ACSIJ Advances in Computer Science: an International Journal, Vol. 3, Issue 5, No.11 , September 2014
ISSN : 2322-5157
www.ACSIJ.org

ACSIJ
WWW.ACSIJ.ORG

AMA: a compound methodology for designing and implementing agent-based systems

Erfan Ghandehari[1], Fatemeh Saadatjoo[2] and Mohammad Ali Zare Chahooki[3]

[1] Department of Computer Engineering, Science and Art University
Yazd, Iran
erfan.ghandehari@sau.ac.ir

[2] Department of Computer Engineering, Science and Art University
Yazd, Iran
saadatjou@sau.ac.ir

[3] Department of Electrical and Computer Engineering, Yazd University
Yazd, Iran
chahooki@yazd.ac.ir

Abstract

Agent oriented software engineering (AOSE) is one of the new developments in computer software technology. This technology provides facilities for design and generation of complex distributive systems in the form of agent oriented methodologies. It also analyses interactions among the agents and calculations based on agent. Various methodologies have already been presented for development of agent oriented software which can be used in different software projects. Regarding the fact that in software projects, selection of the appropriate methodology for development leads to the made product having appropriate quality and efficiency, recognition of the methodologies' weak and strong points in order to apply them in different projects seems crucial. In this article we intend to develop a compound methodology by mixing the strengths of methodologies in all phases. In this connection, the strengths of the three methodologies of AOR, MASSIVE, and ADELFE are extracted based on assessment methods and criteria including concepts and conceptions, modeling language, process and pragmatism. Then, a methodology dubbed "AMA" is developed through mixing the strengths of these methodologies.

Keywords: *agent-oriented software engineering, agent-based system, AOR, MASSIVE, ADELFE.*

1. Introduction

Agent-oriented software engineering is a type of engineering with agents as its main abstraction. In other words, agents are the main components of such software. The agent-oriented approach toward software engineering means dividing the problem into several autonomous and interacting agents which interact with each other to achieve the goal they have been designed for [1].

AOSE was developed to respond to the essential needs of software engineering and agent-based computations [2]. Its main goal is creating the methodologies, tools and facilities required for the simple preparation and maintenance of agent-oriented software [3]. As object-oriented software engineering (OOSE) was not able to respond to the needs of agent-oriented software, the emergent need for a new engineering compatible with agent perspectives led to the development of AOSE from OOSE [4]. One of the main challenges ahead of AOSE is that it lacks a complete software development methodology. Although a large number of agent-oriented methodologies have already been proposed, a few of them fully cover software engineering activities and none of them fully supports the development needs of agent-based systems. Therefore, it currently seems necessary to work on developing an integrated and comprehensive methodology [5-8]. In the following paragraphs we will examine studies aimed at developing agent-based methodologies, which are of highest importance among all the methodologies developed. Dileo et al. (2002) added the ontology modeling phase to MASE's analysis phase. According to this development method, first the purpose and range of ontology required for the agent is determined and then data existing within the range of the system are gathered [9]. In an improvement, Deloach et al. added the ability to model inter-agent organizational relations to the methodology. In this type of development, the analysis and modeling of the organizational structure takes place after the ontology modeling phase [10]. Giving mobility to the system's agents was another improvement to MASE. In this connection, the MOVE command was added to the methodology during the modeling of activities which takes place in the form of a state diagram [11]. Development work on agent-oriented methodologies has not been limited to MASE and still covers GAIA, TROPOS and other methodologies as well. In one of the improvements to GAIA, the ability to model systems implementable on the

internet was added to this methodology by Zambonelli et al. In their research, the ability to model inter-agent relations was added to GAIA given the openness and conflictive objectives of the agents [12]. As for TROPOS, an official goal analysis model was added to the methodology in order to improve it [13]. A method for assigning tasks to roles was also presented by this methodology [14].

One challenge ahead of these methodologies and other development work is that the existing methodologies cannot cover all software engineering activities and, therefore, more research should be carried out in order to develop the next generation of agent-oriented methodologies and increase the chance for adapting these methodologies to multi-agent systems by creating further convergence between the analysis and design phases of agent-oriented methodologies. Given these challenges, the current article tries to identify the strengths and weaknesses of AOR [15], MASSIVE [16] and ADELFE [17] through referring to the experiences of experts and then mix the strengths in a bid to present a new compound methodology. Mixing the strengths of widely-used methodologies could pave the way for developing the next generation of agent-oriented methodologies. The following section will discuss agent-oriented and object-oriented approaches. Section three will introduce parameters for assessing methodologies categorized in four groups of "concepts and conceptions", "modeling language", "process", and "pragmatism". Section four will extract the strengths of each methodology by referring to expert view, while section five will introduce the various phases of AMA methodology. Conclusions and suggestions will be provided in section six.

2. Comparison Between Object-Oriented and Agent-Oriented Approaches

AOSE has evolved from OOSE. In other words, agents have been derived from objects [18]. LIND (2001) compared object-oriented systems with agent-oriented ones in terms of hardware, theory, implementation time, programming language, and designing language [19], producing the following results: (a) objects have a centralized organization, while agents allow distributed computing,(b) objects existing in a system are more integrated than agents, (c) agents could not be created or destroyed as freely as objects, (d) objects have a fixed behavior and structure, but agents learn from their experiences and change their behavior, (e) interactions between objects mostly take place in response to the request of one object, while interactions between agents occur both in response to the environment or requests of other agents, (f) interactions between objects are usually synchronous, but interactions between objects are usually non-synchronous, and (g) agents have a stronger encapsulation than objects.

Since agents are derived from objects, there are also similarities between them. Yet, parameters from both approaches could be mapped to each other in spite of these similarities and differences. Table (1) presents a typical mapping of object-oriented and agent-oriented approaches.

Table 1: mapping of object-oriented and agent-oriented approaches [19]

Object-Oriented Approach	***Agent-Oriented Approach***
Abstract Class	Generic Role
Class	Domain-Specific Role
Class Variables	Knowledge, Belief
Method	Capability
Inheritance	Role Binding
Prototyping	Specific Role + Personal Knowledge
Compound	Holon Agents
Method Invocation	Message Exchange
Cooperation	Interaction

From table (1) it could be concluded that the agent-oriented approach has offered solutions for all capabilities of the object-oriented approach. These solutions are suitable for analyzing and designing agent-based systems.

3. Assessment Criteria and Methods

This section proposes a methodology assessment framework by comparing the features of agent-oriented and object-oriented approaches. This framework is consisted of a set of criteria and roles and includes not only the features of classic software engineering but also the exclusive features of AOSE. To prevent applying a wrong comparative framework to AOSE, criteria and features incorporated in the assessment framework are taken from previous studies [20-23] on comparison of AOSE methodologies. Figure (1) illustrates the assessment framework.

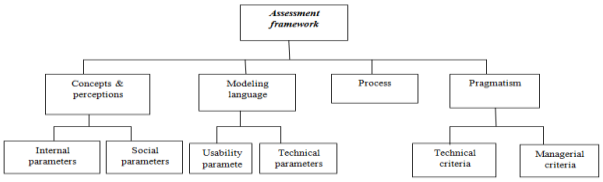

Figure.1 General Framework of agent oriented methodologies assessment

There are usually two types of assessment features for agent-oriented methodologies. The first feature denotes the degree of the methodology's support for a specific feature. It means that the methodology covers that feature with a certain degree of support. A numerical scale is employed for this type of assessment to assign values to each criterion and enable a quantitative comparison. The second feature denotes features supported by the methodology. This article uses the second feature. Each assessment criteria will be discussed below.

3.1 Concepts and Conceptions

Concepts and conceptions include a notion, abstraction, or assumption of the samples specified in the problem. This section deals with the essential assumptions of agents and agent-oriented systems. The internal parameters of this criterion include [20-23]: (a) autonomy: the ability to act or decide without the direct intervention of the controller or other agents, (b) reactivity: the ability to understand the environment and respond to changes, (c) purposefulness: the ability to show purposeful behavior through innovation rather than being merely responsive, and (d) simultaneous implementation: the agent's ability to handle several goals or incidents simultaneously. The social parameters include: (a) teamwork: the highest level of cooperation among agents in which all agents on the team proceed toward a common goal, (b) protocol: this criterion examines levels of support for defining authorized negotiations with respect to a valid streak of messages exchanged between two agents, and (c) communication language: a regular set of messages that defines patterns for authorized interactions between entities and includes the language used for communication between agents.

3.2 Modeling Language

Agent orientation is the basis for each AOSE methodology [20-23]. Modeling language is generally considered a main component of each software engineering methodology when displaying designs in terms of agent orientation. A modeling technique is consisted of a set of models and shows the system and its various features in different levels of abstraction. Usability criteria cover various types of measurements and include [20-23]: (a) understandability and clarity: these criteria determine how good the symbols are and how the syntactic composition of models and symbols is properly defined, (b) eloquence, meaningfulness, and competence: the number of dynamic and static models as well as the number of different viewpoints that illustrate the target system are a good yardstick for measuring these criteria, and (c) ease of use: this feature deals with the ability to access, understand and use the method easily. It is important for the modeling language to be not only easy to understand but also easy to use. Technical parameters include: (a) compatibility: the models should not be incompatible; this feature is of great importance with respect to designing and analyzing the models, (b) follow-up capability: this capability means that designing documents should be easy to understand and follow, (c) filtration: the modeling technique uses a clear path to filter the model through gradual phases in order to enable implementation of the model or, at least, connect the implementation level to designing features, and (d) reusability: reusability supports design components.

3.3 Process

Process is considered an important part of each software engineering methodology and highlights sets of activities and phases as part of software life cycle when building and engineering software systems. These activities and phases form the process and help system analyzers, developers and administrators with software development [20] [21]. Process criteria include [20-23]: (a) requirements analysis: understanding the system and determining its extent and purpose are the main goal of the requirements analysis; this analysis specifies the system's goals and boundaries, (b) architecture design: subsystems, data, data's internal communications, and flow control are defined for the system's architecture design, and (c) implementation: implementation proceeds phase by phase through the features of architecture design and takes place based on the recognition of the mapping between implementation structures and design assumptions. Process also covers parameters such as testing and troubleshooting, establishment, and support and maintenance.

3.4 Pragmatism

Pragmatism is associated with the practical aspects of the development and use of methodologies. This section deals with pragmatism in adopting a methodology for projects in the organization. Pragmatism could be approached from the viewpoints of managerial criteria and technical criteria. Managerial criteria are applied to the methodology’s support and assistance provided to the management. They include the new methodology’s selection cost and completeness and their impacts on the current business architecture [20-22]. These criteria include [20-23]: (a) cost: different types of expenses associated with the methodology, and (b) domain applicability: this criterion is for applications that the methodology has been developed for. Technical criteria [20-23] include: (a) scalability: scalability could be explained by raising one question: could a methodology or a subset of it be employed for

handling different measures of applications? , and (b) distribution: this measurement criterion is for methodologies that are used in developing distributed systems.

4. Analyzing the Assessment's Results

This section conducts an analytical assessment of the methodologies to determine, through referring to expert views, how much they support each of the assessment parameters. Each assessment parameter will be discussed below.

4.1 Concepts and Conceptions

This parameter has two sub-parameters which include internal features and social features.

4.1.1 Internal Features

(a) Autonomy: autonomy is a key feature of agents that distinguishes them from other entities such as objects. All the three agent-oriented methodologies enjoy this feature, with the level of their support for autonomy ranging from medium to good. They all provide various types of support for the agent's autonomy and integrate actions and facilities into the agent. Besides, the collaboration diagram in ADELFE provides agents with a self-decision mechanism for modeling regardless of the environment and other entities.
(b) Reactivity and purposefulness: this feature is fully supported by ADELFE, because this methodology achieves the goals and implements relevant planning. MASSIVE relatively supports this feature since it has a role model, but AOR does not offer a proper model for covering this feature.
(c) Simultaneous implementation: none of the methodologies definitely supports simultaneous implementation.

4.1.2 Social Features

(a) Teamwork: although the methodologies support all the agents, none of them supports agent groups involved in teamwork. Teamwork is the highest level of cooperation among the agents, in which all members of the team work together to materialize common goals. None of the methodologies has offered a solution for achieving this level of cooperation.
(b) Protocol: ADELFE along with its analyzing protocol, i.e. association class diagram, is clearly ahead of the other methodologies. AOR provides no specific model for displaying the protocols and only offers high levels of interaction between the agents. MASSIVE has some protocols, but has not obviously provided any solutions apart from employing AUML.
(c) Communication Language: this feature is observed in all the three methodologies. Since interaction between the agents is associated with some levels of knowledge, the agents communicate through conversation.

4.2 Modeling Language

This criterion has two sub-criteria including usability and modeling technique. Various parameters under each sub-criterion will be presented and the results will be discussed below.

4.2.1 Usability Criteria

(a) Understandability and clarity: these criteria determine how understandable and clear the symbols are and how good the syntactic combination of the models and symbols is defined. The symbols provided by all the three methodologies are fully understandable.
(b) Eloquence, meaningfulness and competence: the number of static and dynamic models as well as the number of different viewpoints that display the target system is a good yardstick for measuring these criteria. MASSIVE has modeled different aspects of dynamic systems and deals with protocols. ADELFE does not provide a strong support for protocols with dynamic system modeling and only provides some support in the design phase. MASSIVE does not provide various viewpoints about the target system, although symbols in this methodology appear to be suitably meaningful. AOR has models for the dynamic and static aspects of the target system and approaches the system from different angles. The modeling language of AOR is not suitable or meaningful since it does not provide a detailed structure of the agents. Also, AOR is not a viewpoint-oriented methodology.
(c) Ease of use: MASSIVE and AOR enjoy symbols that make them easy to use and understand. This criterion is also linked to symbols' understandability and clarity. ADELFE is an exception in this regard however, because it does not provide support tools contrary to the other methodologies and, therefore, users might find it difficult to draw diagrams and check the compatibility of the models.

4.2.2 Modeling Techniques Criteria

(a) Number of ambiguities: the syntactic combination has been properly defined in ADELFE and MASSIVE. For ADELFE, there is no agreement on the syntactic combination, but its semantic definition has been agreed upon. As for AOR, experts believe that its modeling combination has not been defined adequately.
(b) Compatibility: various methodologies have different levels of checking compatibility. MASSIVE supports this process properly, but ADELFE and AOR do not provide

adequate support, something that could be attributed to the availability of support tools.
(c) Follow-up capability: MASSIVE outpaces the other methodologies with respect to supporting this feature. This methodology creates a clear link between its models; for example, roles, agents and actions are linked to each other. Such links enable the developer to extract design models from design structures (e.g. internal architecture of agents).
(d) Filtration: this architecture is supported by MASSIVE and OAR, but experts do not have the same opinion with regard to ADELFE. It reflects the fact that the modeling language of all the methodologies has not been integrated into their development process. Filtration includes repetitive activities, and developers are free to add details to the model during various phases.
(e) Reusability: none of the methodologies definitely provide techniques to support the designing and use of applicable components. Also, reusability of the existing components of each methodology has not been determined.
In total, the three methodologies enjoy suitable modeling languages in terms of the understandability and clarity of the symbols and are able to explain different (static and dynamic) aspects of the final system. They also have a clear semantic combination that reduces the ambiguity of the modeling language. Anyway, MASSIVE supports several features such as compatibility checking, follow-up capability and clarity, while the remaining methodologies need to develop and incorporate these features and support tools. Also, all the three methodologies could be improved by enabling them to support reusability or create reusable components. Thus, software development productivity increases if the aforementioned features are supported.

4.3 Process Area

Process has four very important criteria that will be discussed below.
(a) Development principles: from the viewpoint of software development life cycle, all the methodologies discussed above enjoy an architecture design. Implementation, testing and troubleshooting phase has only been covered by ADELFE. AOR is the only methodology that does not support agent development. Also, the three methodologies do not support the maintenance and support phase. As for software engineering models, AOR has an incremental waterfall process with repetitive activities in each phase, while the other methodologies have top-down processes.
(b) Process stages: these stages should be clearly defined in analysis and design phases. However, MASSIVE does not support the analysis phase. A general characteristic of the three methodologies is that all of them lack decision management in implementing various stages of the process.
(c) Development support concept: there are several chief concepts of development such as reusability prototyping and reengineering that are not clearly supported by any of the methodologies discussed here. Another important fact that determines generality of agent-oriented models is the degree to which the existing software could mix with old agents or systems. None of the methodologies support this feature.
(d) Assessment and quality assurance guidelines: due to the lack of the evolution of agent-oriented methodologies, issues related to cost estimation through quality assurance is not available in the three methodologies. Therefore, the experiences of software engineers should be consulted in this regard.
In total, the three methodologies discussed above cover the architecture design. ADELFE covers implementation, testing and troubleshooting as well. Given what was said above, all the three methodologies need further development in order to provide guidelines for estimating and assuring software quality.

4.4 Pragmatism

This criterion has two subgroups including managerial criteria and technical criteria. Results related to the parameters of these two subgroups will be discussed below.

4.4.1 Managerial Criteria

(a) Cost: achieving methodologies and their support tools required for knowledge level and current applications as well as their availability is almost free of charge for all the three methodologies. Relevant documents are available.
(b) Domain applicability: there are several domain limits in the main techniques and models. For example, not all of these methodologies are suitable for systems susceptible to conflicts. There are also several assumptions related to the capability of the domain of these methodologies. For instance, non-changeability of architecture structures based on time or non-changeability of their agents and services when implementing open systems are another area that is not covered by the three methodologies discussed here.

4.4.2 Technical Criteria

(a) Dynamic and scalable structure: this parameter has not been clearly specified in the methodologies. Specifically, the methodologies do not deal with how to introduce new components or modules into the existing systems. Besides, none of the methodologies discussed here supports open system designs.

(b) Distribution: in total, the methodologies discussed here implicitly support distribution. The specific levels of distribution stem from the nature of agent-bases systems. In fact, the relationship between agents takes shape with the help of message exchanging systems. Put differently, agents do not link to each other unless an interaction is required.

5. Introducing AMA methodology

An analytical assessment of three selected agent-oriented methodologies was conducted in the previous section. Those methodologies were three important AOSE methodologies according to the results of this study. Their strengths and weaknesses were also extracted based on assessment criteria and methods. In this section a method is presented for the unification of those methodologies through mixing their strengths and avoiding their weaknesses and limitations. In fact, parts of these methodologies were used in order to create a new methodology. A comprehensive AOSE methodology should relatively cover at least requirements analysis, architecture, design, implementation, and testing and troubleshooting phases. The maintenance and support phase still belongs to new agent-oriented methods and is not taken into consideration here. Figure (2) illustrates the proposed methodology's phases.

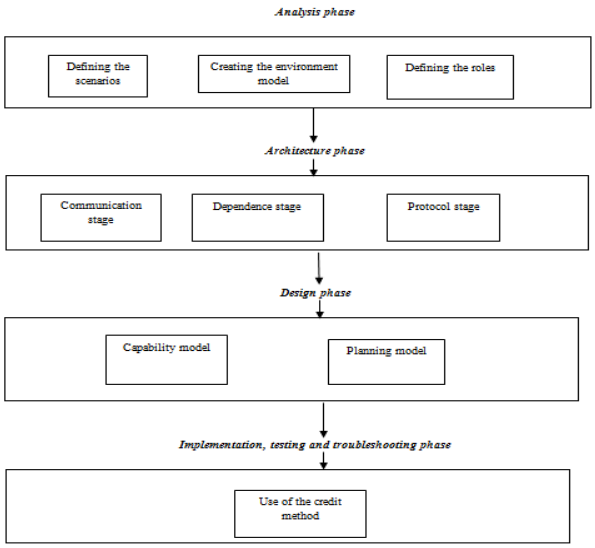

Figure.2 phases of the proposed methodology AMA

Each phase of the proposed methodology will be discussed below.

5.1 Requirements Analysis Phase

This phase has three important stages:

(a) Defining application scenarios: the structure of application cases could be similar to the sequence graph in ADELFE and AOR in which the path linking the system's roles is determined.
(b) Creating an environment model: the environment could turn into a model from the viewpoint of the system and its developers. This viewpoint is similar with the one in MASSIVE which extracts special concepts and organizational relations.
(c) Defining the roles: in the previous stages, the system's analyzer is able to obtain sufficient information for the existing operation in the system to determine key roles. This process, which is similar to the process used in MASSVE, is suitable for defining roles in AMA.
The above-mentioned three stages and the related models provide good support for extracting requirements, identifying the environment, and defining key roles in the system. They improve the developer's understanding of the system's requirements and provide input for the next phase, which is the design phase.

5.2 Architecture Design Phase

The unified methodology follows the system's architecture design in three stages:
(a) The relationship stage: in this stage, the system's actors are extracted and the relationship between them is fully determined. The communication model in ADELFE is a similar technique.
(b) The dependence stage: an important aspect of this stage is that it determines type of agents and the relationship between them. Such information is required when employing resources, implementing actions, or achieving goals. The precedence model in MASSIVE has the same performance.
(c) The protocol stage: in this stage. The system's behavior and interactions could be determined through the diagrams of AUML. AUML's notations are supported by MASSIVE.

5.3 Design Phase

AMA follows the design phase in two stages:

a) Capability model: this model employs UML's activity model from the viewpoint of the agent to model a capability (or a series of relevant capabilities). External incidents are the starting state of the activity model, while internal incidents are its action nodes. The internal and external models in AOR correspond to this model.
(b) Planning model: each planning model is consisted of an action node that can determine additional features through UML's activity diagram. This model is extracted from the previous model.

5.4 Implementation, Testing and Troubleshooting

MASSIVE and AOR have not proposed proper techniques and processes for executing the implementation phase. Since there is a close link between design and implementation phases, models of analysis and design phases could be employed to implement the system in the proposed methodology. Testing and troubleshooting is an important task that should be taken into consideration in the implementation stage. Although MASSIVE and AOR have not provided proper solutions for this phases, the credit existing in ADELFE could be used in the unified methodology.

6. Conclusions and Suggestion

This article introduced a compound methodology for analyzing and designing agent-based systems. The proposed methodology, which has employed a mixture of the strengths of AOR, MASSIVE, and ADELFE, provides the possibility to use high level techniques to handle complexities. Use of a compound solution in the proposed methodology helps to materialize two chief goals: using work-related standards and redefining the main blocks. Future work on AOSE could focus on two areas: 1- assessing developed methodologies through the use of experimental or comparative methods, and 2- developing agent-oriented methodologies through the use of a compound solution and merging methods. Such studies could pave the way for introducing the next generation of agent-oriented methodologies.

References

[1] Zambonelli, Franco, Nicholas R. Jennings, and Michael Wooldridge. "Multi-agent systems as computational organizations: the Gaia methodology." *Agent-oriented methodologies* 6 (2005): 136-171.

[2] Weiß, Gerhard. "Agent orientation in software engineering." *The knowledge engineering review* 16, no. 04 (2001): 349-373.

[3] Tveit, Amund. "A survey of agent-oriented software engineering." In *NTNU Computer Science Graduate Student Conference, Norwegian University of Science and technology*. 2001.

[4] Garcia-Ojeda, Juan C., and Scott A. DeLoach. "agentTool process editor: supporting the design of tailored agent-based processes." In *Proceedings of the 2009 ACM symposium on Applied Computing*, ACM, (2009): 707-714.

[5] DiLeo, Jonathan, Timothy Jacobs, and Scott DeLoach. *Integrating ontologies into multiagent systems engineering*. AIR UNIV MAXWELL AFB AL CENTER FOR AEROSPACE DOCTRINE RESEARCH AND EDUCATION, 2006.

[6] DeLoach, Scott A. "O-MaSE: An Extensible Methodology for Multi-agent Systems." *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks* (2014): 173-191.

[7] DeLoach, Scott A., and Juan C. Garcia-Ojeda. "The O-MaSE Methodology." In *Handbook on Agent-Oriented Design Processes*, Springer Berlin Heidelberg, (2014): 253-285.

[8] Juan, Thomas, Adrian Pearce, and Leon Sterling. "ROADMAP: extending the gaia methodology for complex open systems." In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, ACM, (2002): 3-10.

[9] J. DiLeo, T. Jacobs, and S. A. DeLoach, "Integrating Ontologies into Multiagent Systems Engineering," 4th international bi-conference workshop on agent- oriented Information systems (AOIS 2002), Italy, 2002.

[10] S. A. DeLoach, "Modeling Organizational Rules in the Multiagent Systems Engineering Methodology," Proceedings of the 15th Canadian Conference on Artificial Intelligence, USA, 2002.

[11] A. Self, and S. A. DeLoach, "Designing and Specifying Mobility within the Multiagent Systems Engineering Methodology," Proceedings of the Eighteenth Annual ACM Symposium on Applied Computing, 2003.

[12] Zambonelli, Franco, Nicholas R. Jennings, Andrea Omicini, and Michael J. Wooldridge. "Agent-oriented software engineering for internet applications." In *Coordination of Internet Agents*, Springer Berlin Heidelberg, (2001): 326-346.

[13] Giorgini, Paolo, John Mylopoulos, and Roberto Sebastiani. "Goal-oriented requirements analysis and reasoning in the tropos methodology." *Engineering Applications of Artificial Intelligence* 18, no. 2 (2005): 159-171.

[14] Jureta, Ivan J., Stéphane Faulkner, and Pierre-Yves Schobbens. "Allocating goals to agent roles during mas requirements engineering." In *Agent-Oriented Software Engineering VII*, Springer Berlin Heidelberg, (2007): 19-34.

[15] Wagner, Gerd. "The Agent–Object-Relationship metamodel: towards a unified view of state and behavior." *Information Systems* 28, no. 5 (2003): 475-504.

[16] Lind, Jürgen. *Iterative software engineering for multiagent systems: the MASSIVE method*. Springer-Verlag, 2001.

[17] Picard, Gauthier, and Marie-Pierre Gleizes. "The ADELFE methodology." In *Methodologies and Software Engineering for Agent Systems*, pp. 157-175. Springer US, 2004.

[18] Odell, James. "Objects and agents compared." *Journal of object technology* 1, no. 1 (2002): 41-53.

[19] Lind, Jürgen. "Issues in agent-oriented software engineering." In *Agent-Oriented Software Engineering*, Springer Berlin Heidelberg, (2001): 45-58.

[20] Tran, Quynh-Nhu Numi, and Graham C. Low. "Comparison of ten agent-oriented methodologies." *Agent-oriented methodologies* (2005): 341-367.

[21] Silva, C. T. L. L., P. C. A. R. Tedesco, Jaelson Castro, and Rosa Pinto. "Comparing agent-oriented methodologies using NFR approach." In *IEE Seminar Digests*, (2004): 1-9.

[22] Abdelaziz, T., M. Elammari, and R. Unland. "A Framework for the Evaluation of Agent-oriented Methodologies." In *Innovations in Information Technology, 2007. IIT'07. 4th International Conference on*, IEEE, (2007): 491-495.

[23] Cuesta, Pedro, Alma Gómez, Juan C. González, and Francisco J. Rodríguez. "A framework for evaluation of agent oriented methodologies." In *Proceedings of the Conference of the Spanish Association for Artificial Intelligence*, vol. 147, (2003): 151-152.

First Author received his B.Sc. and M.Sc. degrees from the computer engineering department of Science and Art University, Iran, in 2012 and 2014, respectively. His research interests are

software engineering, agent based systems, data mining, fuzzy systems and distributed system.

Second Author received her B.Sc. and M.Sc. and Ph.D. degrees from the computer Science department of Yazd University, Iran, in 1998, 2000 and 2009, respectively. She is currently an assistant professor in the C. department of Science and Art University, Yazd, Iran. Her areas of interest include software engineering, data base, data mining and fuzzy systems.

Third Author received the B.S. degree in computer engineering from the Shahid Beheshti University in 1999, and the M.Sc. and Ph.D. degrees in computer engineering from Tarbiat Modares University, Iran, 2003 and 2013, respectively. He is currently an assistant professor in the E.C. department of Yazd University, Yazd, Iran. His areas of interest include software engineering, image processing, machine vision and machine learning.