to exhibit goal-oriented behaviour (Dumke et al., 2010)
Social ability (Interaction)	The ability of the agent to affect other agents to satisfy their designed objectives (Dumke et al., 2010)
Adaptability	The ability of the agent to change its structure or their goals according to anew situation (Rejeb, 2005)
Rationality	The ability of the agent to control its decision to generate optimal behaviour (Carlin and Zilberstein, 2012)
Specialisation (role)	The task assigned to a specific individual within a set of responsibilities given to a group of individuals (Campbell and Wu, 2011)
Granularity	The degree of the agent complexity (Dumke et al., 2010)
Organisation	The collection of roles, that stand in certain relationships to one another, and that take part in systematic institutionalised patterns of interactions with other roles (Wooldridge, 2009)
Environment	Is the space in which agents interact with resources and other agents (Weyns et al., 2005)
Modularity	The degree to which a computer program is composed of discrete components such that a change to one component has minimal impact on other components (IEEE, 1990)

Although there is no consensus definition of the agent concept, the revision of the specialised literature (Alonso et al., 2008; Dumke et al., 2010; Wooldridge, 2009) allows us to draw the basic features of *agent* and *MASs*, namely: the autonomy, the reactivity, the pro-activity, the social ability (interaction), the environment, the adaptability, the rationality, the role (specialisation), the granularity and the organisation. Hence, the cited properties of the agent-oriented software can be added to the ISO-9126 quality model as sub-characteristics. Nevertheless, two

changes should be done to ISO-9126 sub-characteristics to avoid overlapping:

- The interoperability in the ISO-9126 quality model should be replaced by the interaction in our quality model because they have the same definition and the interaction notion is more suitable for agent-oriented software.
- The adaptability in the ISO-9126 quality model should be redefined according to the adaptability in the agent context. In addition, it seems important to distinguish the *flexibility* from *adaptability*. In fact, the flexibility, which is the ability of the agent to change its behaviour to satisfy its objectives, can be done by several mechanisms like the reactivity, the social ability (Wooldridge, 2009) or the adaptability (Rejeb, 2005). The learn-ability is a specific kind of the adaptability (dynamic adaptation) (Rejeb, 2005).

Moreover, we think that the modularity as a sub-characteristic is omitted in the ISO-9126 quality model. In fact, the *modularity* is an important technical concept which affects several software quality characteristics like: reusability and maintainability. Hence, a modular development of agents can significantly increase its quality.

Table 3 gives the added and adapted sub-characteristics to ISO-9126 quality model to support the agent-oriented software and their definitions. Indeed, other sub-characteristics can be added to support more specific kinds of agent (like mobility to support mobile agent).

3.3 The relationships between characteristics and sub-characteristics

The third step to develop our quality model for multi-agent systems (QM4MAS) consists in establishing the relationships between characteristics and sub-characteristics. As it is presented in the meta-model of our quality model, characteristics and sub-characteristics are related by affectation relation. So, a characteristic connected to a sub-characteristic indicates that the first is affected by the second. Naturally, the relationships proposed in ISO-9126 quality model should be preserved because the agent-oriented software is, primarily, software. However, these relationships should be extended to cover the added sub-characteristics. According to Dromey (1995), it is difficult to establish the relationships between characteristics and sub-characteristics because each characteristic is affected by *almost all* the sub-characteristics. For example, all the technical aspects of software can affect the maintainability characteristic. Because of that, and in order to product an understandable quality model, only the important relationships are taken into consideration in QM4MAS quality model. The following section explains how we connected each characteristic to a set of sub-characteristics. Obviously, we use the notions (the characteristics and the sub-characteristics) as they are defined in the above sections. Table 4 gives the overall of these relationships.

- Functionality

  The functionality of software refers to the achievement of their stated and implied needs. Agent-oriented software operates in two different levels to satisfy these needs: the agent level and the MAS level. In the first level, the agent attempts to satisfy its goals in an *optimal way* even in the *lack of the intervention* of other agents. So, it seems clear that the *autonomy* and the *rationality* of the agent are the key sub-characteristics that affect the functionality in the agent level. In addition, the interaction between agents allows the satisfaction of the MAS needs. Indeed, the *social ability* (like the coordination, cooperation and negotiation) is the central sub-characteristic that affects the functionality in the MAS level.

- Reliability

  The reliable software can remain operational even with the existence of errors. We think that the main principle to develop reliable software is preventing the errors propagation in order to limit their consequences. In agent-oriented software several techniques can limit the propagation of errors. First of all, if *interaction* is restricted between agents, potential failed agents cannot affect the functionality of other agents. Consequently, the restricted interaction can increase the reliability of the whole MAS. In addition, the autonomy allows the agent to operate without the intervention of other agents. Then, the *autonomy* allows the agent to be insensible to the mistakes made by other agents. Moreover, a well-developed agent can avoid potential failures from one of its parts. It is well-known that the *modularity* limits the propagation of errors. Hence, a modular agent can remain operational if an error affects a module without influence on other modules. In other cases, an agent makes use of the *adaptability* to change its behaviour, its structure or its goals if an error prevents the execution of the initial behaviour or the achievement of the initial goals.

- Efficiency

  The efficiency means a minimum use of resources. We can consider three kinds of resources that can be used by MASs: the execution time, the memory space and the communication bandwidth. In order to limit the communication bandwidth use, *the interaction* within the MAS should be at the least possible. In fact, the *organisation* of a MAS plays a significant role to limit the interaction between agents.

  The *granularity* can be considered in terms of complexity of behaviours which influences execution time of such behaviours, as it can be seen in terms of agent knowledge details which influence the occupied memory space. Especially, *the environment* of software agent is the memory space in which such agent is situated.

Regardless of the resource, the *rationality* increases its effective use because it represents a trade-off between the behaviour goal and the behaviour price.

- Usability

  The agent paradigm can increase the usability of software in several ways. First of all, the *social ability* of the agents which can be based upon ontology and known standards can increase the human-software interaction. Furthermore, the agents can operate without the intervention of others even human, thanks to *autonomy*. Consequently, the effort of using such software by human users can be reduced. Moreover, *the adaptability* (the learn-ability included) allows exploiting the users profile to create, automatically, more personalised interface.

- Maintainability

  The maintainability in agent-oriented software can be done in two different levels: the maintainability of an agent and the maintainability of the whole MAS. Naturally, the maintainability of software is affected by its complexity. Consequently, the maintainability of agent-oriented software is affected by the complexity of both agent and MAS. The complexity of an agent means, generally, its *granularity*. Furthermore, the *modularity* of an agent increases the understand-ability of its structure which positively influence to the maintainability.

  The complexity of the MAS is increased according to the number of the agents that compose the system and the *interaction* between them. However, the complexity of a MAS can be mastered thanks to the *organisation*.

- Portability

  Agents are situated in an environment. Taking an agent from its own environment for integrating it in other environment depends on three factors: *the environment*, *the agent* and *the interaction* agent-environment. Obviously, if the initial environment of the agent is designed to be changeable, agents can also live in a new environment. Moreover, if a weak interaction exists between agent and its environment, we can easily dissociate it from initial environment for integrating it in another environment. The nature of the agent has also an important influence to the portability. In fact, an agent can use its *adaptability* feature (changing its structure or its behaviour) to interact with a new environment.

- Reusability

  In order to simplify software development and increase the productivity in the development, software engineering principles encourage the reusability of existed software component. In agent-oriented software, we can use the reusability principle for two different perspectives: the reusability of some components of an agent to develop other agents or the reusability of an agent in other MASs. Naturally, agents developed as monolithic components render the reuse of their ability a hard task. In contrast, if agents are developed as *independent modules*, it becomes simple to reuse their modules to develop other agents.

  In order to reuse an agent to develop other MASs, *interaction* between this agent and other agents should be as weak as possible. Moreover, *specialisation* of an agent in a few roles allows the reuse of this agent in other MASs in a simpler way. In software engineering, the specific-purpose components have the most chance of being reused compared with the multi-purpose components. Consequently, the multi-purpose agents have a little chance to be reused for developing other MASs. Moreover, if we reuse a multi-purpose agent in other MASs we must confront two situations: changing the agent by eliminating undesirable roles according to new system's needs (waste of effort) or reusing the agent with all its roles with the possibility of never using some roles in the new system (waste of resources).

- Flexibility

  In order to change its behaviour according to its situation, the agent should be able to perceive and react to the environment changes (*reactivity*). Moreover, the agent should *interact* with other agents to change its behaviour according to the whole goal of the MAS. However, the agent should not change its behaviour according to the environment and other agents' situations and omitting their own goals. Indeed, the executed agent's behaviours should take into account the goals of the agent (*the pro-activity*). Beyond this, the agent can change its structure and goals thanks to *the adaptability* feature in order to treat some new situations.

Table 4 The characteristics-sub-characteristics relationships

The characteristics	*Connected sub-characteristics*
Functionality	Autonomy, rationality and social ability
Reliability	Autonomy, social ability, modularity and adaptability
Efficiency	Social ability, granularity, organisation, environment, rationality
Usability	Autonomy, adaptability and social ability
Maintainability	Modularity, granularity, organisation and social ability
Portability	Adaptability, environment and social ability
Reusability	Modularity, specialisation and social ability
Flexibility	Reactivity, pro-activity, adaptability and social ability

As indicated in Table 4, we remark that the social ability sub-characteristic affects all characteristics. We think that the quality in multi-agent software can be studied on two levels: the individual level in which we study the quality of each agent alone, and social level in which the quality of the whole system is considered.

3.4 The QM4MAS metrics

Some authors consider the metrics level of ISO-9126 quality model as the vulnerable point (Radulovic, 2011). In contrast to this view, we think that these authors have not taken into account the purpose of ISO-9126 quality model. According to Deissenboeck et al. (2009), the ISO-9126 is mainly used to *define* quality and it is not classified among metric-based quality models which are used to *assess* the quality. Our proposed quality model shares the same purpose with the ISO-9126 quality model. Consequently, only *some abstract guidelines* are proposed in the metrics level of our proposed quality model. We think that metrics are closely depended on implementation choice of multi-agent software like: implementation paradigm for agent, agent model, development language, …, etc. For example, despite that cohesion and coupling are well-accepted metrics for the modularity of software, assessing cohesion and coupling metrics for knowledge-based systems is different from those of object-oriented software. We applied these *abstract guidelines* through JADE application using more concrete metrics as it is presented in the following section.

Note that the proposition of an exhaustive list of metrics for MAS is beyond the scope of this paper. Several metrics are proposed to assess different aspects of multi-agent software. Consequently, the users of our quality model can reuse adequate metrics or inspire their appropriate metrics from the specialised literature to assess the different sub-characteristics of QM4MAS. The following list gives for each proposed sub-characteristics one or more metrics.

- Autonomy
  1. *Ratio of the lack of requesting services (RLRSs)*: this metric presents the RLRSs and the number of executed behaviours.
  2. *Ratio of resources availability (RRA)*: this metric is based on the availability of resources in the agent to reach its goal.
- Reactivity
  1. *Ratio of states changes (RSC)*: this metric presents the ratio of behaviours broken before reaching its purpose and the number of executed behaviours.
  2. *Time to respond to changes (TRC)*: this metric presents the average time to generate responds to perceived events. A response time to an event is the time from the event occurrence to the generation of the response.
- Pro-activity
  1. *Ratio of achieved purpose (RAP)*: this metric presents the ratio of behaviours that achieved their goals and the number of executed behaviours.
  2. *Time to achieve purpose (TAP)*: this metric gives the average of the execution time to achieve the goals of the executed behaviours.
- Social ability
  1. *Ratio of interaction utility (RIU)*: the interaction between agents is not a goal itself. In fact, agents interact to achieve their goals by coordination, cooperation or negotiation. In several situations, agents perform interaction without achieving their goals. Using this metric, we can assess the ratio of the interaction situations in which the agents achieve their goals and the whole interaction situations.
  2. *Ratio of interaction intention (RII)*: when receiving requirement of services agent can operate differently from offering the required services to the refusing of the cooperation. However, some agents accept the interaction for providing the required services but they cannot achieve this goal (for example, when the time required to provide the service is expired). Consequently, this metric is provided to assess the ratio of interaction situations in which an agent has the intention to cooperate and the whole interaction situations.
  3. *Ratio of understand-ability (RU)*: MASs are often used in open and complex environments. Consequently, two agents can use two different communication languages which negatively influence on the interaction utility. This metric can assess the interoperability between agents using the ratio of understood messages and the communicate messages.
- Rationality
  1. *Goal achievement acceleration (GAA)*: a rational agent is the one that achieves its goals in an optimal way. We can measure the optimisation level of the agent behaviour in measuring its progression to reach its goals according to the time. Hence, a rational agent progresses rapidly toward its goal as possible as it can.
  2. *Goal achievement by using resources (GAUR)*: rational agent takes into account the amount of resources used to achieve a goal. In fact, a rational agent can abandon some goals if their prices are considered expensive. Moreover, a rational agent can be satisfied by a partial achievement of its goal if the price of the result improvement is considered as expensive. This metric is a general form of the GAA metric (considering the execution time as a resource).

- *Specialisation*: we can assess this sub-characteristic by calculating the average number of roles played by agent (ARA).
- Granularity
  1 *Behaviour granularity (BG)*: this metric gives an assessment of the behaviour complexity according to the implementation paradigm of the agent.
  2 *Knowledge granularity (KG)*: the KG represents the complexity of knowledge within agent. In an object-oriented implementation of MASs, for example, we can assess the complexity of the agent's knowledge using different variables declared within the agent.
- *Organisation*: there is no a single type of organisation (Horling and Lesser, 2005). Consequently, it is impossible to define metrics to all organisations' types. However, we think that some metrics can be proposed regardless the organisation style of the MAS:
  1 *The number of sub-organisations*: Generally, organisations are structured as the aggregation of several sub-organisations (Ferber et al., 2004). A sub-organisation can be a group, a community or a level in hierarchical organisation. In fact, this decomposition increases the understand-ability of the MAS. Hence, we can use the number of sub-organisation as a metric for assessing the organisation sub-characteristic.
  2 *The average number of agents by organisation*: Each organisation or sub-organisation is composed of a number of agents. Naturally, as the number of agents increase as the understand-ability of the MAS becomes difficult. Hence, the average number of agents by organisation can be considered as a metric to assess the complexity of the organisation.
  3 *The relationships diversity*: relationships between agents in the organisation can be of several natures like: knowledge link, communication link and authority link. It seems evident that the diversity of relationships existed in an organisation can negatively influence to the understand-ability and the maintainability of the MAS. Hence, the relationships diversity metric assesses the number of relationships' kinds in the MAS.
- *Environment*: according to (Wooldridge, 2009), the complexity of the environment depends on several parameters: the accessibility, the determinateness, the episodically, the dynamicity and the continuity. Hence, we propose to assess the complexity of the environment using the average of these parameters.
- *Modularity*: Generally, the coupling and the cohesion are two well-accepted metrics to assess the modularity of software. Indeed, the coupling and cohesion metrics are closely dependent to the software paradigm. Hence, we can adapt one of these metrics according to the implementation paradigm of agent [knowledge-based system (Kramer and Kaindl, 2004) or object-oriented software (Husein, 2009)].

3.5 The rating in QM4MAS

The given metric can be used with a rating which reflects its importance for assessing the sub-characteristic. Similarly, the importance of sub-characteristics and characteristics can be expressed by rating. In our quality model, we do not specify the rating values because of the various models of agents and the diversity applications of MASs. So, the users of our quality model can specify the rating values according to their application and their agent model. For example, an application that is designed to operate in open system requires more importance of *RU* metric, while another application requires more importance of *RIU* metric.

As it is explained above, the metrics are closely dependent on the implementation paradigm or the implementation platform used to develop the MAS. We propose in the following section the application of the proposed quality model (QM4MAS) to JADE applications through a set of metrics.

4 Applying QM4MAS on JADE applications

JADE is a major open source software project and the most popular software agent technology platform (Bellifemine et al., 2007). Hence, we opted for this platform in order to apply and validate our quality model QM4MAS. The application of QM4MAS passes through the proposition of concrete metrics that reflect the particularities of JADE platform as we suggested previously. As specified in the meta-model of the proposed quality model, metrics require the presentation of several aspects: name, definition, scale, and measurement method. We start this section by presenting the measurement method used to assess the different aspect of JADE application.

4.1 Measurement method of JADE applications metrics

The metrics of software are generally divided into two kinds: static metrics and dynamic metrics. Static metrics are metrics that do not require the execution of the software. In contrast, the dynamic metrics require the execution of software. Despite of the benefits of the dynamic metrics, the static metrics are often more used because of the technological difficulties to collect the dynamic metrics (Tahir et al., 2010).

In order to assess some properties of JADE applications, we use some dynamic metrics. The proposed dynamic metrics are collected during the application execution using aspect-oriented technology.

Aspect-oriented paradigm (AOP) is relatively a recent programming paradigm introduced in 1997 for improving the modularity of the software and making the programming easier and faster (Kiczales et al., 1997). As it is presented in

Figure 2, the AOP principle consists of developing separately the cross-cutting concerns, called aspects, and incorporating them automatically to the system in the adequate points thanks to the waver.

Figure 2 The aspect-oriented programming principle (see online version for colours)

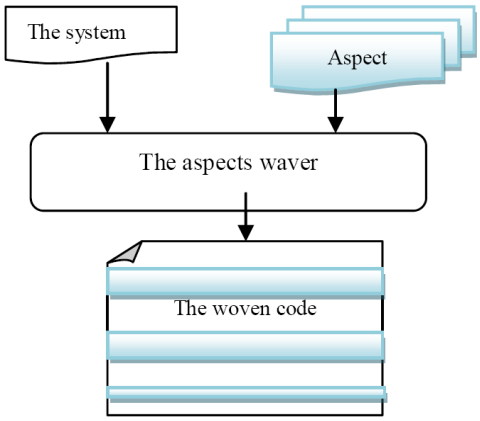


Figure 3 Simple example of *AspectJ* code

```
public class SimpleClass{
   //this is a simple class
   //...
   public void SimpleMethod(){
    //this is a Simple method
    System.out.println("I am Simple Method");
   }
}
```

(a)

```
public aspect SimpleAspect {
   //this is a simple aspect
      //this is the pointcut
   pointcut Point1() : call (public void
     SimpleMethod ());
      //this is the advice
   before() : Point1() {
       System.out.println("I will be executed
       before Simple Method Execution");
       }
}
```

(b)

JADE is a based Java platform allowing developing MASs. Hence, we used *AspectJ* (Laddad, 2003), which is an extension to the Java programming language to support aspect-oriented programming, to develop the different dynamic metrics used to assess some attributes of MAS. In fact, the dynamic metrics are implemented as aspects which allow reusing the same metrics (aspects) to assess JADE applications quality. The aspect represents the central unit of *AspectJ* (Laddad, 2003); it contains the code that expresses the waving rules for crosscutting. In addition to ordinary attributes of normal Java class (like data, methods and classes), the aspect incorporates specific *AspectJ* elements: *pointcuts*, *advice*, *introductions*, *declarations* and *compile-time declaration* (Laddad, 2003). Figure 3 gives a simple example of *AspectJ* programming code. In the part (a) of Figure 3, we show the *SimpleClass* class with its *SimpleMethod* method that displays a simple message ("I am Simple Method"). The part (b) of Figure 3 presents an aspect, called *SimpleAspect*, which specifies a *pointcut* (*point1*) in calling *SimpleMethod* method and an advice which should be executed before the *point1 pointcut*. Executing this code using *AspectJ*, the advice starts its execution displaying the message "I will be executed before SimpleMethod Execution", followed by the execution of *SimpleMethod* method which displays the message "I am Simple Method".

4.2 The proposed metrics to JADE applications

In order to propose the different metrics we have analysed the different JADE programming constructs. Indeed, each subset of JADE programming primitives is used to assess an attributes of MAS like autonomy, reactivity or pro-activity. In fact, some attributes can be assessed without the execution of the software; however, other attributes are dependent to the software behaviour. The dynamic metrics are developed as aspects which based on the execution of some JADE primitives to collect the information about the software behaviour. Obviously, each metric, represented as aspect, incorporates the *pointcuts* (to specify the primitives used to assess this metric) and *advices* (to implement the collection and measurement of the metric). The following list represents the different proposed metrics for JADE application:

- *Specialisation*: as it is quoted above, the specialisation can be assessed using the ARA metric. However, JADE platform has not provided the *role* notion. Consequently, we replaced the number of roles in the metric by the number of behaviours of each agent. Hence, the specialisation can be assessed using the average number of behaviours implemented in agent (ABA) metric.

$$ABA = \frac{N}{\sum_{i=1}^{N} Bi}, \tag{4.1}$$

  where N is the number of the agent and B_i is the number of the behaviours implemented in agent i.

- *Granularity*: two kinds of behaviours can be found in JADE platform: simple behaviours and composite behaviours. In order to assess the BG, we propose to assess the ratio of behaviours compositionality (RBC). Hence, we use the code source of the software to construct trees of behaviours of each agent. Considering each node in the tree as behaviour, the children of a node are the behaviours that compose their parent behaviour. Naturally, the leaves are the simple behaviours. Taking H_i as the height of the behaviour i in the tree of behaviours and B the number of behaviours of the agent, then

$$RBC = 1 - \frac{B}{\sum_{i=1}^{B} H_i}. \tag{4.2}$$

- *Autonomy*: in order to assess the autonomy in JADE application we used the *RLRSs* metric defined above. So, the service requested in JADE application should be done with sending a message. Obviously, the service requesting messages are identified by their performatives (*CFP*, *REQUEST*, *QUERY*, …, etc.). We used the aspect-oriented programming to capture the number of service request messages.
- *Reactivity*: JADE platform provides several constructs used to change the actual state of the agent (*block*(), *restart*(), *changeStateTo*(), *doActivate*(), *doSuspend*(), *doWait*(), *doWake*(), *restore*(), …, etc.). Executing those programming constructs, the agent changes its state. Consequently, we used the aspect-oriented programming to capture the number of changes within the agent behaviours in order to calculate *RSC* metric. Hence,

$$RSC = 1 - \frac{CB}{EB} \tag{4.3}$$

where *CB* is the number of changes in behaviours and *EB* is the number of executed behaviours. It is known that some behaviour can execute these constructs several times, so RSC metric can provide a negative value. Consequently, if the agent executes several instructions to change its state in the same behaviour we should consider the executed behaviour as several behaviours (as the number of these instructions). In this way, the RSC metric values are normalised between 0 and 1.

- *Pro-activity*: the behaviour has achieved its goal or still needs to be run. Thus, the number of the executed methods *done*() which return *true* value reflects the number of behaviours which achieve their goals. Consequently, *RAP* metric can be calculated by the formula

$$RAP = \frac{DB}{EB}; \tag{4.4}$$

where *DB* is the number of done behaviours (behaviours which achieve their goals) and *EB* is the number of executed behaviours.

- Social ability:
  1 *RII*: an agent that receives service request message (*CFP*, *QUERY*, *REQUEST*, …, etc.) can present its intention to interact (cooperation or coordination) by answering message (*PROPOSE* or *AGREE*). In fact, these kinds of messages reflect only the *intention* of the agent to cooperation or coordination because the agent *does not effectively* cooperate or coordinate. Several reasons can prevent the transformation of this intention to an effective fact, for example the proposition of the participant agent can be refused by the initiator agent or the participant agent can fail to achieve its goal. So, $RII = PIR/SR$ where *PIR* is the number of messages that reflect positive intention respond (*PROPOSE* or *AGREE*) and SR is the number of service request messages.
  2 *RIU*: an interaction situation is considered useful if the agent that received the requested service accomplishes its task and responds to the initiator agent. In interaction protocols, the participant agent responds to the initiator agent using *INFORM* message. Thus,

$$RIU = \frac{IM}{SR} \tag{4.5}$$

where *IM* is the number of *INFORM* messages and *SR* is the number of service request messages.

  3 *RU*: in the case where an agent received incomprehensible message, it can reply using *NOT_UNDERSTOOD* message. Consequently,

$$RU = \frac{NUM}{M} \tag{4.6}$$

where *NUM* is the number of *NOT_UNDERSTOOD* messages and *M* is the number of all exchanged messages.

Finally, we note that all the proposed metrics are normalised to be between 0 and 1; where 1 is the best value of the metric and 0 is its worst one.

Figure 4 An example of implemented aspect to assess the different metrics of JADE (see online version for colours)

```
public aspect MetricsAssessment {
    /* This Aspect for the Assessment of
       Metrics */
  pointcut ChangeBehaviourState() :
      (call (* *.block())||
      call (* *.restart()) ||
      call (* *.changeStateTo(*))||
        /* Others JADE Constructs Used
           to Change the State Of the
           Behaviour */
           ) ;
  before() : ChangeBehaviourState(){
        /* Here We Calculate the
           Number Of Changes Executed
           in the Behaviour */}
    /* Here We Implement the others
       Pointcuts */ }
```

4.3 Developed tool and case study

In order to assess automatically the proposed metrics for JADE applications, a tool has been developed. The tool consists of a library of aspects that allowing the automatic measurement and presentation of the above metrics. As an example of the implemented metrics, we present in Figure 4 the essential parts of the *MetricsAssessement* aspect. This aspect is used to assess the different metrics in JADE software. In this aspect, we present the *pointcut*, called *ChangeBehaviourState*, in which we define the JADE constructs used to change the agent's state. An advice

should be executed before this *pointcut* to calculate the number of changes in the state of the agent.

As previously mentioned, the benefit of using aspect-oriented programming allows the application of our tool to any JADE application because the developed aspects are entirely independent to the multi-agent software to be assessed. In order to validate our tool, we used the *portal agent* application which is adapted from *codes-sources* website (http://www.javafr.com//code.aspx?ID=49974). This open source application written on JADE aims to collect objects from agent. It is composed of three agents: *portal*, *seller* and *buyer*. Figures 5 to 7 give a description of each agent in state machine diagram.

As is presented in Figure 5, the *portal* agent starts its behaviour by the creation of the graphical interface and launches the *buyer* and *seller* agents. Then, it passes to block state until the reception of a message. When the portal agent receives the product list, it will display it and terminates its behaviour.

Figure 5 Description of *portal agent* behaviour

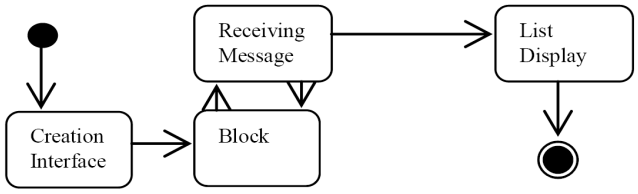


Figure 6 describes the *buyer* agent behaviour. This agent starts its behaviour by the request of product list from the *seller* agent; then it passes to block state, waiting the reception of a message. When it receives the product list, it will relay the product list to *portal* agent and passes to its end state.

The *seller* agent (Figure 7) waits until the reception of a message. If the received message is a request of product list then it will reply by sending the product list and passes to its end state.

Figure 6 Description of *buyer agent* behaviour

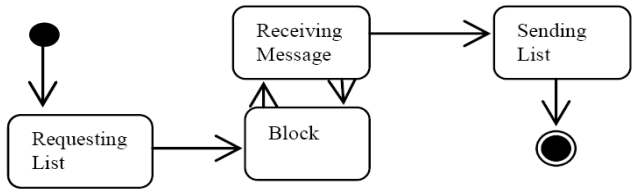
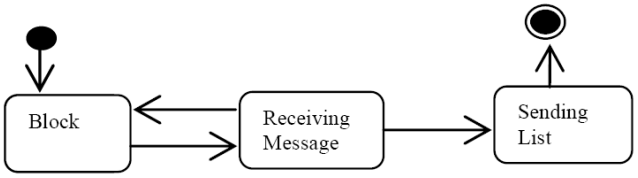
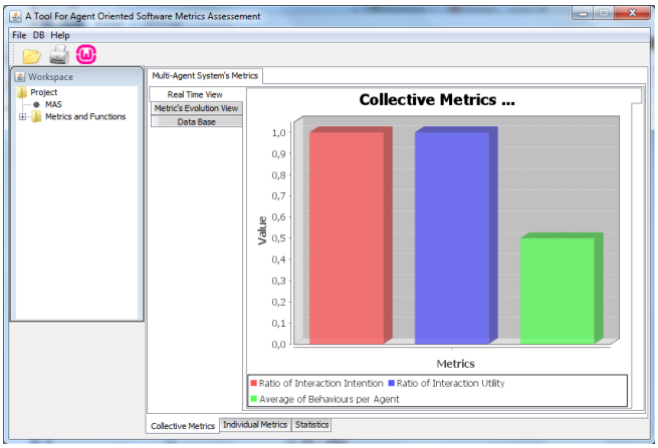


Figure 7 Description of *seller agent* behaviour



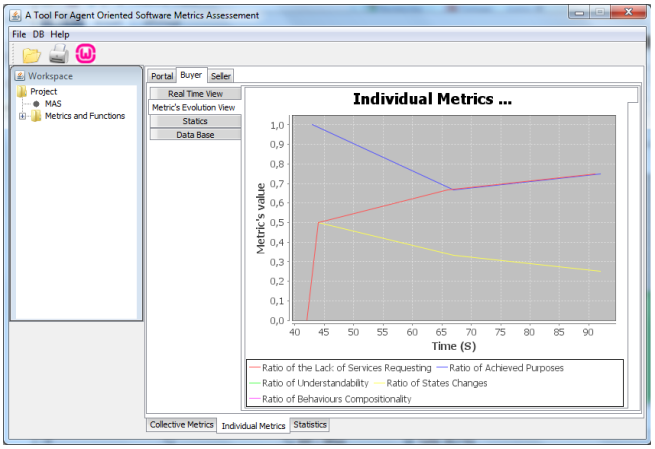
In order to calculate the different metrics, we need only to run the system to be evaluated using our tool. Hence, the assessment process presents the calculated metrics according to their sort: collective metrics or individual metrics. The collective metrics are the metrics calculated using all the agents composing the system. In contrast, the individual metrics are calculated independently for each agent. Moreover, the result of the measurement process can be presented in several ways. First of all, our tool gives a real-time presentation for each metric. For example, Figure 8 shows the real-time presentation of the collective metrics after a few time of execution. After the starting of the software, the *portal* agent subscribes and introduces in the MAS. Known that in this time, the MAS is composed of only one agent (the *portal* agent) which has two behaviours, then the value of the *average of behaviours per agent* (ABA) presented in green colour metric becomes 0.5. The other two metrics, *the RII* presented in red colour and *the RIU* presented in blue colour, have the default value equal to 1, because in this time any messages have been sent.

Figure 8 The real-time presentation of the collective metrics (see online version for colours)



The developed tool provides also the evolution of assessed metrics according to the time progression. We present in Figure 9 the evolution of individual metrics of the *Buyer* agent. This figure shows three diagrams: the *RLRSs* metric in red colour, the *RAPs* metric in blue colour and the *RSC* metric in yellow colour. The other metrics are omitted in the diagram, for the readability reason, because they are not changed during the execution of the application. As an example, we explain here the *RLRSs* metric evolution. In fact, the *buyer* agent has three main behaviours: one to request the product list, one to receive the product list and another one to send the product list to *portal* agent. During the execution of the first behaviour, the *buyer* agent should request a service (the product list). Hence, its *RLRSs* metric becomes 0 (at time 43 seconds). At time 44 seconds, the *buyer* agent started its second behaviour (to receive the product list) and its *RLRSs* metric becomes 0.5 because it executed two behaviours with only one request service. However, the *buyer* agent passed to block state until the reception of the product list. At time 66 seconds, the agent *buyer* has been notified by the reception of a message. Then, it will re-execute the precedent behaviour to receive the product list (its *RLRS* metric becomes 0.66). Finally, the *buyer* agent sends the product list to the *portal* agent at 92 seconds and its *RLRS* metric becomes 0.75.

Figure 9 The individual metrics evolution of the buyer agent (see online version for colours)



Thanks to the incorporated database in our tool, we can save the assessed metrics for further use. Moreover, the developed tool provides the possibility to generate a textual report of the assessed software execution.

5 Conclusions and future directions

Software quality is one of the most important purposes of software engineering. In order to understand, evaluate and predict the software quality, several models are proposed. In fact, each software paradigm has its particularities which require a specific quality model. So, in software engineering, we find quality models for object-oriented software, component-based software and service-oriented software. Despite that MAS is one of the well-known software paradigms, there is a very few work that has been developed to assess the quality of based agent software. Moreover, no overall quality model has been proposed for this paradigm. In this paper, we have customised the ISO-9126 quality model to support agent-oriented software. The proposed model is used mainly to define and assess the quality of MASs. The metrics proposed in this quality model are applied on based JADE applications in order to assess their quality.

This work can be extended in several ways. First of all, we should propose more metrics to assess other attributes of multi-agent software. Obviously, the proposed sub-characteristics do not have the same importance for determining overall MAS quality. In fact, each sub-characteristic has specific weight on the overall quality. These weights can be customised by users of our quality model. We propose to specify these weights according to specific application domains of MASs.

References

Alonso, F., Fuertes, J.L., Martinez, L. and Soza, H. (2008) 'Measuring the social ability of software agents', in the *Proceedings of the 6th ACIS International Conference on Software Engineering Research, Management and Applications*.

Alonso, F., Fuertes, J.L., Martinez, L. and Soza, H. (2009) 'Towards a set of measures for evaluating software agent autonomy', in *Proceedings of the Eighth Mexican International Conference on Artificial Intelligence*.

Alonso, F., Fuertes, J.L., Martinez, L. and Soza, H. (2010) 'Measures for evaluating the software agent pro-activity', in the *Proceedings of the 25th International Symposium on Computer and Information Sciences*.

Alonso, F., Fuertes, J.L., Montes, C. and Navajo, R.J.A. (1998) 'Quality model: how to improve the object oriented software process', in *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 5, pp.4884–4889.

Bansiya, J. and Davis, C.G.A. (2002) 'Hierarchical model for object-oriented design quality assessment', in *IEEE Transactions on Software Engineering*, Vol. 28, No. 1.

Behkamal, B., Kahani, M. and Akbar, M.K. (2009) 'Customizing ISO 9126 quality model for evaluation of B2B applications', in *Information and Software Technology*, Elsevier.

Bellifemine, F., Caire, G. and Greenwood, D. (2007) *Developing Multi-agent Systems with JADE*, Wiley Series in Agent Technologies, England .

Campbell, A. and Wu, A.S. (2011) 'Multi-agent role allocation: issues, approaches, and multiple perspectives', *Autonomous Agent and Multi-Agent Systems*, Vol. 22, No. 2, pp.317–355.

Carlin, A. and Zilberstein, S. (2012) 'Bounded rationality in multi agent systems using decentralized meta-reasoning', in Guy, T., Karny, M. and Wolpert, D. (Eds.): *Decision Making with Imperfect Decision Makers*, pp.1–28, Springer.

Deissenboeck, F., Juergens, E., Lochmannand, K. and Wagner, S., (2009) 'Software quality models: purposes, usage scenarios and requirements', in *WOSQ'09 Proceedings of the Seventh ICSE Conference on Software Quality*, IEEE Computer Society.

Dromey, R.G. (1995) 'A model for software product quality', in *IEEE Transactions on Software Engineering (TSE)*, Vol. 21, No. 2, pp.146–162.

Dumke, R., Mencke, S. and Wille, C. (2010) *Quality Assurance of Agent-Based and Self-Managed Systems*, CRC Press, USA.

Ferber, J., Gutknecht, O. and Michel, F. (2004) 'From agents to organizations: an organizational view of multi-agent systems', in Giorgini, P., Müller, J. and Odell, J. (Eds.): *Agent-Oriented Software Engineering (AOSE) IV, LNCS*, Melbourne, July, Vol. 2935, pp.214–230.

García-Magariño, I., Cossentino, M. and Seidita, V. (2010) 'A metrics suite for evaluating agent-oriented architectures', in the *Proceedings of the ACM Symposium on Applied Computing*.

Goeb, A. and Lochmann, K.A. (2011) 'Software quality model for SOA', in *Proceedings of the 8th International Workshop on Software Quality (WoSQ'11)*.

Gutiérrez, C. and García-Magariño, I. (2009) 'A metrics suite for the communication of multi-agent systems', in the *Journal of Physical Agents*, Vol. 3, No. 2, pp.7–14.

Horling, B. and Lesser, V. (2005) 'A survey of multi-agent organizational paradigms', in *The Knowledge Engineering Review*, Vol. 19, No. 4, pp.281–316.

Husein, S.A. (2009) 'Coupling and cohesion metrics suite for object-oriented software', in *International Conference on Computer Technology and Development (ICCTD '09)*.

IEEE (1990) *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std. 610.12-1990.

IEEE (1998) *IEEE Standard for a Software Quality Metrics Methodology*, IEEE Std. 1061-1998.

ISO (2001) *ISO/IEC 9126-1:2001 Software Engineering – Product Quality*.

Kaushik, A., Soni, A.K. and Soni, R. (2013) 'Radial basis function network using intuitionistic fuzzy C means for software cost estimation', in *Int. J. of Computer Applications in Technology*, Vol. 47, No. 1, pp.86–95.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J.M. and Irwin, J. (1997) 'Aspect-oriented programming', in *Proceedings of the European Conference on Object-Oriented Programming (ECOOP), LNCS*, Springer-Verlag, Finland, Vol. 1241.

Kiren, S. (2006) 'Flexibility of multiagent systems', in Kirn, S., Herzog, O., Lockemann, P. and Spaniol, O. (Eds.): *Multiagent Engineering – Theory and Applications in Enterprises*, Springer.

Kramer, S. and Kaindl, H. (2004) 'Coupling and cohesion metrics for knowledge-based systems using frames and rules', in *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 13, No. 3, pp.332–358.

Laddad, R. (2003) *AspectJ in Action*, Manning Publications, Greenwich, CT, USA.

Lee, K. and Lee, S.J. (2006) 'A quantitative evaluation model using the ISO/IEC 9126 quality model in the component based development process', in Gavrilova, M.L., Gervasi, O., Kumar, V., Kenneth Tan, C.J., Taniar, D., Lagana, A., Mun., Y. and Choo, H. (Eds.): *Computational Science and its Applications – ICCSA 2006, LNCS*, Vol. 3983, pp.917–926.

Lincke, R. and Löwe, W. (2006) 'Validation of a standard- and metric-based software quality model', in *Proceedings of the 10th Ecoop Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE)*.

McCall, J.A., Richards, P.K. and Walters, G.F. (1977) *Factors in Software Quality*, Vol. 1, ADA 049014, National Technical Information Service, Springfield, VA.

Radulovic, F. (2011) *A Software Quality Model for Evaluation of Semantic Technologies*, Master thesis in Artificial Intelligence Research, Universidad Politécnica de Madrid, Spain.

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

Singh, S., Mittal, P. and Kahlon, K.S. (2013) 'Empirical model for predicting high, medium and low severity faults using object oriented metrics in Mozilla Firefox', in *Int. J. of Computer Applications in Technology*, Vol. 47, Nos. 2/3, pp.110–124.

Singh, Y., Kaur, A. and Malhotra, R. (2014) 'A comparative study of models for predicting fault proneness in object-oriented systems', *Int. J. of Computer Applications in Technology*, Vol. 49, No. 1, pp.22–41.

Tahir, A., Ahmad, R. and Kasirun, K.M. (2010) 'Maintainability dynamic metrics data collection based on aspect-oriented technology', *Malaysian Journal of Computer Science*, Vol. 23, No. 3, pp.177–194.

Weyns, M., Schumacher, M., Ricci, A., Viroli, M. and Holvoet, T. (2005) 'Environments in multi agent systems', in *The Knowledge Engineering Review*, Vol. 20, No. 2, pp.127–141.

Wooldridge, M. (2009) *An Introduction to Multi-Agent Systems*, 2nd ed., John Wiley & Sons, UK.