

# Abstract and Concrete Syntaxes for Software Agent Environment Modeling in CArtaGo Infrastructure

Ufuk Firtina  
International Computer Institute  
Ege University  
Izmir, Turkey  
ufukfirtina92@gmail.com

Baris Tekin Tezel  
Department of Computer Science  
Dokuz Eylul University  
Izmir, Turkey  
baris.tezel@deu.edu.tr

Moharram Challenger  
International Computer Institute  
Ege University  
Izmir, Turkey  
moharram.challenger@ege.edu.tr

Geylani Kardas  
International Computer Institute  
Ege University  
Izmir, Turkey  
geylani.kardas@ege.edu.tr

**Abstract**— In this work, a metamodel is introduced for CArtaGo infrastructure which can be used in the modeling of software agents on this infrastructure. The metamodel allows the programming of artifact-based environments for multi-agent systems. It can be used for different agent platforms as it is independent of the specifications of these platforms. Also, a graphical concrete syntax is developed for the proposed metamodel and a modeling tool is provided. The use of syntax and the modeling tool is demonstrated with the JaCaMo Gold Miners case study which consists the development of software agents for finding gold within a certain area.

**Keywords**—Software agents; multi-agent systems, CArtaGo, modeling, metamodel

## I. INTRODUCTION

Software Agents, which are autonomous, reactive, and proactive, have social ability that enable to interact with other agents and humans to solve their problems. To fulfil their tasks and interact with each other, intelligent agents constitute systems called Multi-Agent Systems (MASs) [1].

Programming of agents, their environments and relationships are very important part of MAS development. To target the complexity of MASs, Domain-specific Modeling Language (DSML) studies have gained popularity [2–7]. Various DSML tools have emerged to make creation of agents and their coding easier. In addition to the creation of agents, programming of the environment in MAS is also critical. This process can become very complex duty to the dynamic interaction ability of agents with the environment elements.

To accomplish this process, the abstract Environment programming helps the implementation of agents who interact with their surroundings and use their services and activities to their advantage. The CArtaGo infrastructure [8], which can work independent of any specific agent platform, has been established for this purpose. CArtaGo allows environmental programming for MASs. It can be used in a wide range of agent languages due to its orthogonal structure. There exists various MAS metamodels (e.g. [12–15]) from which abstract and concrete syntaxes for MAS DSMLs originate. However, they currently do not consider the environment modeling according to the specifications of CArtaGo framework. Hence, in order to fill this gap, we propose a metamodel which may pave the development of a complete DSML enabling modeling MAS and its environment and supporting the automatic code generation for CArtaGo framework.

The paper is organized as follows: a brief description of CArtaGo infrastructure is given in the next section. The proposed CArtaGo metamodel is presented in Section III. The concrete syntax for CArtaGo modeling

language and a use case of this language are presented in Section IV. Insights and related work are given in section V and Section VI. The paper concludes in section VII.

## II. CARTAGO OVERVIEW

CArtaGo [8] is a general-purpose infrastructure that allows artifact-based environments to be programmed and executed for MAS. CArtaGo makes it possible to create open workspaces where agents in different environments can work together. In this way, MAS developers have a simple java-based programming model for designing and programming agent computing environments with different objects. Infrastructures play an essential role for keeping useful abstractions alive from design to runtime [8]. Agent infrastructures provide useful services for the creation, management and communication of agents. Agents can join a workspace and use artifacts included in MAS. They can also create new artifacts. CArtaGo provides basic services for agents to create and use these artifacts. So, MAS engineers have a flexible way of designing and constructing all kinds of artifacts. CArtaGo is designed to be independent of any specific agent model or platform. It is intended to be orthogonal for agent models or platforms used to define the architecture and behavior of the agents. As we can see in the JaCaMo [12] approach, CArtaGo is useful when integrated with agent programming languages based on Belief-Desire-Intention (BDI) architecture.

CArtaGo has a layered architecture depicted in Fig. 1. The kernel follows the interactions between agents and artifacts. It dynamically creates agents and artifacts through the given templates. The agent accesses the kernel properties with the agent contexts. The agent contexts establish a connection between the agent and the CArtaGo environment. Also, operational execution requests are collected in a pool by the environmental controllers.

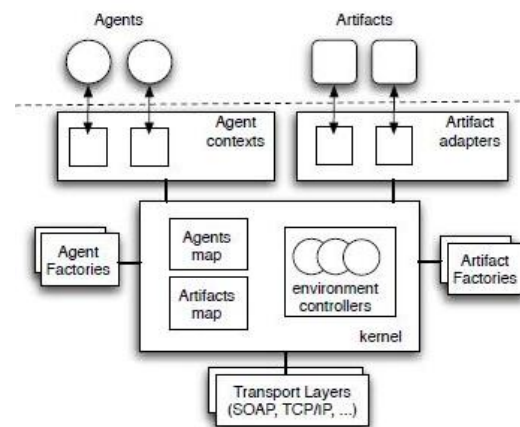


Fig. 1. Abstract architecture of a CArtaGo application (taken from [8])

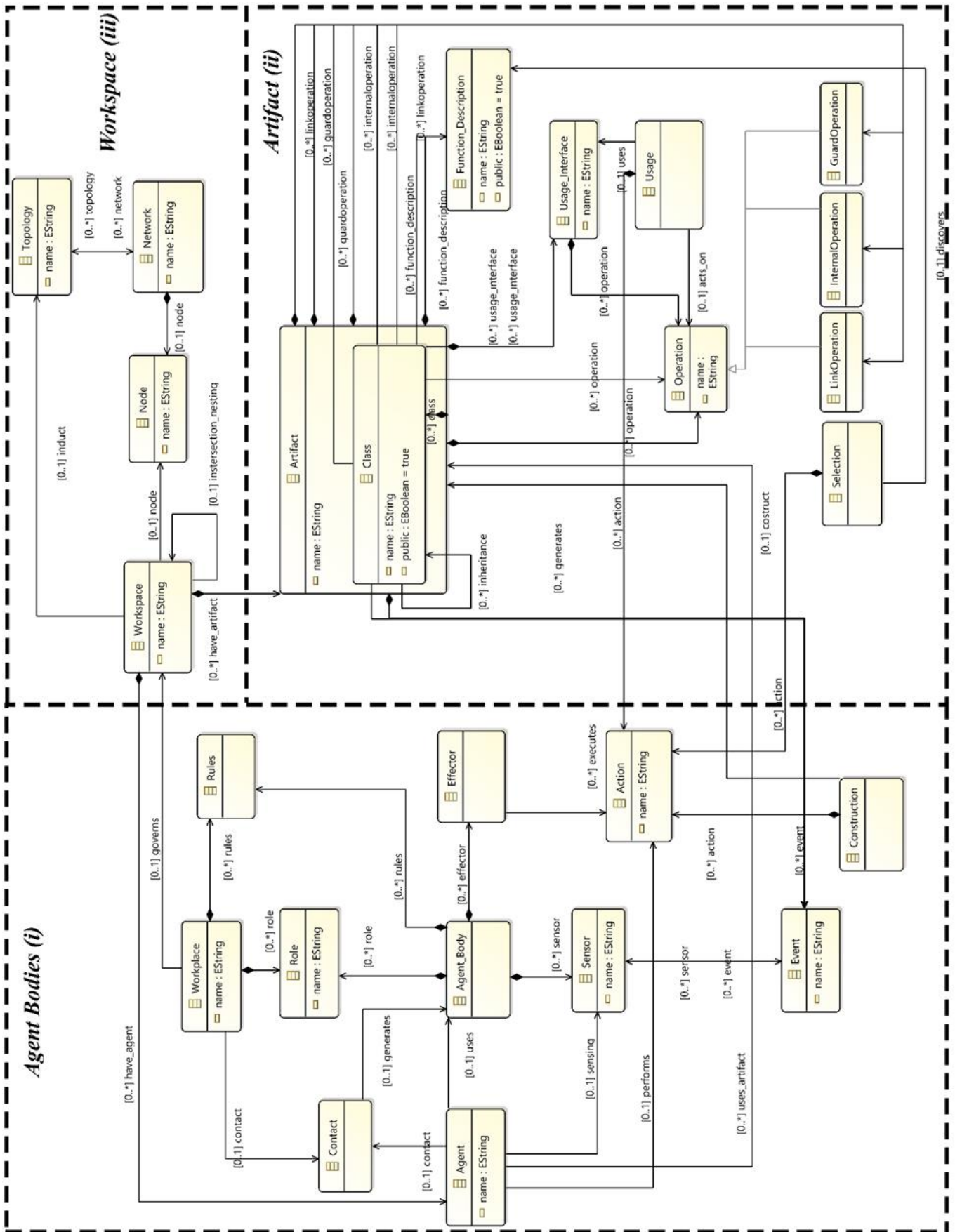


Fig. 2. The Proposed CArtaGo Metamodel

### III. CARTAGO METAMODEL

The concepts and their relations to other concepts without considering their meaning are explained by the abstract syntax of a DSML. In other words, the vocabulary of the concepts provided by the language and how it can be combined to create models or programs is explained by the abstract syntax of a language [4]. In terms of Model Driven Development (MDD), a metamodel describes what models should look like and, defines the abstract syntax. In this section, we discuss our metamodel that makes up the abstract syntax of CARTAGO. The metamodel (given in Fig. 2) adopts the concepts defined in [9-10] and extends them with new MAS concepts and their relations. Elements of the metamodel are written in italics during following discussion.

CARTAGO is, in fact, based on the definitions of Agent and Artifacts (A&A) metamodel introduced in [10]. As a framework, CARTAGO provides a library of predefined general-purpose entity types for artifacts and MAS workspace environments. According to A&A specifications, *Agents* are pro-active entities responsible for the goals that make up the whole MAS behavior. *Artifacts* are reactive entities that enable individual agents to work together at MAS and provide services and functions that shape the agent environment according to MAS needs.

The A&A metamodel is characterized in terms of three basic abstractions [10]: Agents, which representing the proactive components in the system, perform the encapsulation process for autonomous execution of certain activities in the environment. Artifacts, which represent resources, data and media shared by agents in the system, are passive components. Workspaces are containers of agents and artifacts and they are helpful for defining the topology for the environment.

Conforming to the abstractions of A&A, Molesini et al. [9] proposed a conceptual meta-model for CARTAGO. This CARTAGO metamodel consists of three main parts: (i) *Agent bodies* are the elements that make it possible for agents to be identified and operated in the working environment, (ii) *Artifacts* are the basis for the creation of working environments and (iii) *Workspaces* are containers of artifacts and agents and they help defining the topology of the working environment. These parts are also kept in our metamodel and are shown in Fig. 2 with the dotted lines.

The agent body has *effectors*, *actions*, *sensors* and *events*. Effectors are used to perform actions in the work environment. The sensors collect events from the working environment. Agents interact with work environments through their own bodies. Actions are used to select, create and run artifacts.

Artifacts are the basic structures managed by CARTAGO. Agents use *artifacts* by running a list of *operations* contained within artifact *usage interface*. Operations must be run to generate the observable event that is collected by the sensors of the agents or to update the internal state of the artifact. Each artifact has a *functional description*. With this definition, agents use artifacts more efficiently. The *operating instructions* clearly define the functions of the artifact and how to use it.

Artifacts are in work areas defined by a *topology*. Workspaces are collection of agents and artefacts. Agents













can dynamically add or remove artefacts from the workspace. Agents can dynamically enter or exit the workspace. Also, workspaces provide an environment for the interaction between agents and artifacts. Finally, workspaces provide functionality for generating and perceiving events. This allows access and use of artefacts.

Abovementioned concepts and their relations are already defined in [9-10] and also included in our metamodel. However, we experienced that the definitions given in A&A [10] and SODA [9] are not enough to derive a syntax for a DSML that can be used for implementing MAS on CartAgO. Especially, some additional entities and attributes are needed to be included inside the CartAgO for generating software codes during exact MAS implementations. For this reason, we first added the *Class* meta-entity and its attributes into the metamodel that links the *Artifact*. Instances of this *Class* will constitute the code part of the Java-based Artifact descriptions in a MAS implementation on the CartAgO infrastructure. The functions, operations and other required structures, depending on the *Class* meta-entity, are also defined in the metamodel (see lower right part in Fig. 2). In order to use different types of operations inside the CARTAGO code structure, we need to define new meta-entities of different type operations such as *LinkOperation*, *InternalOperation* and *GuardOperation*. These new meta-entities are added into our metamodel via inheritance relations, as being the specializations of the Operation meta-entity. Definition of these new entities may lead to the increase in throughput, i.e. a DSML based on this metamodel will be capable of generating more detailed code for MAS implementations.

### IV. USE OF THE METAMODEL

While the concepts represented in a language, and the relationships between those concepts are expressed by the specification of abstract syntax, a mapping between meta-elements and their representations are provided by concrete syntax for models. In short, the concrete syntax is the set of notations which provide a graphical/textual representation of the concepts. In this section, we discuss graphical concrete syntax for the proposed metamodel which maps the abstract syntax elements of CARTAGO to their graphical notations.

Table 1. Graphical Notations

Concept	Notation	Concept	Notation
Agent		Artifact Relation	
Artifact		Link Relation	
Artifact Class		Guard Relation	
Operation		Operation Relation	
Internal Operation		Internal Relation	
Link Operation			
Guard Operation			

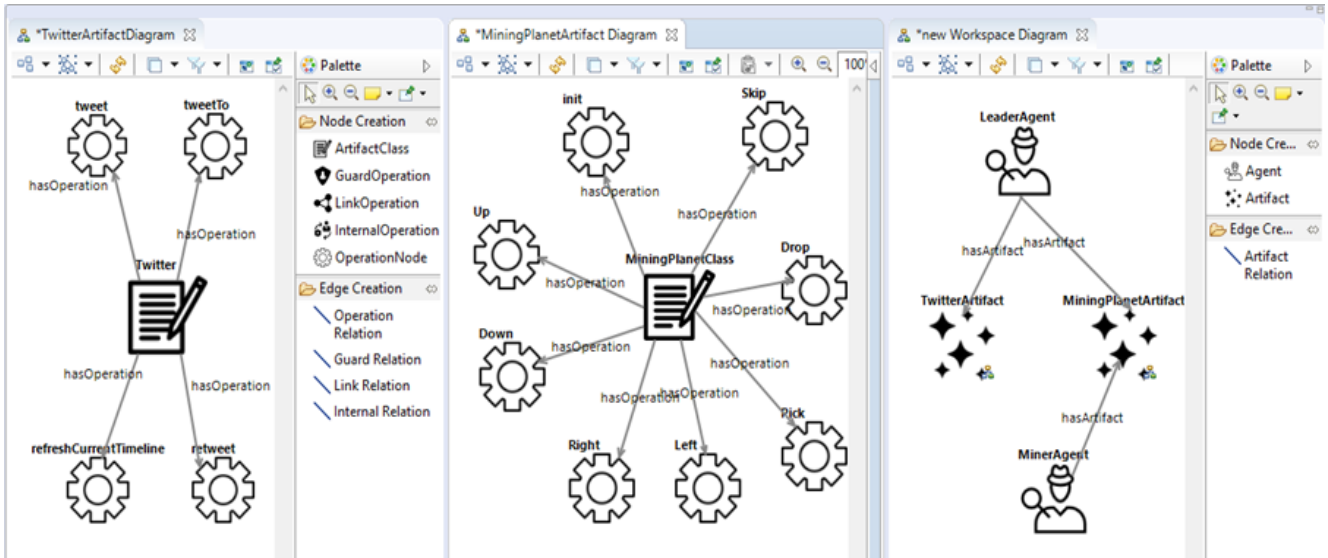


Fig. 3. Gold-Miners model in the proposed CARtAgO modeling tool

Based on the metamodel discussed in the previous section, we have developed a concrete syntax which enables the use of Cartago concepts during MAS development. Graphical notations used in this syntax are given in Table 1. Agent developers can use this syntax inside a modeling tool which is also developed in this study. This modeling tool for CARtAgO infrastructure is constructed upon Eclipse Sirius<sup>1</sup>. Eclipse Sirius allows us to create a concrete syntax for a metamodel in Ecore. We can create concrete syntax using the provided notations for the relations and elements in the metamodel.

The main elements of our modeling tool are workspace and artifact diagrams. Although we have an element representing agents, modeling the internal of agents is not within the scope of this study. In here, we consider the modeling agents and the environment they reside. Agents and artifact structures can be modeled in the workspace diagram provided by the tool. Links, showing a relation between an agent and an artifact, can be easily created. When a developer double-clicks on an artifact, a new diagram opens. In this diagram, the agent developer can model the artifact class. Again, based on the abstract syntax definitions, ArtifactClass, Operation, GuardOperation, LinkOperation and InternalOperation elements can be used in an artifact diagram. There are also links to associate these elements with ArtifactClass. With these structures, an agent developer can model the whole artifact structure of the MAS-to-be-implemented.

To illustrate using the proposed syntax and graphical modeling tool, let us consider modeling Gold-Miners<sup>2</sup> case study artifacts. In this case study, it is aimed to find gold within a certain area by agents. There are two different agents in this system, Leader Agent with mission of informing the agents about the locations where the gold is

located and Miner Agent that carries the gold which is found in the field. Our goal is to model the artifact structures to program the environment of these agents. In this case study, there are two artifact files namely twitter.java and MiningPlanet.java. We simply tried to model these artefacts and workspace shown in Fig. 3. Also, it is possible to combine this model with different models created in other MAS.

## V. INSIGHTS

When a MAS structure is being programmed, it is necessary to define the internal structure of the agents, the environment they interact with and the relationships between them. With the creation of these structures, MAS can become very complex. This complexity can be resolved with a higher abstraction level using modeling technique.

In this respect, the creation of the CARtAgO metamodel can be very useful for facilitating environment programming in a DSML. CARtAgO abstract syntax can be used by integrating with other Agent DSMLs. The fact that the latest version of CARtAgO has a connection with the Jason Agent Programming Language[13] makes it particularly possible to use with the BDI agent model.

In this study, we have created a syntax that form the basis for a DSML. This work can be elaborated to create a concrete syntax that allows for more detailed modeling by extending the metamodel with agent internals. To reduce the complexity in MAS programming, the targeted code can be automatically generated.

Therefore, the code generation can provide part of the target code from the model created in the DSML which can increase the development performance and reduce the number of errors. In the current version of our work, there is no support for code generation. However, as our next work, the code generation will be realized.

As a result, the creation of the CARtAgO metamodel and related concrete syntax allows for environment programming for MAS, and enables the derivation of a full-fledged DSML for MAS implementation.

<sup>1</sup> Sirius Modeling Tool, <https://eclipse.org/sirius>

<sup>2</sup> JaCaMo Gold-Miners Project, <http://jacamo.sourceforge.net/tutorial/gold-miners/initial-gold-miners.zip>

## VI. RELATED WORK

Considering the studies related to Software Agent Environment programming for MASs, Boissier et al. [12] conducted a study with JaCaMo platform to integrate agent-oriented programming, organization-oriented programming and environment-oriented programming.

This work presents a simple example metamodel for the JaCaMo platform. They created a multi-agent system example with the construction of a house. However, no abstract and/or concrete syntax is discussed for CartAgO in this study. In [9], Molesini et al. aimed to compare some of the infrastructures supported by MAS through a case study. In this work, a conceptual UML metamodel is presented for CArTAgO.

Also, Omicini et al. [10] focus on the modeling of agent environments and first-class variants of MAS environments and artifacts. They presented sample applications of agent-related research fields. They performed a detailed study on the artifact structure that CArTAgO focused on. But this study does not propose any metamodel.

In [8], the concept of environmental programming in MAS is introduced and a concrete computation and programming model based on the abstraction applied by the CArTAgO framework is described. The paper also includes a description of the main concepts related to artifact-based environments and related CArTAgO technology. However, this study does not include any work on CArTAgO metamodel.

The work herein contributes to the efforts discussed above by defining and implementing new abstract and concrete syntaxes for CartAgO which leads to develop a complete DSML for MAS and environment modeling.

## VII. CONCLUSION

In this work, we have introduced a CArTAgO modeling framework which provides artifact-based environment programming for MAS, independent of agent platforms.

To this end, we have developed a CArTAgO metamodel in Eclipse Ecore. The metamodel includes the original Artifact, Agent and Workspace main elements and their relations pertaining to CartAgO infrastructure in addition to the newly defined Artifact entities to support MAS implementation. Also, in this way, a concrete syntax is provided by suggesting some graphical notations and symbols and a graphical modeling tool is introduced to model MAS based on CArTAgO. This tool allows for environment modeling of any agent platform and forms the basis for integration with other agent internal modeling frameworks.

In our next work, we are aiming to realize the integration of the proposed metamodel and tool with Jason agent programming language [13]. In addition, icons used for concrete syntax can be improved for better user interaction using physics of notations principles [14].

## REFERENCES

[1] M. Wooldridge and N. R. Jennings, "Intelligent agents: theory and practice," *Knowl. Eng. Rev.*, vol. 10, no. 02, p.

115, Jun. 1995.

- [2] G. Kardas, B. T. Tezel, and M. Challenger, "Domain-specific modelling language for belief–desire–intention software agents," *IET Softw.*, Apr. 2018.
- [3] E. J. T. Gonçalves *et al.*, "MAS-ML 2.0: Supporting the modelling of multi-agent systems with different agent architectures," *J. Syst. Softw.*, vol. 108, pp. 77–109, Oct. 2015.
- [4] M. Challenger, S. Demirkol, S. Getir, M. Mernik, G. Kardas, and T. Kosar, "On the use of a domain-specific modeling language in the development of multiagent systems," *Eng. Appl. Artif. Intell.*, vol. 28, pp. 111–141, Feb. 2014.
- [5] C. Hahn, "A Domain Specific Modeling Language for Multiagent Systems," in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, 2008, no. AAMAS '08, pp. 233–240.
- [6] J. M. Gascueña, E. Navarro, and A. Fernández-Caballero, "Model-driven engineering techniques for the development of multi-agent systems," *Eng. Appl. Artif. Intell.*, vol. 25, no. 1, pp. 159–173, Feb. 2012.
- [7] J. Faccin and I. Nunes, "A tool-supported development method for improved BDI plan selection," *Eng. Appl. Artif. Intell.*, vol. 62, pp. 195–213, 2017.
- [8] A. Ricci, M. Viroli, and A. Omicini, "CArTAgO: An Infrastructure for Engineering Computational Environments in MAS," in *3rd International Workshop "Environments for Multi-Agent Systems" (E4MAS 2006)*, 2006, pp. 102–119.
- [9] A. Molesini, E. Denti, and A. Omicini, "From AOSE Methodologies to MAS Infrastructures: The SODA Case Study," in *Engineering Societies in the Agents World VIII*, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 300–317.
- [10] A. Omicini, A. Ricci, and M. Viroli, "Artifacts in the A&A meta-model for multi-agent systems," *Auton. Agent. Multi. Agent. Syst.*, vol. 17, no. 3, pp. 432–456, Dec. 2008.
- [11] A. Ricci, M. Piunti, and M. Viroli, "Environment programming in multi-agent systems: an artifact-based perspective," *Auton. Agent. Multi. Agent. Syst.*, vol. 23, no. 2, pp. 158–192, Sep. 2011.
- [12] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi, "Multi-agent oriented programming with JaCaMo," *Sci. Comput. Program.*, vol. 78, no. 6, pp. 747–761, Jun. 2013.
- [13] R. H. Bordini, J. F. Hbner, and M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak using Jason*. Chichester, UK: John Wiley & Sons, Ltd, 2007.
- [14] D. Moody, "The 'Physics' of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering," *IEEE Trans. Softw. Eng.*, vol. 35, no. 6, pp. 756–779, Nov. 2009.