A metrics suite for evaluating agent-oriented architectures

Iván García-Magariño
Dept. Software Engineering and Artificial Intelligence
Facultad de Informática,
Universidad Complutense de Madrid, Spain
ivan_gmg@fdi.ucm.es

Massimo Cossentino
ICAR CNR,
Consiglio Nazionale delle Ricerche
Palermo, Italy
cossentino@pa.icar.cnr.it

Valeria Seidita
Dip. Ingegneria Informatica,
Universitá degli Studi di Palermo,
Palermo, Italy
seidita@dinfo.unipa.it

ABSTRACT

The Multi-agent Systems (MASs) paradigm continues to consolidate itself as a new branch of software engineering. Traditional software engineering strongly recommends to apply metrics in software developments. However, several research groups of experts in agent-oriented software engineering agree that classical software metrics and object-oriented metrics cannot directly measure the quality of MAS architectures. For this reason, this work proposes a suite of metrics to measure certain quality attributes of MAS architectures, considering agents and their organization. Most of these metrics are inspired by object-oriented metrics but they are adapted to agent-oriented concepts. Proposed metrics are validated by the application to four problem domains and eight architectures.

Keywords

multi-agent systems, agent-oriented software engineering, architectures, metrics

1. INTRODUCTION

Only a few specialists can determine the quality of *Multi-agent System* (MAS) architectures. Hence, designing MASs is a trial/error process, in which most designers barely know the quality of a design until the system is running. In other words, designing MASs lacks the appropriate mechanisms to determine the quality in the initial stages of the development process. From this point forward, *agent-oriented architectures* refer to designs of MASs that, for instance, describe agents, their organization and their social interaction.

On the other hand, in other software engineering development paradigms, such as the object-oriented one, designers can rely on metrics, such as object-oriented metrics (e.g. [7] surveys some of these metrics), to determine the architectural quality in the early stages of development.

Software architecture designs [11] are incrementally improved by means of an iterative process with the following steps: collect stakeholders' interests, analyze architecture according to stakeholders' goals, create other architectures, rank the alternatives, and make decisions. Ranking alternatives usually regards a trade-off between benefits and costs, and a trade-off among different quality models. Although most of these steps could be directly applied to *Agent-oriented Software Engineering* (AOSE), ranking of alternatives still remains as a challenge due to the lack of reliable metrics for measuring agent-oriented architectures and the impossibility of direct application of object metrics, as literature [5] states. Hence, the absence of the appropriate metrics for agent-oriented architectures is arguably the tiny barrier that prevent engineers from applying mature processes ([12] provides a survey of these processes) for improving agent-oriented architectures.

In the literature, which is further described in Section 4, there are some attempts to provide metrics of MASs for a few quality attributes such as: *scalability* [16] that studies the relationship between performance and size in terms of number of agents, and *performance* [9] that is related with metrics via communication activity. Nevertheless, the literature still lacks appropriate metrics for measuring certain quality attributes of agent-oriented architectures, as before reported. For this reason, the aim of this work is to provide some quality models that allow engineers to quantitatively assess several quality attributes that are not explored yet in agent-oriented architectures. Although stakeholders can demand countless quality attributes in MASs, this work focuses on a metrics suite for measuring extensibility, modularity and complexity of agent-oriented architectures, as a point of start.

The hypothesis of this work is that some of the presented metrics are strongly related with the aforementioned quality attributes, as Figure 1 indicates. This hypothesis founds on the experience of two research groups in the AOSE field and an experimental basis. The experimental setup selects four problem domains (i.e. crisis-management, distribution of cinema tickets, bikes production, and bookshops) and eight different architectures.

The remainder of this paper is organized as follows: the next section describes the suite of metrics for agent-oriented ar-

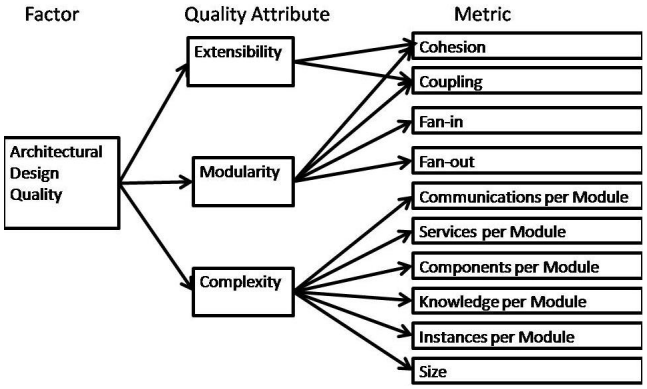

Figure 1: Measuring some quality attributes of agent oriented architectures

Object-oriented concepts		Agent-oriented concepts		
Name	**Description**	**Name**	**Description**	
Data Type coupling	Two modules use the same data type	Ontology Sharing	Two agents share knowledge for communicating	*Necessary*
Data coupling	Data from one module is used in another	Knowledge coupling	An agent uses knowledge (facts instantiated from the ontology) of another agent	⇓
Control coupling	One module may control actions of another	Behavioral coupling	An agent can control the behavior of other	⇓
Content coupling	A module refers to the internals of another module	Inner Structural Coupling	An agent uses internal elements of another agent	*Against MAS principles*

Table 1: Kinds of coupling in both object-oriented and agent-oriented architectures

chitectures; Section 3 presents the experimental basis justifying the presented metrics and their relationships with certain quality attributes of agent-oriented architectures; then, Section 4 compares the current work with other metrics applied in MASs, and Section 5 mentions the conclusions and future lines of research.

2. METRICS SUITE FOR AGENT-ORIENTED ARCHITECTURES

The following sub-sections present metrics for MAS architectures according to the measured quality attributes: extensibility, modularity and complexity. It is worth noting that this work takes inspiration from the *factors-criteria-metrics* approach proposed by McCall [14]. According to that (see Figure 1), some metrics are proposed to assess some quality attributes of the system. There is no unique relationship among metrics and quality attributes and, moreover, we do not claim that our suite is complete. Some aspects of the selected quality attributes (extensibility, modularity, complexity) are not covered by the presented metrics and moreover the discussed quality attributes do not depict a complete picture of the architectures of MASs. Nonetheless, this approach can provide useful hints in order to choose between different design options and/or evaluate some architectural aspects of the designed system before implementing it.

The presented metrics can only be applied to non-open MASs, because the architectures of MASs are not determined beforehand in open systems. In particular, there are parameters of the metrics that cannot be determined in open MASs such as the number of types of agents and the dependencies among them.

2.1 Metrics for extensibility

The *extensibility* metrics evaluates whether an agent-oriented architecture is designed to include hooks and mechanisms for expanding/enhancing the architecture with new capabilities without having to make major changes to the infrastructure of the architecture.

The metrics presented in this work for extensibility are intrinsically related to the concepts of *modules* and *components*, which, however, are not traditionally applied in the AOSE field. For the presented metrics, *agents* are the modules of the architecture; whereas, *tasks* are the components. This means we address tasks (significative pieces of agents behavior) as elementary and agents as aggregating them. This cannot be the right perspective in some approaches. For instance, in a role-oriented approach, the designer can be more interested in evaluating the architecture by considering roles and tasks as modules and components. Furthermore, in a pure *Belief-Desire-Intention* (BDI) approach, goals can be considered the components because they may preserve the autonomy property of agents. As it is common in AOSE, no unique way exists that can fit all design approaches. We think our proposal of considering agents as modules and tasks as components is reasonable and can be adopted in many cases but anyway, the approach remains perfectly general and designers can adapt the metrics to their needs. For this reason, from this point forward we will prefer using terms like module and component instead of agent and tasks so that our metrics description remains neutral to the specific implementation that some designer may need to have. It is worth noting that this proposal differs from other works, such as [5], that directly apply object-oriented metrics considering packages, classes or methods as candidates for modules and components, in the software of a MAS. Our proposal is to measure the agent-oriented architecture, while these other works measure the architecture of the software implementation of a MAS, which is usually more related with the MAS platform than the agent-oriented architecture itself.

The dependencies between components need to be detected for applying the presented metrics. In object-oriented architectures, dependencies can consider: data type coupling, data coupling, control coupling and content coupling. Table 1 matches these dependencies with agent-oriented dependencies, inspired by the following similarities. In object-oriented architectures, some modules can share data types (i.e. *data type coupling*); which is equivalent to several kinds of agents that share some languages (i.e. types of data exchanged). These languages are usually expressed in terms of ontologies to be shared (i.e. *ontology sharing*). Moreover, some modules need to obtain data from other modules (i.e. *data coupling*). In a similar manner, some agents may need

	Name	Definition
Ch_m	Cohesion of a Module	The number of internal dependencies of a module divided by the number of possible dependencies in the module according to the number of its components (see Equation 1). This metric can be applied to any kind of internal dependencies similar to the ones presented in Table 1.
Ch_a	Cohesion of an architecture	The average of the cohesion of each module of an agent-oriented architecture.
Cp	Coupling	The number of external dependencies divided by the number of possible external dependencies according to the number of modules (see Equation 2). A dependency can be of any kind of dependency presented in Table 1. This metric can be applied to any kind of dependencies presented in Table 1.

Table 2: Metrics for Extensibility in agent-oriented architectures

to request information from other agents (i.e. *knowledge coupling*). The *control coupling* would be equivalent to an agent that has control over the actions of other agents (i.e. *behavioral coupling*), which is against of the principles of autonomy in MASs [4]. The equivalent of content coupling in agent-oriented architectures, which is the direct access of an agent's private information from another agent (i.e. *inner structural coupling*), is also against the principles of MASs [4].

As a remark, in the aforementioned kinds of coupling, *internal dependencies* and *external dependencies* respectively denote dependencies between two components of the same module and dependencies between two components of different modules. For instance, in the INGENIAS metamodel [15], two components (i.e tasks) are bidirectionally dependent as long as they share any of the following elements: frame facts, internal/external application, resources. In the PASSI metamodel [3], dependencies between modules (i.e. agents) are: service dependencies, when an agent provides a service to another agent, and resource dependencies, when an agent consults the resource of another agent.

Metrics affecting the estimation of extensibility are presented in Table 2. The *Cohesion* metric (Ch_m) measures the dependence within a module, which is calculated with the Equation 1 as the number of internal dependencies divided by the number of possible dependencies in the module. Modules with only one component are assigned the maximum cohesion value, which is one; in addition, the suite provides another *Cohesion* metric (Ch_a) for calculating the average of the cohesion of each module of a given architecture. The cohesion of each module usually implies that it provides a coherent set of functionalities. For instance, let us suppose the same agent is used for buying goods on the Internet for his owner as well as for taking care of his agenda. Probably, the agent will include several tasks that can roughly be divided as belonging to the two main goals the agent pursues. In several applications this can be a desired features but the fact that each module is associated with one unique and coherent set of functionalities makes the architecture more understandable and, consequently, more extensible probably in several scenarios. In other words, the agent reported in the previous example would violate the *balanced distribution of responsibilities* criterion G. Booch in [1] considers as one of the three basic attributes of a good architecture. Arguments could be raised against the fact these rules seamless apply to the agent-oriented context as well, but we think a distribution of responsibilities among the agents in the system is anyway a good for MASs too.

The *Coupling* metric measures the dependence among modules of an architecture; it is obtained with Equation 2, as the number of external dependencies among modules divided by the number of maximum possible external dependencies. If an architecture has only one module, then the coupling metric obtains the maximum result, which is one. The low coupling is directly related with the extensibility, because a module can be added, removed or changed, according to new requirements, almost without interfering with other modules. Thus, the effort for incorporating new requirements is low when the coupling is low.

$$\mathbf{Ch_m(M)} = \begin{cases} \frac{IntDep(M)}{MaxDep(M)} & \text{if } NC \geq 2 \\ 1 & \text{if } NC = 1 \end{cases}$$

where:
- M is a module
- IntDep(M) is the number of internal dependencies (of a given dependency kind simimilar to the ones presented in Table 1)
- $MaxDep(M) = \frac{NC*(NC-1)}{2}$ (maximum number of possible internal dependencies)
- NC is the number of components of the module.

(1)

$$\mathbf{Cp(X)} = \begin{cases} \frac{ExtDep(X)}{MaxDep(X)} & \text{if } N \geq 2 \\ 1 & \text{if } N = 1 \end{cases}$$

where:
- X is an agent-oriented architecture
- ExtDep(X) is the number of external dependencies (of a given dependency kind presented in Table 1)
- $MaxDep(X) = \frac{N*(N-1)}{2}$ (maximum number of possible external dependencies)
- N is the number of components of the architecture.

(2)

It is worth remarking that these metrics can find a limit in the existence of infinite dependencies like it could happen for certain interaction protocols applied to open MASs. By now, we leave this configuration out of the scope of our framework.

2.2 Metrics for modularity

Modularity is the degree to which a MAS component may be separated and recombined. Modularity can depend on many variables in agent-oriented architectures. Some of these variables are the *Cohesion* and *Coupling* metrics, which have been already presented in the previous section, and the *Fan-in* and *Fan-out* metrics, which are defined in Table 3 and are inspired by [10]. However, more variables are planned to be included. It is worth noting that, even if some metrics are shared between modularity and extensibility (i.e. coupling and cohesion), modularity and extensibility are differ-

	Name	Definition
Fi	Fan-in	The number of incoming communication dependencies of a module.
Fo	Fan-out	The number of outgoing communication dependencies of a module.

Table 3: Metrics for the Modularity in agent-oriented architectures

	Name	Definition
ACmM	Average of Communications per Module	The number of communication protocols divided by the number of modules.
ASM	Average of Services per Module	The number of services divided by the number of modules.
AKM	Average of Knowledge per Module	The number of knowledge elements divided by the number of modules.
ACM	Average of Components per Module	The number of components of the system divided by the number of modules.
Sz	Size	Sum of the number of modeling elements of each kind. If necessary, the sum can be weighted by their kinds.
AIM	Average of Instances per Module	The average of instances of each module in the deployment.

Table 4: Metrics for Complexity in agent-oriented architectures

ent quality attributes, since extensibility regards adaptation to new requirements with low costs in different scenarios while modularity regards recombination of several architectures.

In modular architectures, modules usually have a high cohesion and a low coupling. In this manner, modules can be separated from each other and recombined without great costs.

The *Fan-in* and *Fan-out* metrics respectively measure the number of incoming and outgoing communication dependencies, when the communication is either explicit or implicit. These metrics can be indicators of which module of an architecture is responsible for a high coupling and, consequently, the cause of the low modularity of the architecture. Regarding metamodels, communication dependencies can have different names, such as *Interaction Units* in the INGENIAS metamodel and both *Service Dependency* and *Resource dependency* in the PASSI metamodel.

2.3 Metrics for complexity

Complexity characterizes a MAS with many parts in intricate arrangement. Inspired by the complexity measure [13] in traditional software engineering, this work presents several metrics with regards to the relationships among agent-oriented architectural elements, amounts of elements per module (i.e. knowledge elements, components and instances) and the global amount of elements (i.e. size). These metrics are defined in Table 4.

The *ACmM* and ASM metrics measure the average of relationships, respectively communications and services, for each module. These metrics are inspired by [13], where complexity of architectures is related to the ratio of edges per nodes in their *graph-like* representation.

Moreover, the *ACM, AKM* and *AIM* metrics respectively measure the average of the following elements per module: components, knowledge elements, and running instances. All of these metrics are intrinsically related with the amount of parts of each module. Thus, these metrics measure the complexity of the modules, which is related with the complexity of agent-oriented architectures since they are made of modules. It is worth noting that we are here not claiming that complexity is equal to size of the system; this is too simplistic and not true, but rather we argue that the presented elements concur to estimate the complexity because of their (possible) influence on that. As a remark, the AKM metric considers the following elements: ontological elements, frame facts, internal/external applications, and resources.

The *Size* metric measures the amount of elements in agent-oriented architectures, by summing the number of modeling elements of each kind. This sum can be weighted by the kinds of the modeling elements regarding, for instance, the difficulty of understanding them or their amount of information. However, this work has not adopted any weight at the present. It is worth mentioning that size and complexity are different concepts, since complexity measures understandability. Nevertheless, this study advocates that the size metric together with other metrics are strongly related with understandability. The *Size* metric can count the following concepts among others: components (i.e. tasks), modules (i.e. roles), knowledge elements (i.e. frame facts, ontological elements and so on), applications, communications (i.e. interactions), messages, initial states, goals, deployments, testing packages, internal relationships, external relationships and services.

3. VALIDATION OF THE PROPOSED METRICS SUITE

Pairs of architectures are measured in four problem-domains for the validation of the presented metrics. These architectures are briefly described in order to make the difference of quality evident in each pair. In each pair, the possible situations can be: quality of one architecture is higher (denoted with "+") for a given criterion than the other (denoted as "-"); or both architectures reach the same level (both denoted with "=") for a given criterion.

The first problem domain is the *crisis-management* [6] of a city, in which a poisonous material is released. The central medical services cannot heal all the poisoned people; thus, the MAS coordinates the people on the ground so they can help each other. For instance, the crisis-management problem domain can be addressed with two MAS architectures, in both of which, *Coordinator* agents coordinate the people on the ground, *Network* agents manage communications among Coordinator agents to each other, reporting news to the *Information* agent, which keeps an updated map of all the poisoned locations. In the first proposed MAS architecture (see Figure 2, with further description in [6]), which is called *Crisis-management*, each Coordinator agent contains the information about the user it interacts with, and the Information agent only contains an updated map of the affected positions. As one can observe in this architecture,

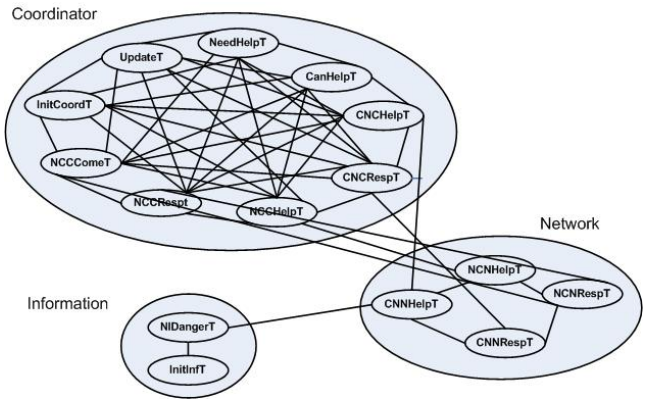

Figure 2: Architecture for crisis-management

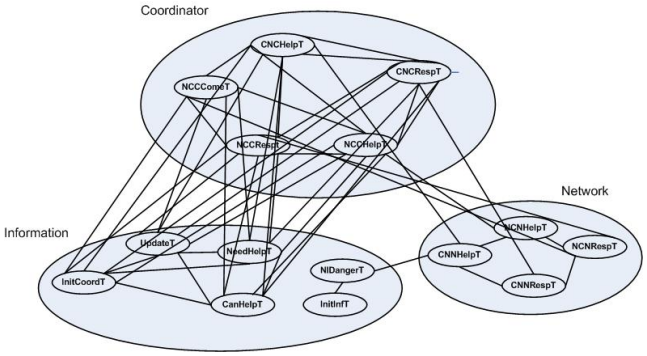

Figure 3: Architecture for crisis-management that includes the user state in the Information agent

the dependencies among tasks of different agents are few, and they are necessary for the coordination among different agents. On the contrary, in the second proposed architecture (see Figure 3), which is called *CrisisManagement-UserState*, both the map of the affected locations and the state of the user reside in the Information agent. The number of interactions is considerably higher in this architecture than in the previous architecture, because the Coordinator agents need to interact with the Information agent for both changing and consulting the user state. As this architecture reflects in Figure 3, the Information and Coordinator agents perform many tasks that share knowledge. This architecture is less extensible than the first one, in the scenario of adapting to new kinds of crisis. In the second architecture, these kinds of extensions would involve changes in both the Coordinator and Information agents, and the interactions between these two kinds of agents; while in the first architecture, only changes in the Coordinator agents would be necessary. Moreover, the modularity of the second architecture is decreased because both the Coordinator agent and Information agent are too dependent on each other, and these two modules are difficult to be separated and recombined with others.

In the *Cinema* problem-domain, a MAS assists users in buying cinema tickets. Among other kinds of agents, a *Buyer* agent searches for a given movie ticket, by interacting and negotiating with *Seller* agents. In the first architecture,

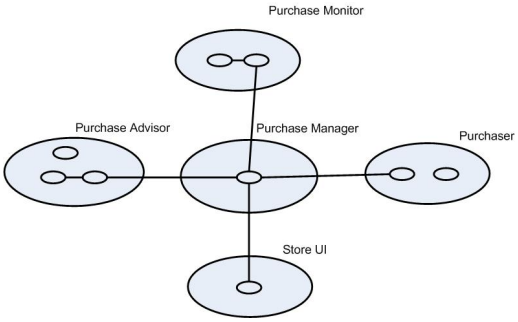

Figure 4: Excerpt of the architecture of the bookshop MAS

which is available from [8] and it is called *Cinema*, the Buyer role contains a directory of all the available cinemas. However, in the second architecture, called *Cinema-Directory*, the directory of all the cinemas is not within the Buyer role, but in the Seller role. Therefore, the buyer has to interact in the first place with the seller to know the list of existing cinemas scheduling a given movie. Then, it has to interact again with the seller that has the film chosen by the user. The increment of communications in the second architecture makes it difficult the separation of these two modules (i.e. Buyer and Seller agents) and the recombination with other modules.

In the *PPSBikes* domain, a MAS manages the production and distribution of bicycles. It includes agents for the production plant supply of new bicycle components and the administrative part of the distribution process. The differences between the first architecture, called *PPSBikes* and described in [2], and the second architecture, called *PPSBikes-Incidences*, are that the second architecture includes an agent, called *Incidences*, that is exclusively responsible for managing the *unresolved lots*; while in the first architecture the management of unresolved lots is joined with lots assignment and plant management within the *Administrator agent*. Since these three tasks are very related to each other, the same agent should consider them together in order to obtain coherent modules and to facilitate the adaptation to future changes related with lots; hence, the first architecture is easier to be extended in scenarios related with management of lots.

Finally, in the *Bookshop* problem domain, a MAS manages the distribution of books from the suppliers to the shops and on-line customers of a bookstore. An excerpt of the first proposed architecture is shown in Figure 4 and is further described in [2]. It includes, among the others, the *Manager* agent, *Shop* agents, and *Advisor* agents. Moreover, the *Bookshop-Broker* architecture is a variation of the first architecture, in which there is a Broker agent that tries to take advantage of some of the other agents, by providing the same service than the others being an intermediator. As one can observe in its architecture in Figure 5, the design quality decreases, because different agents duplicate services, hindering the understanding of modules. The second architecture makes the changes of services difficult since these services are duplicated.

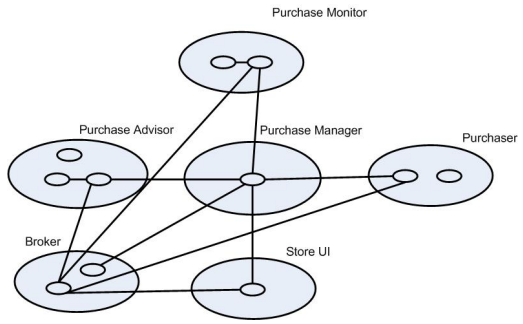

Figure 5: Excerpt of the architecture of the bookshop MAS with a broker

Architecture	Ch_a	Cp	Modularity	Extensibility
Crisis-management	0.83	2.00	+	+
Crisis-Management-UserState	0.67	8.67	-	-
Cinema	1.00	2.00	+	+
Cinema-Directory	0.78	2.67	-	-
PPSBikes	0.27	0.10	+	+
PPSBikes-incidences	0.22	0.20	-	-
Bookshop5	0.53	0.09	+	+
Bookshop5-Broker	0.48	0.20	-	-

Table 5: Coupling and cohesion in MAS architectures

3.1 Validation for extensibility

The validation uses the pairs of MAS architectures presented at the beginning of this section. The architectures of each pair are different, but both of them satisfy the same functional requirements in the same problem domain. Table 5 compares the architectural quality for extensibility with the measurement of cohesion and coupling metrics.

As one can observe in the two versions of the Crisis-management problem-domain (see Table 5), when the extensibility is higher (as justified for the *Crisis-management* architecture before in this document), the cohesion is higher ($0.83 > 0.67$) and coupling is lower ($2.00 < 8.67$). The same strong relationship occurs for the two architectures of the *Cinema* problem domain: the architecture with a higher extensibility obtains a higher cohesion ($1.00 > 0.78$) and a lower coupling ($2.00 < 2.76$).

PPSBikes-incidences architecture is considered worse in architectural extensibility than the *PPSBikes* for the reasons mentioned before. In the *PPSBikes-incidences* architecture, the cohesion has lower ($0.22 < 0.27$) and the coupling is higher ($0.20 > 0.10$).

In addition, in the *Bookshop* problem domain, the architecture with the Broker agent has a lower extensibility according the reasons mentioned before. The least extensible design of this pair has a lower cohesion ($0.48 < 0.53$) and a much higher coupling ($0.20 > 0.09$).

According to the results of this study in several problem domains, the architectural extensibility is directly related with the proposed cohesion metric, and inversely related with the proposed coupling metric.

	Coordinator		Network		Information	
	fan-in	fan-out	fan-in	fan-out	fan-in	fan-out
Crisis-management	2	2	2	3	1	0
Crisis-Management-UserState	3	4	2	3	3	1
	User		Buyer		Seller	
	fan-in	fan-out	fan-in	fan-out	fan-in	fan-out
Cinema	1	1	3	4	2	2
Cinema-Directory	1	1	4	5	3	3
	Monitor		Advisor		Manager	
	fan-in	fan-out	fan-in	fan-out	fan-in	fan-out
Bookshop	0	1	1	0	1	2
Bookshop-Broker	0	2	1	1	1	3
	Purchaser		Store		Broker	
	fan-in	fan-out	fan-in	fan-out	fan-in	fan-out
Bookshop	1	0	0	0		
Bookshop-Broker	1	1	0	1	5	0

Table 6: Fan-in and Fan-out in MAS architectures

3.2 Validation for modularity

The coupling and cohesion metrics are also compared with modularity in Table 5. The same arguments proposed in the previous section lead us to conclude that modularity is also directly related with the cohesion metric and inversely related with the coupling metric.

For detecting causes for low modularity, designers can by now observe whether a value of fan-in and fan-out for a given module is high in comparison to the values of other modules of the architecture. However, we are planning to provide a set of recommended values in the future.

As one can observe in Table 6, for the *CrisisManagement-UserState* architecture, the fan-in and fan-out values of both the *Coordinator* and *Information* modules (i.e roles) have increased in relation with the values of the *Crisis-management* architecture. In this case, the metrics detect a high fan-out in Coordinator module in comparison to other modules, which is the result of a high number of communication protocols originated by the attempt of coordinating people without the necessary information. This partially proves the usefulness of these metrics in detecting architectural problems.

Moreover, in the *Cinema-Directory* architecture (see Table 6), both the fan-in and fan-out of the *Buyer* module are high in comparison to other values. Therefore, the metrics detect the communications in the *Buyer* are high, and the reason is the *Buyer* does not have any necessary information: directory of cinemas, the prices, timetables and availability. A designer would change the architecture for incorporating more information in the Buyer for improving the modularity, when knowing the results of these metrics.

Architecture	Size	ACM	AKM	ACmM	ASM	AIM	Complexity
Cinema	58	3.00	4.67	1.33	1.00	2.33	-
Crisis-Management	101	5.00	6.67	2.00	1.00	7.00	+
PPS-Bikes	89	2.60	9.20	1.00	0.20	1.00	=
Bookshop	123	1.80	3.90	3.40	0.20	1.00	=

Table 7: Complexity in MAS architectures

As one can observe in Table 6, in the *Bookshop-Broker* architecture, the Broker agent has a high value in the fan-in metric in comparison to other values. In addition, this value is much higher than the other fan-in and fan-out values of other modules. According to this result, a designer should think that there is a problem with this module and evaluate whether a better architecture can be designed. For instance, different kinds of Brokers can be defined for each buying relationship, or this agent could even be removed. Both decisions would improve the quality of the architecture, and decrease this fan-in value.

In conclusion, according to this study, a high value of fan-in or fan-out in a module in contrast to other values is usually a sign of an architectural problem that hinders modularity.

3.3 Validation for complexity

The measurement of some of the presented metrics are compared with complexity of architectures in Table 7. For the complexity, practitioners need to observe the whole MAS architectures, which are omitted in this paper for the sake of brevity but are available from [6, 8, 2]. When comparing the Cinema and Crisis-management architectures, the following metrics are related with complexity: ACmM, ACM, AKM, AIM and Size. However, ASM is not necessarily related with the architectural complexity.

In the comparison between PPSBikes and Bookshop architectures (see Table 7), experts assess a similar complexity for both architectures. Metrics provide several variables that have effect on complexity. To begin with, the average of services, and the average of instances are exactly the same. The amount of elements (Size metric) is higher and communications (ACmM metric) are more complex in Bookshop example; whereas the components (ACM metric) and the knowledge (AKM metric) are more complex in the PPSBikes architecture. Therefore, the human assessment of quality and the metrics assessment matches, because both mechanisms indicate that the PPSBikes and Bookshop examples are approximately of the same complexity (with pros and cons).

In conclusion, the ACmM, ACM, AKM, AIM and Size metrics are related with the complexity of agent-oriented architectures.

4. RELATED WORKS

Rana and Stout [16] propose some metrics for measuring the scalability in MASs. This work focuses on predicting the performance of MASs when there is a larger number of agents instantiated in a MAS, thus they relate the number of instances and performance.

Moreover, Helsinger et al. [9] propose several metrics to measure the performance in MASs. These metrics are applied at run time, and they evaluate the number of different kinds of messages that are used in a broadcast communication, denoted as *blackboard* communication. That work also measures the size of exchanged messages.

Schroeder [18] defines some metrics to measure the distance between agents. Between communicative agents, they count the sharing interests. For mobile agents, that work considers the physical distance. Although that work is specific for the agent-oriented domain, its aim is not the quality of the design but the visualization of MASs.

Nevertheless, our work measures some quality attributes that are not explored by the aforementioned works, which are extensibility, modularity and complexity of agent-oriented architectures.

Garcia et al [5] applied object-oriented metrics for measuring modularization in MASs, besides other software metrics such as lines-of-codes. According to that study, coupling is one of the indicators that have effect on MASs modularity. However, that work concluded that a low cohesion is obtained in MASs for components. That study only refers to object-oriented components, such as object-oriented classes. In addition, Garcia et al [5] indicated that the Object-oriented pattern obtained worse results than agent-oriented patterns for MASs from a modularization point of view. This work indicates that the difficulty is to match object-oriented elements with agent-oriented abstractions. Furthermore, Sant Anna et al [17] presented a complete set of modularity measurement values, in order to compare aspect-oriented and non aspect-oriented architectures in AOSE. However, in that work, the metrics consider components in a pure object-oriented view. Conversely, our work defines new metrics in terms of agent-oriented concepts, obtaining results that are more relevant to AOSE.

5. CONCLUSIONS AND FUTURE WORK

This work provides a metrics suite for measuring agent-oriented architectures. A brief study, involving four problem domains and eight MAS architectures, advocates that these metrics are strongly related with the following quality attributes: extensibility, modularity and complexity.

One of our short-term goals is to provide a reliable set of recommended values for the metrics, so designers can measure their agent-oriented architectures and know whether their designs are within a recommended range for each quality attribute. Furthermore, the presented metrics can be tuned by introducing some weights in the equations. In particular, this work plans to include weights in the *ACmM* metric to consider that some kinds of communications make architectures more complex. For instance, a *call-for-proposals* communication, in which an agent asks a group of agents for several proposals, is more complex than an *one-to-one-inform* communication, in which only a message is transmitted between two agents.

Moreover, the empirical relationship between measurement values and experts assessment can be confirmed with a wider range of examples and a wider number of experts. These metrics are also planned to be validated by means of other metrics.

Finally, these metrics can be applied to study which AOSE methodologies obtains architectures with certain quality attributes, and more metrics can be added to the presented suite in the future.

6. ACKNOWLEDGMENTS

This work is supported by the project "Agent-based MOdeling and Simulation of Complex Social Systems (SiCoSSys)", funded by Spanish Council for Science and Innovation, with grant TIN2008-06464-C03-01.

7. REFERENCES

[1] G. Booch. The defenestration of superfluous architectural accoutrements. *IEEE Software*, 26(4):7–8, 2009.

[2] M. Cossentino. From Requirements to Code with PASSI methodology. *Agent-oriented Methodologies, Chapter IV, B. Henderson-Sellers and P. Giorgini (editors)*, pages 79–106, 2005.

[3] M. Cossentino, S. Gaglio, L. Sabatucci, and V. Seidita. The PASSI and Agile PASSI MAS Meta-models Compared with a Unifying Proposal. *Multi-agent Systems And Applications IV: 4th International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2005, Budapest, Hungary, September 15-17, 2005: Proceedings*, LNCS 3690:183, 2005.

[4] S. Franklin and A. Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. *Lecture Notes in Computer Science*, 1193:21–36, 1997.

[5] A. Garcia, C. Sant Anna, C. Chavez, V. da Silva, C. de Lucena, and A. von Staa. Separation of concerns in multi-agent systems: An empirical study. *Lecture notes in computer science*, 2940/2004:343–344, 2004.

[6] I. García-Magariño, C. Gutierrez, and R. Fuentes-Fernández. The INGENIAS Development Kit: a practical application for crisis-management. In *The 10th International Work conference on Artificial Neuronal Networks (IWANN2009)*, volume 5517 of *Lecture Notes in Computer Science*, pages 537–544. Springer, 2009.

[7] M. Genero, M. Piattini, and C. Calero. A survey of metrics for uml class diagrams. *Journal of Object Technology*, 4(9):59–92, 2005.

[8] *Grasia* web: `http://grasia.fdi.ucm.es` (in "Software"→ "Additional Material for Papers" or "Trainning"→ "Full Development Examples").

[9] A. Helsinger, R. Lazarus, W. Wright, and J. Zinky. 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*, pages 843–850. ACM New York, NY, USA, 2003.

[10] S. Henry, D. Kafura, and K. Harris. On the relationships among three software metrics. *ACM SIGMETRICS Performance Evaluation Review*, 10(1):81–88, 1981.

[11] R. Kazman, L. Bass, and M. Klein. The essential components of software architecture design and analysis. *The Journal of Systems & Software*, 79(8):1207–1216, 2006.

[12] R. Kazman, L. Bass, M. Klein, T. Lattanze, and L. Northrop. A basis for analyzing software architecture analysis methods. *Software Quality Journal*, 13(4):329–355, 2005.

[13] T. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.

[14] J. McCall, P. Richards, and G. Walters. Factors in Software Quality (Vol. 1,2 and 3). Technical report, Nat'l Tech. Information Service, Springfield, Va. NTIS AD-AO49-014, 015, 055, Nov. 1977.

[15] J. Pavón and J. Gómez-Sanz. Agent Oriented Software Engineering with INGENIAS. *Multi-Agent Systems and Applications III*, 2691:394–403, 2003.

[16] O. Rana and K. Stout. What is scalability in multi-agent systems? In *Proceedings of the fourth international conference on Autonomous agents*, pages 56–63. ACM New York, NY, USA, 2000.

[17] C. Sant Anna, C. Lobato, U. Kulesza, A. Garcia, C. Chavez, and C. de Lucena. On the modularity assessment of aspect-oriented multiagent architectures: a quantitive study. *Int. J. Agent-Oriented Software Engineering*, 2(1):34–61, 2008.

[18] M. Schroeder. Using singular value decomposition to visualise relations within multi-agent systems. In *Proceedings of the third annual conference on Autonomous Agents*, pages 313–318. ACM New York, NY, USA, 1999.