MASRML - A DOMAIN-SPECIFIC MODELING LANGUAGE FOR MULTI-AGENT SYSTEMS REQUIREMENTS

Gilleanes Thorwald Araujo Guedes[1] Iderli Pereira de Souza Filho[1] Lukas Filipe Gaedicke[1] Giovane D'Ávila Mendonça[1] Rosa Maria Vicari[2] and Carlos Brusius[2]

[1]Curso de Engenharia de Software, Pampa Federal University, Alegrete, RS, Brazil
gilleanesguedes@unipampa.edu.br, lukasgaedicke@unipampa.edu.br, iderlisouzaf@gmail.com, giovanedavila@gmail.com
[2]Departamento de Informatica, Federal University of Rio Grande do Sul, Porto Alegre, RS, Brazil
rosa@inf.ufrgs.br, cbrusius@uol.com.br

ABSTRACT

MASRML – Multi-Agent Systems Requirements Modeling Language – is a UML-based Domain-Specific Modeling Language conceived for the requirements modeling in multi-agent system projects. Along this work the extended metamodel developed to support the language is described and the applicability of this DSML in the requirements identification of a multi-agent system is demonstrated using the new mechanisms produced to model specific functional requirements for this kind of system. This work also includes how the DSML was validated and the impressions collected during the validations.

KEYWORDS

UML, Metamodels, Stereotypes, Requirements Engineering, Multi-Agent Systems, Agent Roles, AgentRoleActors, InternalUseCases

1. INTRODUCTION

Throughout the years, researchers in the software engineering (SE) area created software development methods and modeling languages in order to produce software with more quality, better structured, more reliable, and easier to maintain. Now in the artificial intelligence (AI) area, the use of agents as helpers in software applied to several dominions is spreading [4], [14], [23] and the industry interest for multi-agent systems (MAS) is growing up [35]. This kind of software has demonstrated to be a good alternative for the development of complex systems [1], [4], [7], leading to an increase in its development [14].

Nevertheless, developing MAS is complex and brought new challenges to the SE area [4]. These challenges led to the arising of a new sub-area that mixes concepts from both the Software Engineering and the Artificial Intelligence, known as AOSE – Agent-Oriented Software Engineering – whose goal is that of proposing processes and languages to design and to model agent-aided software.

Within this context, some attempts were made to extend the Unified Modeling Language (UML) – a modeling language largely used in the SE – to be applied in multi-agent systems (MAS) projects, such as [3] [9], [10], [32], [36]. In our research, though, we realized that some UML resources were underutilized and could be better adapted and applied for the MAS needs, such as the use-case diagram (UCD), which allows to model the system functional requirements and who

can access these functionalities. This diagram, therefore, is mainly employed in the requirements engineering, an essential process for the achievement of a good software project.

Nevertheless, the UCD does not support the representation of concepts like software agents or agent roles, and it is equally unable to model specific requirements for MAS, like goals, plans, perceptions, or actions – internal behaviors associated to the roles interpreted by agents. Thus, considering the importance of the requirements engineering for a good software project and the lack of mechanisms for MAS functional requirements modeling in the UCD, we developed a UML-based DSML (Domain-Specific Modeling Language) specifically to address this issue.

The choice of UML as the base language from which we extended our DSML metamodel is justified because it is a language widely accepted and understood in the SE area. Since UML concepts are very spread out and understood, we believe it would make easier the learning and understanding of the innovations introduced by our language and its application would be more intuitive by whom already had used UML. Thus, we adapted and extended the metamodel in which the UML UCD is based, since this diagram is used mainly for the identification of which functional requirements the system will need and who will access them.

In this metamodel, we created new metaclasses and stereotypes to represent MAS requirements. Our objective was to adapt the UCD to be used to identify the internal functional requirements of a MAS, such as: which agent roles must be supported by the software; which are the goals associated to these roles; what conditions must be satisfied so that these goals can become intentions; what plans should exist for these goals to be achieved; which are the perceptions needed for each agent to have when it takes up a given role; and which are the external actions that will be allowed for a specific role.

This paper is organized as follows, first we introduce some conceptual definitions. Next, we discuss the related works. After we present our DSML, describing its metamodel. Following, we present a case study describing an extended UML UCD produced by the application of our DSML to identify particular requirements of a MAS. Next, we discuss how we had validated our work and finally we present our conclusions.

2. BACKGROUND

2.1. Agents, Multi-Agent Systems, and The BDI Model

An agent is a process situated in an environment, designed to achieve a purpose through an autonomous and flexible behavior. The environment is the application domain where the agent will work [11]. We adopt the BDI (Belief, Desire, Intention) cognitive model of [13], who states that the purpose of an agent can be fully specified by the definition of its beliefs and desires, and that its behavior is implied by its intentions.

Beliefs represents the agent knowledge about itself, about other agents and users, and about the environment. Agents can update their beliefs to reflect changes detected in the environment. Desires specify the set of state of belief (the goals) which the agent wants to achieve. Intentions represent the set of desires that an agent believes can be achieved. When an agent believes a desire can be achieved, it becomes an intention and the agent will act to achieve it.

An agent relates with the environment by means of actions and perceptions. An action is a change caused directly by the agent. An agent must have knowledge about what its actions would cause in the environment, and how its actions are related to its intentions. A perception is an information about the environment and/or about the users and other agents received through one or more of the agent sensors. This information can change the agent beliefs.

A multi-agent system (MAS) is composed by a set of interacting agents. In MAS, many agents may assume the same role at the same time. When an agent assumes a role, it becomes committed with the goals assigned to the role.

2.2. UML, Metamodels, Models, Stereotypes, and DSMLs

UML is a standard language for specifying, constructing, and documenting system artifacts [38]. It provides tools to be used in the analysis, design, and implementation of software [39]. The UML specification is defined by metamodeling. There is a difference between metamodels and models, the first defines the semantics for the way of modeling elements within a model, while the last represents a vision of a physical system, as it is an abstraction of the software with a purpose that determines what must be included and what is irrelevant [38], [39].

Within the context of metamodeling, there are the stereotype metaclasses. These metaclasses extend and assign new characteristics and/or constraints to other metaclasses and can only be used in conjunction with the metaclasses they extend. When a metaclass uses the stereotype named, “stereotype”, it means that the metaclass was derived from another metaclass and that it is possible to model stereotyped elements in a model in a way that differentiates them from the original elements. However, this new element cannot suppress the original features inherited from ancient elements [38], [39].

A DSML (Domain-Specific Modelling Language) provides primitives to express higher-level abstractions and constraints of a targeted domain, promoting productivity by the reusing of domain concepts and quality by carrying integrity constraints [27].

2.3. Requirements Engineering and UML Use-Case Diagram

Several authors state that a correct requirements specification is fundamental for a software project to be successful [12], [15], [16], though they highlight that this is not an easy task. Thus, requirements engineering involves all the system life-cycle activities dedicated to identify user requirements, to analyze requirements to derive additional requirements, to document requirements, and to validate the specified requirements [12].

Regarding the requirements modelling, many authors recommend to employ the UML Use-case diagram (UCD) [12], [15], [16]. Use cases are a way to capture systems requirements. UCDs identify subjects, use cases, and actors as key concepts. A subject represents the system modeled. Use cases represent the external requirements of a subject. Each use case specifies a functionality provided by the subject to its users. Actors represent external entities that interact with the subject and access its functionalities [38].

2.4. Related Works

We retrieved the related works by means of a systematic mapping and a systematic review[1]. In the mapping, we retrieved works that either proposed MAS methodologies, adapting or, at least, using UCDs for their purposes, or works that, though they do not define a complete methodology or language, adapted the UCD for representing MAS requirements. In the systematic review, we searched for languages derived from UML specifically to the MAS design that extended the UML UCD.

Among the MAS methodologies, Prometheus [8], ADELFE [17], OSOAD [18], TROPOS [22], and MASSIVE [21] apply the UML UCD without specifying agents or its internal behaviors.

[1] Mapping protocol: https://drive.google.com/file/d/1xbG7LNAiCiwWh8hBtM_L0DRrJnummly8/view?usp=sharing; Review protocol: https://drive.google.com/file/d/1TzUUannt8u-WBIgo60ecqbg1qWesel6w/view?usp=sharing.

PASSI [19], ASPECS [20], and MAS-CommonKADS [31] somehow try to adapt UCD to specify agents or the particular requirements for these agents.

PASSI [19] applies UCD to identify normal use cases and actors. The use cases are grouped in UML packages representing agents. Each package-agent encompass the use cases under its responsibility. PASSI does not represent the agents' internal behaviors.

ASPECS [20] produces a UCD, identifying normal actors and use cases. The use cases are grouped in packages with the stereotype <<organization>> in order to determine the agent roles contained in the organizations and the interactions of these organizations. However, the actors and use cases, are normal ones.

MAS-CommonKADS [31] represents agents as square head actors associated to internal use cases. However, it does not define what kind of behavior these internal use cases represent (like goals). Still there is no metamodel to represent the agent or internal use case and in the example, though there are two internal agents interacting with use cases, these are not internal use cases, just normal ones accessed by a normal actor.

Regarding the languages derived from UML to the MAS design that adapted UCD, we identified Depke's proposal [3], M-UML [9], AML [10], AUML [32], and AMOLA [36]. Depke in [3], extended UCD by creating a square head actor to model agents inside the system and a special use case, named goal case, to specify the agents' goals. However, there is no way to differentiate reactive agents from cognitive ones and there is not a representation of perceptions associated to the goals so that they became intentions, nor any representation of plans associated to the goals. Still, there is no metamodel depicting the new concepts.

M-UML [9] models mobile agents and its functionalities through actors and use cases with an "M". However, these symbols only could be used if there were stereotypes derived from the actor and the use-case metaclasses and this metamodel is not presented. Moreover, the mobile use cases are just normal ones that mobile agents interact with.

Regarding AML, Trencansky in [10] states that it is possible to apply mental states to the use cases through requirements representation based on goals, but without presenting examples. There is an example in [5] of an AML extended UCD in which actors represent agents, placed inside the system, while use cases represent functionalities associated to the agents. However, these use cases do not differ from normal ones, since they do not represent behaviors of agents, only normal functionalities the agents should execute. Anyway, as use cases represent functionalities that must be accessed by external actors, they cannot be used to represent internal functionalities. Still, we did not find any extension of the use case metaclass and even if it exists, the semantics of a use case cannot be modified in an extended metaclass.

Laouadi in [32] extended AUML by creating 3 stereotypes of Actor and 2 of Use Case. The actor stereotypes represent agents within the system, external agents, and real time agents, while the use case stereotypes model functionalities performed by agents and functionalities in which real-time agents interact. The stereotyped use cases, however, only represent functionalities the agents should perform, differentiating little from normal use cases, not explicitly representing agent internal behaviors. Besides, no metamodel was presented containing the new stereotypes and, in our opinion, these stereotypes infringe some rules of the Actor and UseCase metaclasses.

In a more recent work, Laouadi in [33] proposed to extend the UML UCD itself to model MAS and real-time MAS. The author created stereotypes to represent, functionalities executed by agents and functionalities executed by real-time agents. Stereotypes were also created to describe

agents within an organization, external agents, and real-time agents. Our critic to this work are the same of the previous one.

The AMOLA language [36] makes not any adaptation of the UML use-case diagram for MAS requirements representation; however, it uses UML Actor to represent agents placing them inside the system boundaries. Besides, it assigns normal use cases to these agents. As we had stated before, actors cannot be applied this way.

With relation to works that, though they do not define a complete methodology or language, had adapted or, at least, applied the use-case diagram for representing MAS requirements, we found some works that we discuss next.

Heinze in [30] and Papasimeon in [34] created stereotypes to represent internal agents and use cases accessed by the agents. Heinze recognizes that there is a difference between actors and agents, since agents are internal and actors are external, however he argues that stereotypes are enough to solve the problem. We disagree, since it is not correct to create a stereotype that suppresses original features of the concept that is being stereotyped.

In Flake et alii in [25], agents represent active objects and are modeled as square head actors inside the system. There were also created stereotypes to use cases to represent agent's goals and reactive actions, besides normal use cases are associated to the agents as they were internal use cases. However, we did not find any metamodel representing the stereotypes. Moreover, we believe that normal use cases cannot be associated to agents, since external actors must access them and we do not believe that use cases could be extended to represent internal functionalities, since this infringes UML rules.

Collier in [24] created two stereotypes to identify roles assumed by agents and use cases associated to agents. Hamidane in [6] also created two stereotypes for representing agents as square heads actors and use cases accessed by agents. These use cases, however, do not identify the kind of behavior they represent (like goals). Neither [6] nor [24] demonstrated any metamodel. Again, we believe simply to extend these metaclasses to represent role agents or the internal behaviors associated to them is not enough.

Rodriguez et alii in [23] represent agent roles employing actors with the <<role>> stereotype. This authoress employs stereotyped normal actors, so they cannot be used to represent agents (or their roles) that are part of a system, only external agents. Moreover, [23] does not differentiate reactive and cognitive roles and uses only simple stereotyped associations to determine when a resource can be perceived or modified.

Analyzing these works, we realized that only [14], [19], and [20] had created a metamodel to represent the new concepts proposed for MAS requirements modeling. Regarding to the adaptation of the UML Actor for representing agents or agent roles, except for AML, all the other works simply created stereotypes to be applied up on UML actors and placed them inside the system boundaries, without creating new metaclasses to represent them nor providing any information about how these extensions were made.

Moreover, for a UML metaclass to be employed in a way it was not designed to, it is necessary to extend this metaclass creating a new one with new features, but without suppressing the original constraints. Therefore, the choice of creating stereotypes for the Actor metaclass to represent agents or agent roles would not be entirely correct, since the actors represent external entities to the system, as is stated in [38], and it is not possible to remove any constraint from an original metaclass when a stereotype is applied to it.

Therefore, the extended stereotyped actors are infringing a rule established to the Actor metaclass. To allow an actor to be placed inside the system, it would not be enough to extend the actor metaclass but to create a new one from primitive UML metaclasses that were not subordinated to this rule. Furthermore, we did not find in any work a way to differentiate reactive agents or roles from cognitive ones, since these kinds of agents own different features and we believe it can be necessary to detach these differences in environments composed by heterogeneous agents.

Regarding the use case adaption to represent internal functionalities accessed by agents or their roles, we also believe it is not correct, since use cases represent functionalities that must be accessed by external actors, so they cannot be used to represent internal functionalities that external actors cannot access. Thus, the works that simply apply use cases to represent agents' internal behaviors are also infringing UML rules.

3. THE DSML

As regards the justification to develop a DSML exclusively turned to the functional requirements modeling for MAS, we believe that the requirements of this kind of software have particular features, since in MAS there are usually several behaviors assigned to agent roles. These behaviors are internal processes, which cannot be accessed by normal external users who do not even know about their existence. These behaviors contain steps necessary for desires (goals) to become intentions, for plans to be executed, to allow an agent to sense and to act in the environment, etc. Therefore, we consider important and necessary to identify these particular requirements in MAS projects.

Werneck et alii corroborate this in [37] stating that the increase in the use of MAS has generated challenges that were not studied until then, including the way of how to adapt the requirements elicitation to deal with agent properties as autonomy, sociability, and proactivity. Papasimeon et alii in [34] add that SE on MAS demands the specification of those agent behaviors needed to provide documented requirements to the project and implementation phases. Rodriguez et alii in [23] also state that the requirements modeling in MAS requires abstractions, techniques, and notations that had been particularly adapted for this kind of domain. Nakagawa et alli in [7] complement highlighting that roles are the fundamental components of MAS and the extraction of these roles is one of the most important tasks based on the requirements description.

Moreover, Slhoub et alii in [35] state that there is a real demand to develop high-quality MAS requirements specification, highlighting that the complexity associated with MAS development and their target application domains often drives the complexity of the MAS analysis phase and results in poor quality requirements specification artifacts. Such a problem can significantly affect the entire MAS development and create further technical problems to stakeholders.

Considering that UML is a standard language widely accepted and understood and that agents composing MAS have particular requirements that are not supported by the UML UCD, we created a DSML derived from UML to address this issue. Thus, we took the metamodel in which is based the UML UCD and we added new metaclasses and stereotypes to it in order to identify the particular requirements of this kind of system.

Initially we intended to develop only a UML profile extending the original metaclasses used by the UCD. Therefore, in a first work [28], we extended the Actor metaclass to represent agent roles, since, according to [38], this metaclass represents a kind of role interpreted by an entity that interacts with the system but is external to it. However, most of the times, the agents are not external to the software, instead, they usually are inserted in the system environment. Thus, we

realized that it was necessary to adapt this concept, considering that agents are internal to the system, i. e., they belong to the software.

Nevertheless, it is not possible to take off any constraints applied to a metaclass from an extended metaclass. Therefore, instead of extending the Actor metaclass, we decided to create a new metaclass, that we named AgentRoleActor. This new metaclass was derived from the BehavioredClassifier metaclass, the same metaclass from which the Actor metaclass had been derived. In this new metaclass we copied all the features of the Actor metaclass, including the same symbol (to allow compatibility with CASE tools), but suppressing the constraint that an actor must be external to the system, instead an AgentRoleActor must be internal.

Figure 1 shows the metamodel we extended from UML to support our DSML. A previous version of this metamodel was already published in [29], but in this work, we are presenting the last version of this metamodel, not yet published.

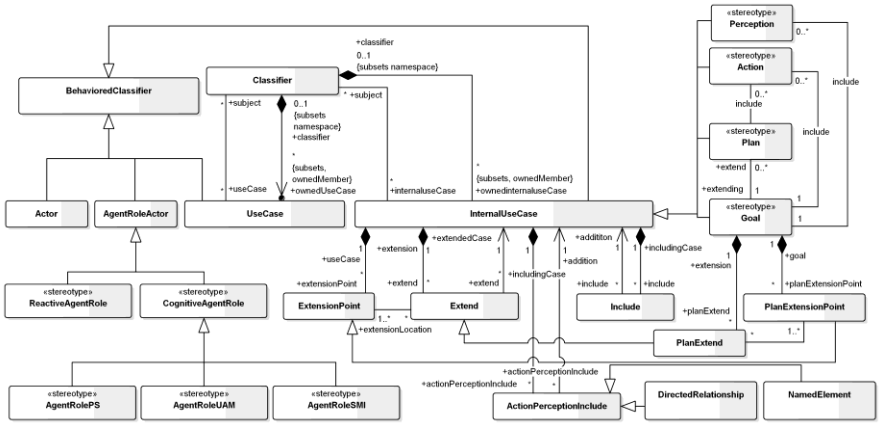

Figure 1 The DSML Metamodel for Requirements Modeling in MAS Projects

From Agent Role Actor we derived the Reactive Agent Role and Cognitive Agent Role metaclasses to represent roles interpreted by reactive and cognitive agents. We applied the stereotype named “stereotype” in both metaclasses, which means that they will be applied as stereotypes on AgentRoleActors and will attribute special features to them. We created these stereotypes to establish a difference between reactive and cognitive agent roles, since each group presents different features.

We specialized the CognitiveAgentRole metaclass by deriving the AgentRolePS, AgentRoleUAM, and AgentRoleSMI metaclasses. These metaclasses were inspired in Vicari and Gluz [11], who suggested the use of specialist agents Problem Solving (PS), Users and Agents Modelling (UAM), and Social Mediated Interactions (SMI). PS agents have knowledge about the problem focused by the application. UAM agents are able to make cognitive models of the users and other agents that interact with the system. Finally, SMI agents have knowledge about how to mediate social interactions between the user and the system.

It is important to highlight that our approach is different from the works previously described. In our work we created the concept of AgentRoleActor to represent agent roles internal to the system and we specialized this concept into more specific agent roles. We did not extend the Actor metaclass, we created a new metaclass with similar but not equal characteristics. The previous works tried to extend the UML Actor concept to represent agents, which, in our opinion infringed UML rules, since it modified the actor characteristics, moreover, the huge majority of

the works did not present a metamodel showing how the extension was made. Some works did not even make an extension, simply using normal actors to represent agents.

Regarding use cases, [12], [15], and [16], defend its application in requirements engineering as being very useful to identify, analyse, document, validate, and track system requirements. According to [39], use cases can be used to specify the system's external requirements. A use case specifies a set of steps performed by the system. Its execution generates an observable result that has some utility for one or more actors.

Aiming to adapt the use case concept for MAS requirements modeling, in order to represent agent internal behaviors, we derived the metaclass InternalUseCase (IUC) from the same metaclass that UseCase was derived. All IUCs are internal to the system, thus, external entities do not have knowledge about them and cannot use them. We also created relationships equal to those existing between the UseCase metaclass and the metaclasses Include, Extend, and ExtensionPoint for IUC metaclass, so all the associations allowed to normal use cases are also compatible with IUCs. It should be noted that we maintained the same symbol of the UseCase metaclass for the sake of compatibility with the existing CASE tools.

From the InternalUseCase we extended some metaclasses to be employed as stereotypes and to attribute special features to the IUCs. Thus, to model IUCs containing the necessary steps for an agent to perceive or to perform a given external action, like receiving or sending a message to another agent, we created the metaclasses, Perception and Action.

A third stereotype, Goal, was derived from the IUC metaclass, to represent goals. According to Dam in [2], the concept of goal is one of the most important agency concepts, and it contributes to the agents' pro-activeness. This metaclass represents the desires an agent wishes to achieve. The documentation of an IUC with the Goal stereotype will contain a description of an agent's desire and the conditions for that desire to become an intention. Two previous works [3] [25] also represented use cases as agent goals, though none of them had created a metamodel as we did. Furthermore, we do not consider enough, nor even correct to extend the UseCase metaclass for this purpose.

For a goal to be achieved, normally there must be one or more plans to accomplish this goal. Considering that a plan associated to goal will be executed only if the goal become an intention, we derived a fourth metaclass from the IUC metaclass, called Plan, to identify the plans associated with the goals of an agent role.

Since a plan is only triggered after some condition is satisfied, we decided to derive the metaclass, PlanExtend, from the Extend metaclass and associate it with the Goal metaclass. The Extend metaclass represents an association between use cases which specifies how and when the behavior of the extended use case can be inserted in the behavior defined in the extending use case upon a condition [39]. We chose this way to differentiate normal Extend associations from PlanExtend associations, thus a planExtend association turns clear that the extension refers specifically to a behavior extension of a goal by the behavior of a plan.

Finally, we derived the metaclass, ActionPerceptionInclude from the same metaclasses that the Include metaclass was derived. We proceeded this way because the semantics of the Include metaclass states that it is intended to be used when there are exactly similar behavior excerpts in the behavior of two or more use cases [39]. Agent roles can share perceptions and actions, but not always, so we created this new metaclass containing the same characteristics of the Include metaclass. However, we suppressed the constraint that limited the use of this new metaclass to

common behaviors in more than one IUC. Its function is to include the behavior of perceptions and actions in one or more goals.

Thus, since a goal to become an intention needs a condition to be satisfied and the agent must perceive that this condition was satisfied, when modelling a goal, we associate one or more IUCs with the perception stereotype to the IUC with the goal stereotype by means of the actionPerceptionInclude (aPI) association. It means that, when the behavior of the Goal is to be executed by an agent, the behavior of the Perception will be executed too. How it is performed is contained in the goal IUC documentation. In the same way, as the plans associated to a goal only are executed when the goal becomes an intention, we associated the Plans to the Goal by means of planExtend associations, to highlight that the plan behavior only will extend the goal behavior when a condition be satisfied.

Again, we highlight that we created the concept of Internal Use Case. Differently from the studied works, we did not extend the metaclass UseCase, we created a new one with similar but not equal features. We specialized this metaclass for it to be able to model agent behaviors like goals or perceptions. Some few works we have analysed tried to adapt the UseCase concept to represent internal functionalities associated to the agents (basically goals), nevertheless, we consider that it is not correct, since a use case represents a functionality that must be accessed by external entities while agents are internals ones. Moreover, external users do not have knowledge about the internal functionalities associated to the agents. Still, almost none of these works presented a metamodel depicting how the UseCase metaclass was specialized.

Regarding a CASE tool support, we do not develop a specific tool for our DSML yet. Nevertheless, many tools are suitable to produce our models, since we avoided to create new symbols to our metaclasses, adopting the traditional symbols of the Actor and UseCase metaclasses. Particularly the Enterprise Architect tool proved to be rather adequate, since it permits to define new stereotypes and to edit the old ones.

4. A CASE STUDY – THE HERACLITO ENVIRONMENT

Heraclito is a MAS that aims to help in the propositional logic learning [27]. It allows the students to choose exercises (hypothesis) and tries to help them in its solving. The exercises owns different complexity levels and the simple choice of a given level exercise impacts in the student model. This environment owns three agent roles, StudentModelling, Mediator, and Specialist. Though this system was already developed, in this case study we are focusing in its new version that will add some new functionalities and modify some of the old ones.

The agent who assumes the StudentModeling role greets and introduces himself to each student who logs in the system. The goal of this role is to create a model of each student interacting with the system. This model is built from the attitudes (or the lack of them) of a student to solve exercises. From the choice of an exercise and from the correct or incorrect application of deduction rules by a student, the agent evolves the student model and sends it to the Mediator.

The Mediator role represents the function of a teacher, selecting teaching-learning strategies, based in the student model, to aid the student in its reasoning process during an exercise solving. Each time a student executes an operation in the problem solving, the Mediator forwards it to an agent assuming the Specialist role, who analyses the operation. Depending on the Specialist answer and considering the student model, the Mediator send approving messages, error messages, or offers help.

The Mediator interacts with the student when: (I) the student remains much time idle, in this case the Mediator offers help; (II) when the student himself asks for help; (III) when the Specialist informs a student action is correct or incorrect (in this case the Mediator must determine which is the better strategy to be taken, based on the student model, to aid or to boost the student); (IV) when the student achieves a significant percentage of the problem solving and when just one step is left to end the test. In these cases, the Mediator must boost the student to conclude the exercise.

When the student asks for help, the Mediator must decide, based on the student model, among (I) to provide a tip, indicating which is the next step to be taken to resume the solving process; (II) to present an example suitable to the problem the student is trying to solve; and (III) how many steps are left to finish the exercise.

The agent who takes the Specialist role has for its goal to evaluate the student actions. When the action refers to an exercise choosing, the agent must solve the problem until the end. When the action refers to a rule application, the Specialist must verify whether the rule is correct or not. Every time a student applies a rule in the solving of a problem, this event is perceived by the StudentModeling, who forward it to the Mediator, who, by its turn, forwards it to the Specialist. The Specialist verifies if the student action is correct, wrong, or if it is irrelevant. With this information the Mediator, considering the student model, decides how to interact with him.

Figure 2 shows an extended UCD, produced by means of our DSML, representing the Heraclito functional requirements. We modeled an Actor, named Student, to represent the students who operate the system. We placed this actor outside the system boundaries since this is a normal actor. After, we identified the functionalities this actor can access ("Log On", "Solve Exercise", and "Ask for Help"). As external actors can access these functionalities, we modeled them as normal use cases. The use cases "Log On" and "Ask for Help" are self-explanatory. The use-case, "Solve Exercise" represents the process where a student selects an exercise and solves it.

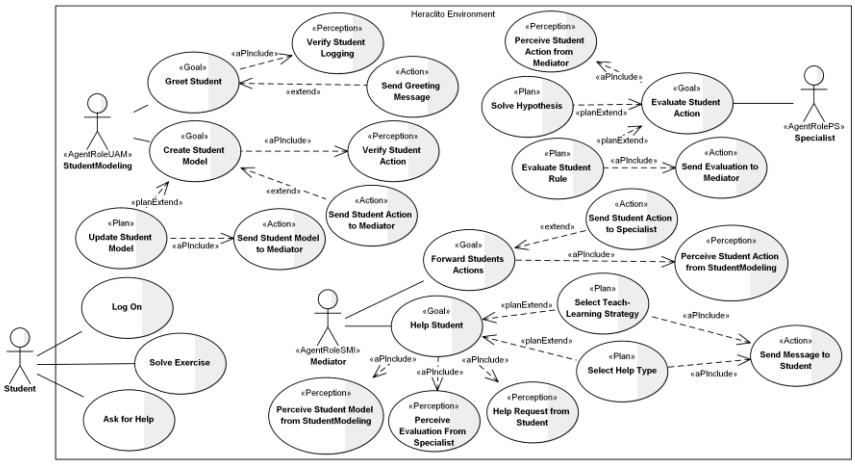

Figure 2 Extended Use-Case Diagram for Heraclito

Next, we modeled the agent roles, StudentModeling, Mediator, and Specialist. We assigned to the first the stereotype <<AgentRoleUAM>>, since the agents who assume this role must have the knowledge of how to model the students' profiles. Mediator received the stereotype AgentRoleSMI, since it interacts with the students. Finally, Specialist received the stereotype AgentRolePS, since the agent who takes this role must have the knowledge to solve the problems. We identified two goals to the agent role, StudentModeling, to greet the students and to create a student model to each student interacting with the system. We modeled these goals as two IUCs

with the <<Goal>> stereotype and associated them to the role by means of normal use case associations (lines linking the actor with the use cases).

For the first goal to become an intention, it is necessary that the agent who takes the StudentModeling role perceives when a student logs on. We modeled this behavior as an IUC with the stereotype <<Perception>> to make clear that this behavior represents an environment probing. As this perception complements the Goal behavior for greeting a student, we associated this behavior to the goal through an actionPerceptionInclude (aPInclude) association, meaning that the execution of the Goal behavior implies in the execution of the Perception behavior as well. The goal documentation defines the moment in which this complementation will occur and under which conditions.

Moreover, if this goal became an intention, the agent will act to greet the student, resulting in sending a message in the environment so the student can see it. As it is an action that affects the environment and can be perceived by users or other agents, we modeled it by an IUC with the <<Action>> stereotype, highlighting that this behavior contains steps for an agent to affect the environment. We associate this action to the goal (since it complements the goal) by means of a normal extend association, signifying that the action behavior will only complement the goal behavior when a condition be satisfied. We proceeded this way because the agent only will greet a student when he realizes the student has logged in. We did not consider necessary to create a plan associated to this goal because it is not necessary a process solely to greet a student.

The second goal (create student model), becomes an intention when the agent perceives a student action (or the lack of one after a given time). Thus, we modeled this perception as an IUC with the <<Perception>> stereotype and associated it to the goal through an aPInclude association. If this perception is true, automatically the goal forwards this information to the agent who takes the Mediator role. Since it is an action perceived by other agent we modeled it as an IUC with the <<Action>> stereotype and associated it to the goal with a normal extend association, to represent that the goal behavior only will be complemented by the action when the condition be satisfied.
Furthermore, when the goal becomes an intention, it also triggers a plan to update the student profile, thus we represented it as an IUC with the <<Plan>> stereotype and associated it to the goal by means of a planExtend association, to make clear that the plan it represents can complement the goal behavior when a condition be satisfied. As the execution of the plan implies in an action perceived by the Mediator, we represent this action as an IUC with the <<Action>> stereotype and associated it to the plan.

The agent role, Mediator, by its turn, has two goals, the first is a very simple one that is to forward all student actions to the Specialist. The second is that of helping a student. It is represented as an IUC with the <<Goal>> stereotype. There are three perceptions associated to this goal, since the agent who takes this goal must perceive when a student model is sent from StudentModeling; when an evaluation is sent from Specialist; and when a student asks for help. Thus, we represented all these perceptions as IUCs with the <<Perception>> stereotype and associated them to the goal through aPInclude associations. Each one of these perceptions triggers an alternative scenario of the Help Student goal, as will be detailed in the documentation of this goal.

The perception of a student model is needed to enable the other alternative scenarios execution. The perception of an evaluation from the Specialist or of a request for help from a student triggers the plans “Select teach-learning strategy” and “Select help type”, associated to the goal through planExtend associations. These plans generate messages sent to the student, whose content will depend on the student model.

Concerning the Specialist role, its goal is to evaluate the student’s tests. For this goal to become an intention, the agent interpreting the Specialist role must perceive a student action forwarded

from the Mediator. When it occurs, there are two possibilities, if the action refers to the selection of a hypothesis, then the agent must solve it. Otherwise, if the action refers to a rule application to solve the exercise, then the agent must verify if, (I) the rule helps in the hypothesis solving; (II) the rule does not impact in the solving; or (III) the rule is wrong. This analysis is sent to Mediator and he will use it to select a strategy suitable to the student profile. It can be noticed that we used IUCs with goals, perceptions, plans, and actions stereotypes to represent these behaviors.

5. INTERNAL USE CASES DOCUMENTATION

The requirements modeled by use cases represent system behaviors. However, the steps of these behaviors are not graphically represented in the use case diagram. The scenarios and steps of a use case are usually detailed in its documentation. In this section, we present a way of documenting our internal use cases. We took as an example the Mediator goal, “Help Student” as can be seen in the Table 1.

Table 1. Mediator Goal, “Help Student”, Documentation

Internal Use Case Name	Help Student
Stereotype	Goal
AgentRoleActor	Mediator
Summary	This internal use case describes the steps followed by the agent who assumes the Mediator Role to help a student while solving an exercise
Perceptions	Student Model from StudentModeling
	Rule Evaluation From Specialist
	Help Request from Student
Main Scenario	
AgentRoleActor Actions	
1. Execute Internal Use Case, "Perceive Student Action from StudentModeling"	
2. Execute Internal Use Case, "Perceive Evaluation From Specialist"	
3. Execute Internal Use Case, "Help Request from Student"	
Alternative Scenario I – Student Model from StudentModeling Perceived	
AgentRoleActor Actions	
Enable the Execution of the alternative scenarios II and III	
Alternative Scenario II – Evaluation From Specialist Perceived	
Pre-Conditions	The student model must have been perceived
AgentRoleActor Actions	
Execute Internal Use Case, "Select Teach-Learning Strategy”	
Alternative Scenario III – Help Request from Student Perceived	
Pre-Conditions	The student model must have been perceived
AgentRoleActor Actions	
Execute Internal Use Case, "Select Help Type”	

In this documentation, we identified the IUC title, the stereotype applied on it, the main agent role actor that interacts with the IUC, a summary of the IUC function and, as it represents a goal, the perceptions necessary to it become an intention.

In the main scenario of this documentation, all the perceptions associated to this goal are executed in sequence. As this goal is associated with three perceptions, it means that the behavior of these perceptions will be included in the behavior of the goal. Thus, the execution of the main scenario basically senses the environment to verify the reception of a new or updated student model; an evaluation from the Specialist; or an request for help from a student. Always any of these perceptions be true, an alternative scenario will be performed. Thus, if a new student model is perceived, the first alternative scenario is executed, which enables the execution of the other alternative scenarios (when the necessary perceptions occur), since a student model is needed to their executions.

The second alternative scenario will be executed when (and if) an evaluation from Specialist is perceived. When this condition is true, it is triggered the plan "Select Teach-Learning Strategy" where the Mediator will decide how to boost the student.

The execution of the third alternative scenario will occur when a student asks for help. It will trigger the plan, "Select Help Type". The execution of this plan considers the student model to choose about which kind of help message the agent will send.

6. VALIDATION

We believe that our DSML allows modelling specific requirements for multi-agent systems. We created mechanisms for representing the agent roles and which internal functionalities these agent roles can execute in the system. In order to verify whether our DSML is adequate to this task, we applied it in three different occasions, in a scientific event mini-course, in an undergraduate subject, and in a master degree subject. Except in the master's degree subject, where all students ensured they knew how to apply UCD to model requirements, we revised the concepts of this diagram and show examples of how to apply it, though the students are supposed to already own previous knowledge about this diagram.

After it, we presented and explained the DSML metamodel and we presented some examples of extended use-case diagrams representing MAS particular requirements. After that, we presented some exercises based in real MAS already developed, so that the students produced extended use-case diagrams using our DSML.

After the exercises correction, we asked the students to fill a form with three multiple-choice affirmations and an open question. The affirmations had a Likert-scale with five options: totally agree, agree, I do not agree nor disagree, disagree, and totally disagree. The affirmations were "The metamodel adequately expresses the concepts it represents"; "The metamodel provides effective support to the MAS requirements modelling"; "The metamodel covers the minimal concepts needed to the MAS requirements modelling". The open question asked the students about what difficulties did they found.

In the three validation sessions, a total of 21 students agreed to evaluate our DMSL. The opinions about the three Likert affirmations are contained in the Table 2.

Analysing this table we realized that regarding to the first affirmation, 33,33% of the respondents totally agreed, while 66,66% agreed. With relation to the second affirmation, 19,05% totally agreed, while 80,05% agreed. Finally, concerning the third affirmation, 52,40% totally agreed, while 47,60% agreed.

Table 2. Opinions on the Affirmations – First DSML Evaluation

Affirmation	Totally agree	Agree	I do not agree nor disagree	Disagree	Totally disagree
The metamodel adequately expresses the concepts it represents	**7**	**14**	**0**	**0**	**0**
The metamodel provides effective support to the multi-agent systems requirements modelling	**4**	**17**	**0**	**0**	**0**
The metamodel covers the minimal concepts needed to the multi-agent systems requirements modelling	**11**	**10**	**0**	**0**	**0**

Regarding the open question, 5 respondents stated they had no difficulties using the DSML; 7 said that they had some initial difficulties, since it was the first time they used the DSML, but managed to solve the exercises rightly; 2 claimed some problems in using CASE tools; 3 found the learning curve very high, considering their little initial knowledge; one stated that the models produced can get a little polluted visually; another one claimed some difficulty to understand some of the stereotype acronyms; another one to remember the metaclasses names; another one in understanding which stereotypes to apply in determined situations; another one in how to specify agents; another one in how to specify agents and perceptions; and the last in understanding that plans were goals extensions and how to associate the plans to the goals.

A second validation was made with a group of 5 master's degree students. We presented the DSML metamodel and show some application examples. After it, we presented a textual description about Heraclito [27]. Based on this description, the students should identify the system requirements using our DSML. They should identify both normal actors and use cases and the agent roles and their internal behaviors. After producing the model, the students were asked to fill another form, where we added two more Likert affirmations (the metamodel is easy to learn and the metamodel is easy to apply) and one more open question about any suggestion to improve the metamodel. The Table 3 shows the results collected in this experiment.

Table 3. Opinions on the Affirmations – Second DSML Evaluation

Affirmation	Totally agree	Agree	I do not agree nor disagree	Disagree	Totally disagree
The metamodel adequately expresses the concepts it represents	**3**	**2**	**0**	**0**	**0**
The metamodel provides effective support to the multi-agent systems requirements modelling	**3**	**2**	**0**	**0**	**0**
The metamodel covers the minimal concepts needed to the multi-agent systems requirements modelling	**3**	**2**	**0**	**0**	**0**
The metamodel is easy to learn	**1**	**4**	**0**	**0**	**0**
The metamodel is easy to apply	**1**	**3**	**0**	**1**	**0**

Analysing the table we realized that regarding the first three questions, 60% of the respondents totally agree with the affirmations and 40% agree. With relation to the fourth affirmation that the metamodel is easy to learn, 20% totally agree and 80% agree. Finally, with relation to the fifth affirmation, 20% totally agree with, 60% agree, and 20% disagree that the metamodel is easy to apply.

Regarding the first open question about which difficulties they had found using the metamodel, 60% stated they did not had any difficulty; 20% said to have some difficulty to know where to start; and 20% declared some difficulty to identify plans from the enunciation, but not in the using the DSML.

With relation to the second open question about suggestions to improve the metamodel, only two suggestion were collected, the first suggestion was to define a set of guidelines to help in the models creation, while the second suggestion was to develop a CASE tool to help in the models development. Regarding the first suggestion, we are already developing a process specific for requirements engineering for MAS, where we are establishing stages, steps, and guidelines for accomplish this task.

Regarding the second suggestion, though there is not a particular CASE tool to our DSML, mostly of the existing tools are suitable to produce our models, as we adopted the same symbols used in the standard UCD. However, we intend to develop a tool and eventually we can think whether could be useful creating new symbols to differentiate agent roles from normal actors and internal use cases from normal use cases.

Besides analyzing the forms, we also analyzed the diagrams produced by the students in the second validation to verify how correct they are comparing with our own model. In the table IV, we enlisted the main goals, perceptions, plans, and actions contained in our solution and compared it with the students' diagrams.

Analyzing the models produced by the students, we realized that all the students identified correctly all the agent roles with the correct stereotypes. Regarding the solution of the first student, all goals, actions, and mostly of the perceptions were identified correctly, though he

modelled a perception about the "student actions" assigned to the Mediator that we believe might include the request for help from a student, so we counted it in the partially correct answers column. This student identified all the plans but represented them as actions, so we considered they were partially identified too.

The second student identified correctly all the goals and most of the perceptions but one, the perception of student actions forwarded by the Mediator. The student identified two of the main plans correctly, but the plan "Evaluate student rule" was represented as an action, so we counted it as partially correct. The student identified correctly the main actions, except for one, that may or may be not correct, depending on the interpretation, so we count it as partially correct.

The third student identified a goal named "Analyze student", we think it could encompass the creation of the student model, but we prefer to count it as partially correct, while the other plans where correctly identified. Still regarding the partially correct plan, the student associated to it the perception "Verify change in the model", which we are in doubt if it could be considered as the perception of "Verify student actions", so we counted this perception as partially correct. Also this student identified a perception "current student situation" which we believe is somewhat similar to consult the student model, so we counted it as partially correct too. However, this student did not identify the perceptions of requesting for help by the Student actor nor the evaluation sent by the Specialist. Still, he did not identify the plan for updating the student model, but the other plans were correctly identified. Finally, this student only identified the action of sending the student action to Specialist, ignoring all the others.

Table 4. Requirements Identified by the Students in the Second Validation

Agent Roles Stereotypes	**Right Answers**	**Partially Correct Answers**
StudentModeling – AgentRoleUAM	**5**	
Mediator – AgentRoleSMI	**5**	
Specialist – AgentRolePS	**5**	
Main Goals	**Right Answers**	**Partially Correct Answers**
Create Student Model	**4**	**1**
Help Student	**5**	
Evaluate Student Exercise	**4**	**1**
Main Perceptions	**Right Answers**	**Partially Correct Answers**
Students Actions	**4**	**1**
Student Model from StudentModeling	**2**	**1**
Evaluation From Specialist	**3**	
Help Request from Student	**2**	**1**
Student Action from Mediator	**4**	
Main Plans	**Right Answers**	**Partially Correct Answers**
Update Student Model	**3**	**1**
Select Teach-Learning Strategy	**3**	**2**
Evaluate Student Rule	**2**	**2**
Main Actions	**Right Answers**	**Partially Correct Answers**
Send Student Model to Mediator	**3**	
Send Message to Student	**3**	
Send Student Action to Specialist	**4**	**1**
Send Evaluation to Mediator	**3**	

Regarding the fourth student, though he identified correctly two of the goals, he identified the goal to evaluate student exercise as a plan, so we counted it as partially correct. He did not identify the perceptions of student model nor the requesting for help of a student by the Mediator. With relation to the plans he correctly modelled the plan to evaluate students rules, however, he identified the "select teaching-learning strategy" as a goal (so we counted it as partially correct) and he did not identify the plan for updating the student model. In addition, he did not identify the action of sending the student model to the Mediator, nor the sending messages to the student.

The fifth student identified correctly mostly of the requirements, he just did not identify the perception of student model by the mediator and the action of sending the evaluation results from the Specialist to the Mediator.

We realized that the students, in some cases, did not identify some goals, plans, perceptions, or actions. We believe that it was due to the lack of experience of the students. We think that with more training, the results would enhance. We also think that the text containing Heraclito system needs might be not so easy to interpret to allow the students to identify these requirements correctly. Moreover, maybe the problem presented to the students could be too complex considering their lack of experience.

All the requirements were identified at least by two of the students and, if we consider the partially correct identifications, this number grows up. The students were more successful in the goals identification. On the other hand, they had more difficulties to identify the plans associated to the goals and we agree that sometimes goals and plans are a little difficult to differentiate, however, it is important to model the plans associated to the goals. Some of the students, in the open questions, had highlighted this difficulty.

Some perceptions were identified in smaller numbers, mainly "Student model from StudentModeling" and "Help request from Student". We think some students considered unnecessary to express the first, while we believe some students considered "asking for help" as a kind of attitude performed by the Student actor and represented it as a unique perception together with the attitude of writing rules to solve an exercise.

Some students also did not model the actions of sending messages between the agents. We think that the students can have considered these actions as steps of the plans and therefore this representation could be unnecessary. We partially consider this correct, the roles actions could be internal to the plans and we only represent these actions because we think it is important to highlight the actions affecting the environment.

Another fact that may had affected the students' models is that we did not ask them to document the IUCs. We believe that describing textually the scenarios and steps of each IUC allow to validate the work and to highlight modelling mistakes.

Nevertheless, in spite that all the respondents answered positively to the most questions of our form, as none of the models was completely correct and particularly the third one was a little incomplete, we realized that a process highlighting how to produce this kind of model is necessary. This process is already in development.

6.1. Validation Threats

We validated the DSML in groups of individuals who did not know our work, in order to avoid bias in its application. However, we acknowledge that the number of subjects in the experiments was small as well as the number of multi-agent systems modelled. New validations with different multi-agent systems and with different subjects must be performed to gather more considerations about the DSML validity.

7. CONCLUSIONS

Currently, our DSML is exclusive to the requirements modeling for MAS. For developing this DSML, we extended the UML metamodel, so the Use-Case Diagram can be applied in this specific domain. Our DSML allows representing reactive and cognitive agent roles as well as modeling internal behaviors like perceptions and actions the agents should own when interpreting a role, plus the goals these agents should achieve, the plans to satisfy these goals, and the conditions for these plans to be executed.

Regarding the Internal Use Case concept proposed, we broke, in some way, with the UML use cases diagram philosophy, since this diagram is turned to the representation of the system external requirements, while internal use cases were conceived to represent MAS internal requirements. However, while in much systems the functional requirements can be identified as the functionalities that must be offered to the external users, it is not so true when it refers to MAS. This kind of software, unlike other systems, is characterized just by the existence of software agents – autonomous and proactive entities that, when assuming roles, perform internal functionalities in the system, while external users normally have no idea of which functionalities are these or even that they exist.

Thus, we consider important to identify requirements that represent the internal behaviors needed for the roles taken by agents that composed a system, since we believe that is necessary to represent the internal behaviors associated to the aforementioned roles during the requirements engineering process. We think that the simple identification of the MAS external requirements is too vague, considering that the internal functionalities assigned to the agent roles often constitute an essential part of the software processing to be modelled. Therefore, we believe that the representation of these internal requirements is equally or even more important than the external requirements normally identified by the UML standard use-cases diagram.

One could argue that using this DSML could cause some rigidity, for the need to identify all MAS requirements before building the project up on them. Still, some systems are very complex and, perhaps, some requirements might not be completely identified during the requirements engineering and only along later phases the unidentified requirements would be spotted. One could further argue that plans should not be identified during the requirements engineering, for this could be premature.

However, it is important to highlight that our DSML is not a software development process, but a modelling language that can be applied by any method and it is up to these methods to determine how it will be applied. Thus, it would be possible to develop an iterative and incremental process, where only the most important requirements would be identified in the first cycle and latter detailed in the next phases. Whenever a cycle was concluded, new requirements could be elicited in a new cycle and the process would be repeated until it is considered that the project was finished. Therefore, the identification of plans would not necessarily be made in the execution of the first cycle, but in later cycles, however even then the plans should be identified as requirements. When looking at a use-case diagram, this would allow for an overview of the

software functionalities that would include whichever plans would be associated to which goals and what would be the basic conditions for those plans to be triggered.

Regarding the actions modeling as functional requirements, some people can argue that these actions are part of the plans or the goals and should only be referred in the IUCs documentations. We could agree with this, since the behavior of an action can have few steps. However, we think it is important to highlight graphically the actions affecting the environment, to make clear that these actions are essential behaviors in the system.

Some people argued that the diagrams produced might contain too much information or be a little polluted in situations where there are many agent roles and/or many goals, perceptions, plans, and/or actions. A situation like this can be avoided by addressing each agent role individually, i. e. creating a diagram exclusively for each role.

As a future work, we intend to propose a requirements engineering process in order to define step-by-step how to apply this DMSL for both specifying common and specific requirements for MAS. We intend also to propose a requirements specification document containing the results of this process. We already have a scratch of this process, containing some stages and guidelines, though it must be refined and validated.

In another future work we intend to explore the conceptual modelling for MAS in order to capture the concepts necessary to these systems, representing information needed for the agents like their beliefs, perceptions, goals, messages, plans, etc. We intend to verify if UML class diagram is enough to this task and whether it was necessary to adapt it, as we did to the use-case diagram. In the same way, we intend to apply UML behavioral diagrams, like activity diagrams to detail the internal use cases behavior, also verifying the adaptation level necessary to its application in multi-agent systems.

REFERENCES

[1] Boes, Jérémy & Migeon, Frédéric. (2017) “Self-organizing multi-agent systems for the control of complex systems.” Journal of Systems and Software, 134, pp12–28.

[2] Dam, H. Khanh & Winikoff, Michael, (2013) “Towards a Next-Generation AOSE Methodology”, Science of Computer Programming, 78(6), pp684-694.

[3] Depke, Ralph, Heckel, Reiko & Küster, Jochen, (2002) “Formal agent-oriented modeling with uml and graph transformation”, Science of Computer Programming, 44(2), pp229–252.

[4] Dorri, Ali. Kanhere, Salil & Jurdak, Raja, (2018) “Multi-Agent Systems: A Survey”, IEEE Access, 6, pp28573–28593.

[5] Ganzha, Maria. Omelczuk, Adam. Paprzycki, Marcin. Wypysiak, Mateusz, (2012) “Information resource management in an agent-based virtual organization-initial implementation”, Computer Science and Information Systems, 9(3), pp1307–1330.

[6] Hamidane, Fathi. Mokhati, Farid & Belleili-Souici, Habiba, (2010) “Towards formalizing multi-agent systems functional requirements in maude”, International Journal of Advanced Research in Computer Science, 1(2).

[7] Nakagawa, Hiroyuki. Yoshioka, Nobukazu. Ohsuga, Akihiko & Honiden, Shinichi, (2012) “A Framework for Validating Task Assignment in Multiagent Systems Using Requirements Importance” Lecture Notes in Computer Science, pp443–458.

[8] Perepletchikov, Mikhail & Padgham, Lin, (2005) "Use case and actor driven requirements engineering: An evaluation of modifications to Prometheus" Multi-Agent Systems and Applications IV, Pechoucek, M. Petta, P. Varga, L. eds., Berlin, Heidelberg, pp203–212.

[9] Saleh, Kasem & El-Morr, Christo, (2004) "M-UML: An extension to UML for the modeling of mobile agent-based software systems" Information and Software Technology, 46(4), pp219–227.

[10] Trencansky, Ivan & Cervenka Radovan, (2005) "Agent Modeling Language (AML): A Comprehensive Approach to Modeling MAS" Informatica. Vol. 29, No 4, pp391-400.

[11] Vicari, Rosa & Gluz, João, (2007) "An Intelligent Tutoring System (ITS) View on AOSE" International Journal of Agent-Oriented Software Engineering, vol. 1, 1746-1383, pp295-333.

[12] Berenbach, Brian. Paulish, Daniel. Kazmeier, Juergen & Rudorfer, Arnold (2009) Software & Systems Requirements Engineering In Practice. McGraw-Hill.

[13] Bratman, Michael (1999) Intention, Plans, and Practical Reason. CSLI Publications.

[14] Julian, Vicente & Botti, Vicente, (2019) "Multi-Agent Systems", Applied Sciences, 9, 1402, DOI:10.3390/app9071402.

[15] Regnell, Bjorn, (1999) Requirements Engineering with Use Cases – A Basis for Software Development. Lund University, Sweden.

[16] Sommerville, Ian (2011) Software Engineering - 9th. Edition, Addison Wesley.

[17] Bonjean, N. Mefteh, W. Gleizes, M & Maurel, C, (2014) "ADELFE 2.0". Handbook on Agent-Oriented Design Processes. Springer-Verlag, pp19-64.

[18] Cao, Longbing, (2015) "OSOAD Methodology", Metasynthetic Computing and Engineering of Complex Systems. Springer, London, pp111-129.

[19] Cossentino, Massimo & Seidita, Valeria, (2014) "PASSI: Process for agent societies specification and implementation". Handbook on Agent-Oriented Design Processes. Springer-Verlag, pp287-329.

[20] Cossentino, Massimo. Hilaire, Vincent. Gaud, Nicholas. Galland, Stephane & Koukam, Abederrrafia, (2014) "The ASPECS process". Handbook on Agent-Oriented Design Processes. Springer-Verlag, pp65-114.

[21] Lind, Jurgen, (2001) "The MASSIVE Method". Iterative Software Engineering for Multiagent Systems. Springer-Verlag, ISBN 978-3-540-45162-4, DOI 10.1007/3-540-45162-5,

[22] Morandini, Mirko. Dalpiaz, Fabiano. Nguyen, Cu & Siena. Alberto, (2014) "The Tropos Software Engineering Methodology". Handbook on Agent-Oriented Design Processes. Springer-Verlag, pp463-490.

[23] Rodriguez, Lorena. Insfran, Emilio & Cernuzzi, Luca, (2011) "Requirements Modeling for MultiAgent Systems", Multi-Agent Systems - Modeling, Control, Programming, Simulations and Applications, Intech web.org, pp3-22.

[24] Collier, Rem. O'Hare, Gregory & Rooney, Colm, (2004) "A uml-based software engineering methodology for agent factory", 16th International Conference on Software Engineering and Knowledge Engineering, Banff, CA.

[25] Flake, Stephan. Geiger Christian & Küster Jochen, (2001) "Towards UML-based Analysis and Design of Multi-Agent Systems", Proceedings of ENAIS'2001, Dubai.

[26] Frank, Ulrich, (2011) "Some guidelines for the conception of domain-specific modelling languages" 4th International Workshop on Enterprise Modelling and Information Systems Architectures, Gesellschaft für Informatik, Bonn, pp93-106.

[27] Galafassi, Fabiane, Galafassi, Cristiano. Vicari, Rosa & Gluz, Joao, (2019) "Heráclito: Intelligent Tutoring System for Logic", Advances in Practical Applications of Survivable Agents and Multi-Agent Systems: The PAAMS Collection, PAAMS 2019, Lecture Notes in Computer Science, vol.11523, pp251-254.

[28] Guedes, Gilleanes & Vicari, Rosa, (2010) "A UML Profile Oriented to the Requirements Collecting and Analyzing for the MultiAgent Systems Project", 22nd International Conference on Software Engineering and Knowledge Engineering - SEKE2010, Redwood, California, USA.

[29] Guedes, Gilleanes & Vicari, Rosa (2011) "Applying a UML Metamodel to the Requirements Modeling in Multi-Agents Systems Projects - The APA Case Study", XV Portuguese Conference on Artificial Intelligence, Vol. 1, pp609-623.

[30] Heinze, Clinto. Papasimeon, Michael & Goss, Simon, (2000) "Specifying agent behaviour with use cases", Pacific Rim International Workshop on Multi-Agents, Springer, pp128–142.

[31] Iglesias, Carlos. Garijo, Mercedes. Gonzalez, Jose & Velasco, Juan, (1997) "Analysis and Design of Multi-Agent Systems using MAS-Commonkads" International Workshop on Agent Theories, Architectures, and Languages, Springer. pp313–327.

[32] Laouadi, Mohamed. Mokhati, Farid & Seridi-Bouchelaghem, Hassina, (2010) "A novel formal specification approach for real time multi-agent system functional requirements", German Conference on Multiagent System Technologies, Springer, pp15–27.

[33] Laouadi, Mohamed. Mokhati, Farid. Seridi-Bouchelaghem, Hassina, (2013) "Towards an organizational model for real time multi-agent system specification", Science and Information Conference, IEEE, pp577–584.

[34] Papasimeon, Michael & Heinze, Clinton, (2003) "Specifying Requirements in a Multi-Agent System with Use Cases", Defence Science and Technology Organization, Technical Report.

[35] Slhoub, Khaled. Carvalho, Marco & Bond, Walter, (2018) "Recommended practices for the specification of multi-agent systems requirements", IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2017, pp179–185.

[36] Spanoudakis, Nikolaos. Moraitis, Pavlos, (2008) "The Agent Modeling Language (AMOLA)", 13th International Conference on Artificial Intelligence: Methodology, Systems, and Application, Springer-Verlag, Berlin, pp32–44.

[37] Werneck, Vera. Kano, A. & Cysneiros, Luiz (2007) "Evaluating ADELFE Methodology in the Requirements Identification", 10th Workshop on Requirements Engineering, Canada, pp3-24.

[38] OMG – Object Management Group, (2011), "OMG Unified Modeling Language - Version 2.4.1", http://www.omg.org.

[39] OMG – Object Management Group. (2017), "OMG Unified Modeling Language - Version 2.5.1", http://www.omg.org.