

A Metamodel for Jason BDI Agents

Baris Tekin Tezel¹, Moharram Challenger², and Geylani Kardas³

- 1 International Computer Institute, Ege University, Izmir, Turkey
baris.tezel@deu.edu.tr
- 2 International Computer Institute, Ege University, Izmir, Turkey
moharram.challenger@mail.ege.edu.tr
- 3 International Computer Institute, Ege University, Izmir, Turkey
geylani.kardas@ege.edu.tr

Abstract

In this paper, a metamodel, which can be used for modeling Belief-Desire-Intention (BDI) agents working on Jason platform, is introduced. The metamodel provides the modeling of agents with including their belief bases, plans, sets of events, rules and actions respectively. We believe that the work presented herein contributes to the current multi-agent system (MAS) metamodeling efforts by taking into account another BDI agent platform which is not considered in the existing platform-specific MAS modeling approaches. A graphical concrete syntax and a modeling tool based on the proposed metamodel are also developed in this study. MAS models can be checked according to the constraints originated from the Jason metamodel definitions and hence conformance of the instance models is supplied by utilizing the tool. Use of the syntax and the modeling tool are demonstrated with the design of a cleaning robot which is a well-known example of Jason BDI architecture.

1998 ACM Subject Classification I.6.5 Model Development, I.2.11 Multiagent systems

Keywords and phrases metamodel, BDI agent, multi-agent system, Jason

Digital Object Identifier 10.4230/OASIS.SLATE.2016.8

1 Introduction

In agent-oriented software engineering (AOSE), many metamodels (e.g. [1, 18, 19, 20, 13, 2, 7, 10]) exist for describing the intelligent software agents and multi-agent systems (MASs) which are composed of these agents. A group of these metamodels (e.g. [1, 18, 2]) only provides the definition of traditional AOSE methodologies [14] while another group (e.g. [19, 12, 13, 7, 10]) aims at more general specification of agent systems from different aspects varying from agent internals to MAS organizations. They present abstract syntaxes for domain-specific MAS modeling languages (such as [17, 12, 8, 11]) and hence enable the model-driven development (MDD) of MAS [15]. Taking into account the OMG's Model-driven architecture (MDA) specifications¹, it is proper to indicate that above metamodels support the platform-independent MAS modeling which is abstract from the underlying agent implementation and/or execution platforms.

On the other hand, platform-specific modeling of agent systems is also possible by using metamodels. AOSE researchers propose metamodels which are specific to real MAS

¹ OMG Model Driven Architecture, <http://www.omg.org/mda/>.



implementation platforms such as JACK², JADE³ or JADEX⁴. For instance, [13] defines the metamodels of JADE and JACK platforms while [16] and [8] consider the development of MAS on JADEX platform. Definition of MAS metamodels at these different abstraction levels and application of model transformations between these platform-independent and platform-specific MAS metamodels enable the implementation of modeled agent systems in abovementioned MAS platforms.

This paper introduces our ongoing work on the platform-specific modeling of Belief-Desire-Intention (BDI) agents [22] according to the specifications and features of Jason platform⁵ which provides the implementation of agents using a Prolog-like logic programming language called AgentSpeak [21]. We discuss the derivation of a metamodel for Jason and construction of a graphical modeling tool in which agent developers can model Jason systems conforming to the introduced metamodel. We believe that the work presented herein contributes to the abovementioned MAS metamodeling efforts by taking into account another BDI agent platform which is not considered in the existing platform-specific MAS modeling approaches. The work introduced in [9] is the single exception which also aims at the modeling of Jason agents. However, the given metamodel does not support the reusability of same concepts like beliefs, events, actions plans, goals and rules for different agents of a MAS if required. The metamodel proposed in this study supports the reusability of all related entities inside the whole MAS model.

Rest of the paper is organized as follows. In section 2, Jason platform is briefly discussed. Section 3 introduces the metamodel we propose for Jason. Section 4 covers the definition of the concrete syntax, implementation of a modeling environment for Jason agents and exemplification of its usage. Section 5 concludes the paper.

2 Jason

Jason [3] is a Java-based interpreter for an extended version of a Prolog-like logic programming language called AgentSpeak [21]. AgentSpeak is based on the well-known BDI architecture [22] for software agents which originates from Bratman's human practical reasoning theory [5]. In BDI architecture, agents constantly monitor their environment and respond instantly to the changes in the environment. This reaction depends on agent's mental attitudes. An agent has three types of mental attitudes which are belief, desire and intention.

Beliefs are information about an agent's itself, other agents and the environment that the agent is located. Desires express all possible states of affairs which might be achieved by an agent. One desire is a potential trigger for an agent's actions. Simply, desires are often considered as options for an agent. Finally, intentions represent the states of affairs which have been decided to work towards by the agent. Intentions may be delegated goals or results of considered options.

Simply, an AgentSpeak agent is defined by a set of beliefs, rules and plans. Beliefs represent initial knowledge of an agent. Rules are logic expressions or mathematical equations. Plans constitute the actions and/or subgoals to achieve the current goal.

A plan of an AgentSpeak agent consists of a triggering event, a context and a body element. The triggering event specifies the events for which that plan is suitable. The context

² JACK Autonomous Software, <http://aosgrp.com/products/jack/>.

³ JAVA Agent DEvelopment Framework, <http://jade.tilab.com/>.

⁴ JADEX Active Components, <https://www.activecomponents.org/#/project/news>.

⁵ Java-based interpreter for an extended version of AgentSpeak, <http://jason.sourceforge.net/wp/>.

represents whether the plan is applicable according to the beliefs of the agent. Body is a sequence of basic actions and/or subgoals.

Logic programming based on AgentSpeak [4] serves a computationally efficient capability for BDI agent development. With providing a Java-based interpreter, Jason extends the expressiveness of AgentSpeak during implementation of cognitive agents.

3 A metamodel for Jason

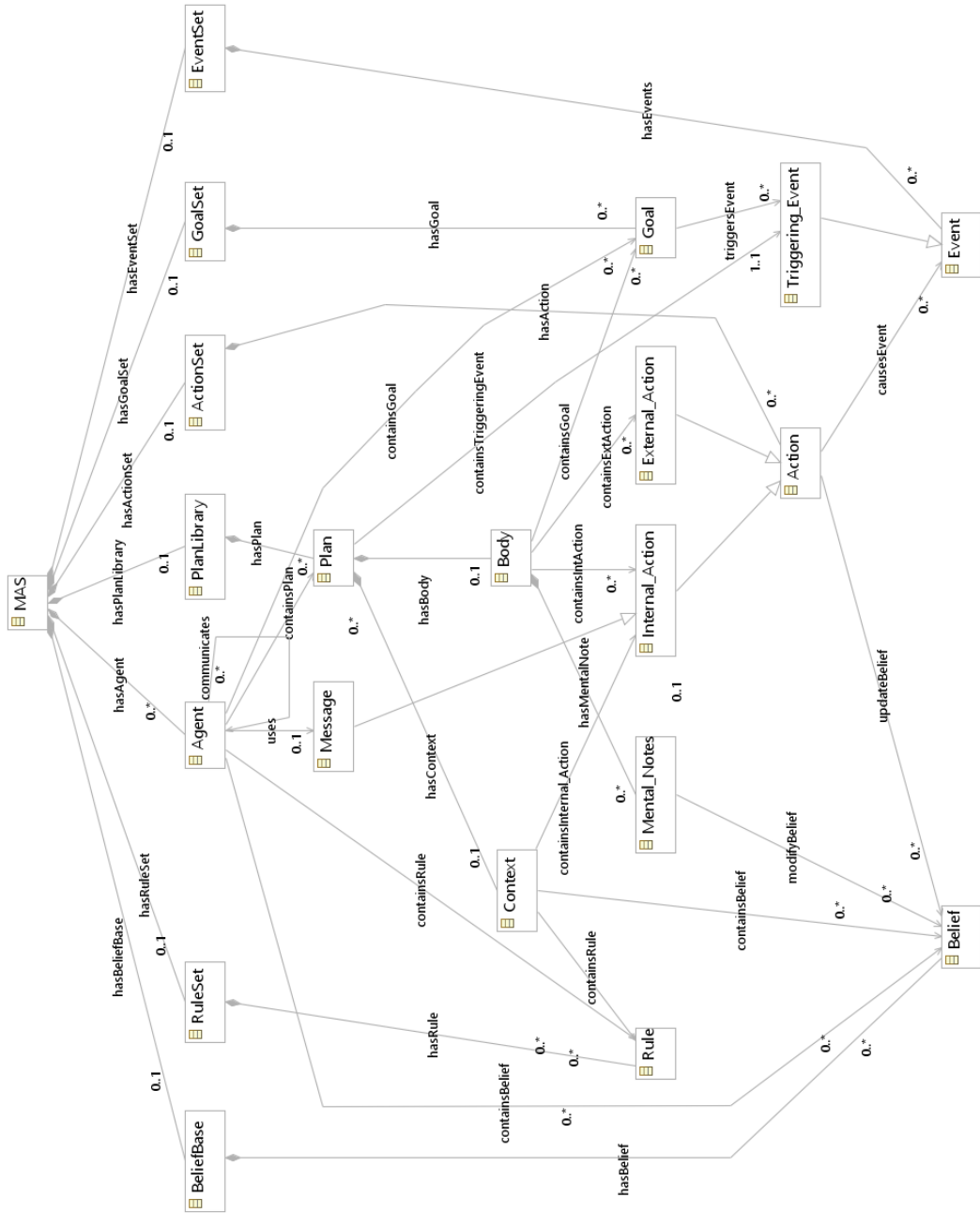
A metamodel, which may provide an abstract syntax of a modeling language for Jason platform, is presented in this section. As shown in Figure 1, the metamodel provides the meta-entities and their relations required for both the internal BDI agent architecture and MAS organization on Jason platform. Derivation of the main elements and their associations is mainly based on the definitions given in [4]. Moreover, the metamodel complies with the Extended Backus-Naur Form (EBNF) of Agent Speak Language [4] with the required level of abstractions. During the discussion below, the meta-entities of the proposed Jason metamodel are given in the text with italic font.

A *Multi-agent System* (MAS) is mainly composed of agents. However, the metamodel includes five different entities other than the *Agent* entity for the complete representation of the MAS composite structure. These are *BeliefBase*, *EventSet*, *RuleSet*, *PlanLibrary*, *ActionSet* and *GoalSet*. Each of them is denoted by individual meta-elements which are owned by the agents. This helps the metamodel's support on the reusability for the agent developers.

BeliefBase is consisted of possible beliefs which may be adopted by agents in order to use for initial beliefs or context of plans. But, above mentioned *BeliefBase* of a MAS is different from the belief base of individual agents. It has a static structure while belief base of an agent can be changed during the execution of reasoning cycles. Also, each belief represents knowledge about an agent or the environment. *EventSet* contains events within the environment. In Jason platform, each event is represented by a triggering event which is also a meta-element (*Triggering_Event*) included in the proposed metamodel. Events happen as a consequence of belief or goal changes inside an agent's mind. *RuleSet* is composed of rules. Each *Rule*, which allows arriving at a judgement based on beliefs of an agent, can simplify making certain conditions used in the context of plans. Rule concept is directly related with logic programming especially in Prolog.

In the metamodel, *ActionSet* stores actions which can be used by all agents. Actions, which are included in the body of plans, represent what an agent is capable of performing. The Jason metamodel provides two kinds of actions which are internal actions and external actions. Internal actions (represented by the *Internal_Action* meta-entity) are executed inside an agent's mind. So, they cannot change the environment. Communication actions of an agent, which are called as *Messages* in the Jason metamodel, are also internal actions. On the other hand, external actions (represented by *External_Action* meta-entities) directly change the environment.

An agent has *Goals* to achieve. Bringing together all candidate goals for each agent within the environment creates the *GoalSet*. In fact, goals are the desired states, which are achieved by the executed plans. There are two types of goals, including achievement goals and test goals. An achievement goal represents a state of the environment which is desired to be achieved by an agent. A test goal is used to retrieve knowledge from beliefs of an agent or to check something expected what is actually believed by the agent, while executing a plan body. Those agent types are included in the metamodel as being attributes



■ **Figure 1** The Jason metamodel.

of *Goal* meta-entity. However, these attributes can not be shown in Figure 1 since attribute compartments are closed in the metamodel given in the figure due to space limitations.

An agent reacts against events. Those reactions of an agent are represented in the metamodel by *Plans*. As previously mentioned, a plan, which represents the skills of an agent, has three distinct parts. These are the triggering event, the context and the body. *Triggering_Event* is the post-condition of a plan. *Context* is the pre-condition of a plan and composed of beliefs and rules which allow deducing based on the knowledge. The *Body* of a plan is simply a list of actions. Besides, the body has sub-goals and *Mental_Notes* which modify the belief base. All possible plans which can be used by an agent, are covered in a *PlanLibrary* meta-element. Finally, an agent has initial beliefs, plans and pursuing goals. Initial beliefs and pursuing goals are adopted from the related sets defined in the metamodel (e.g. *BeliefBase*, *GoalSet*). Also, plans of an agent are selected from the *PlanLibrary*.

4 Concrete Syntax

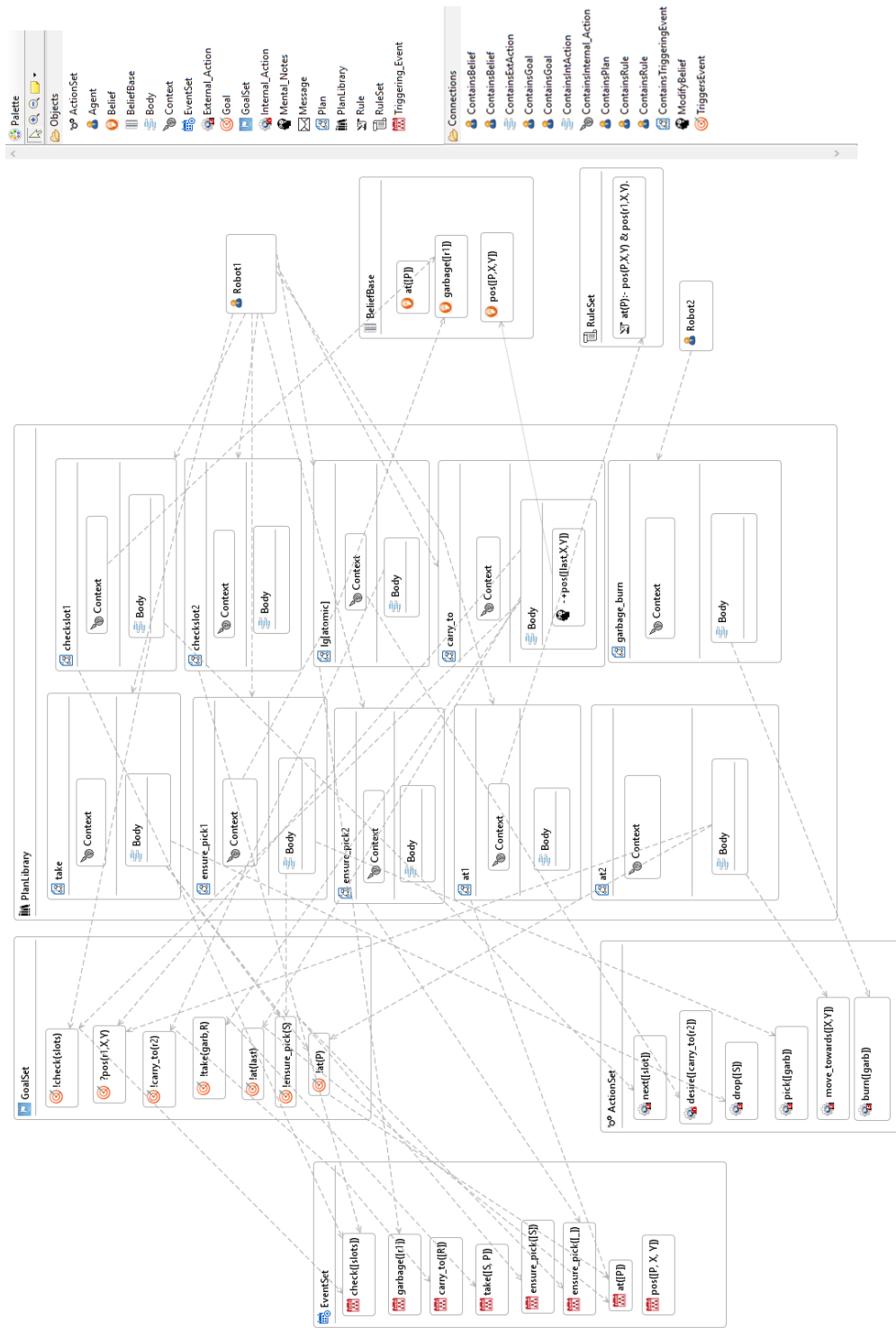
Whilst the specification of abstract syntax inside a metamodel includes those concepts that are represented in the language and the relationships between those concepts, concrete syntax definition provides a mapping between meta-elements and their representations for models. In fact, the concrete syntax is the set of notations which facilitates the presentation and construction of the language. This section discusses the graphical concrete syntax which maps the Jason platform's abstract syntax elements presented in the previous section to their graphical notations.

We use Epsilon EuGENia⁶ for constructing the concrete syntax from the Jason metamodel. Providing a tool for implementing a graphical modeling editor from a single annotated Ecore metamodel and hence facilitating the construction of a concrete syntax of a metamodel caused us to prefer EuGENia in this study. After setting the graphical notations for the abstract syntax meta-elements, we employ EuGENia to tie notations to the domain concepts. The graphical notations are listed in Table 1. Meta-elements which are not used directly during the creation of a Jason instance model, do not have notations in the table. For instance, the Action meta-element, which is inherited by internal action and external action, is not needed in an instance model. But, internal action and external action have to be existed in an instance model. Moreover, some composition relations, which are modeled with compartments in their holder elements, do not have graphical notations. On the other hand, rest of the composition relations which are modeled with connections to their holder elements, have the same graphical notations with their owner.

Use of the derived concrete syntax inside the EuGENia-based modeling editor can impose some restrictions/controls for the conformance of designed models to the specifications of our Jason metamodel. Benefiting from adopting the Ecore as the meta-metamodel while the production of the graphical modeling tool, the graphical concrete syntax succeeds in providing the check of constraints such as compartment, number of relationships between model elements and source and destination elements in a relationship. Finally, defined inheritance relationship in the Jason metamodel obligates users while modelling. Inheritance relationship constraint in question checks whether a sub-entity in a Jason instance model includes all of the attributes and relationships of its super-entity.



















In order to demonstrate the use of the proposed concrete syntax during modeling a Jason MAS, let us consider the design of a cleaning robot. Cleaning robot is one of the well-known

⁶ Epsilon Eclipse GMT Component: EuGENia, <http://www.eclipse.org/epsilon/doc/eugenia/>.



■ **Figure 2** Instance model for the cleaning robot example designed inside the provided graphical Jason modeling editor.

■ **Table 1** Jason concepts and their notations provided for the graphical concrete syntax.

<i>Concept</i>	<i>Notation</i>	<i>Concept</i>	<i>Notation</i>
Agent		Rule	
Belief Base		Plan	
Event Set		Context	
Rule Set		Body	
Plan Library		Internal Action	
Action Set		External Action	
Goal Set		Message	
Belief		Goal	
Triggering Event		Mental Note	

examples of Jason BDI architecture which is provided by Bordini and Hubner [3] for the scenario described in [6].

The screenshot from our EuGENia-based modeling tool, given in Figure 2, shows the instance model conforming to Jason metamodel designed for the cleaning robot example. The cleaning robot example consists of robots (agent instances) that collect and burn garbage. In our example, *robot1* is responsible for collecting garbage. In order to collect garbage, *robot1* has different plans. The agent adopted each plan from the plan library again given in the instance model. Also, each plan mainly covers the required action which is adopted from the action set. So, when we add an action to the action set of the model, it is available for any plan. This idea works not only for actions but also works for beliefs, goals, events and rules. The model given in Figure 2 shows that *robot1* agent's main goal is checking slots. The agent continues to search until finding garbage. When it finds garbage, it desires to carry garbage toward *robot2* which burns garbage. Whole system runs until there is no garbage in the environment.

5 Conclusion

A metamodel, which can be used for modeling Jason BDI agents, has been introduced in this paper. The metamodel provides the modeling of agents with including their belief bases, plans, and sets of events, rules and actions respectively. Derivation of a graphical concrete syntax based on this metamodel also enabled us to develop a graphical modeling tool for the MDD of Jason agents. In this tool, MAS models can be checked according to the constraints originated from the Jason metamodel definitions and hence conformance of the instance models is supplied.

Our next work will include automatic code generation from the instance models designed with the tool introduced in this paper. Hence, agent developers will achieve executables for Jason platform via MAS modeling. Following this work, our aim is to integrate Jason metamodel and its modeling tool into the tool set of our existing domain-specific MAS modeling language called SEA_ML [8]. Hence, it will be possible to model BDI agents in the platform-independent level and after executing a chain of model-to-model and model-to-text transformations, agent developers can automatically achieve the implementation of their MAS models in the Jason platform.

References

- 1 Carole Bernon, Massimo Cossentino, Marie-Pierre Gleizes, Paola Turci, and Franco Zambonelli. A study of some multi-agent meta-models. In *5th Int'l Workshop on Agent-Oriented Software Engineering*, volume 3382, pages 62–77, 2005.
- 2 Ghassan Beydoun, Graham Low, Brian Henderson-Sellers, Haralambos Mouratidis, Jorge J. Gomez-Sanz, Juan Pavon, and Cesar Gonzalez-Perez. FAML: A generic metamodel for MAS development. *IEEE Transactions on Software Engineering*, 35(6):841–863, 2009.
- 3 Rafael H. Bordini and Jomi F. Hübner. BDI agent programming in Agentspeak using Jason. In *Proceedings of the 6th International Conference on Computational Logic in Multi-Agent Systems*, pages 143–164, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11750734_9.
- 4 Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, 2007.
- 5 Michael E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, Massachusetts, 1987.
- 6 Lars Braubach, Alexander Pokahr, Daniel Moldt, and Winfried Lamersdorf. Goal representation for bdi agent systems. In *Proceedings of the Second International Conference on Programming Multi-Agent Systems*, pages 44–65, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/978-3-540-32260-3_3.
- 7 MMoharram Challenger, SSebla Demirkol, SSinem Getir, and Geylani Kardas. A domain specific metamodel for semantic web enabled multi-agent systems. In *1st International Workshop on Domain Specific Engineering*, volume 83, pages 177–186, 2011.
- 8 Moharram Challenger, Sebla Demirkol, Sinem Getir, Marjan Mernik, Geylani Kardas, and Tomaz Kosar. On the use of a domain-specific modeling language in the development of multiagent systems. *Engineering Applications of Artificial Intelligence*, 28:111–141, 2014.
- 9 Massimo Cossentino, Antonio Chella, Carmelo Lodato, Salvatore Lopes, Patrizia Ribino, and Valeria Seidita. A notation for modeling Jason-like BDI agents. In *Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on*, pages 12–19, July 2012. doi:10.1109/CISIS.2012.203.
- 10 Iván Garcia-Magarino. Towards the integration of the agent-oriented modeling diversity with a powertype-based language. *Computer Standards & Interfaces*, 36:941–952, 2014.
- 11 Enyo José Tavares Goncalves, Mariela Inés Cortes, Gustavo Augusto Lima Campos, Yrleyjander Salmito Lopes, Emmanuel Sávio Ssilva Freire, Viviane Torres da Silva, Kleiner Silva Farias de Oliveira, and Marcos António de Oliveira. MAS-ML2.0: Supporting the modelling of multi-agent systems with different agent architectures. *The Journal of Systems and Software*, 108:77–109, 2015.
- 12 Christian Hahn. A domain specific modeling language for multiagent systems. In *7th Int'l Conf. on Autonomous agents and Multi-agent systems (AAMAS 2008)*, pages 223–240, 2008.
- 13 Christian Hahn, Cristián Madrigal-Mora, and Klaus Fischer. A platform-independent metamodel for multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 18(2):239–266, 2009.
- 14 Brian Henderson-Sellers and Paolo Giorgini. *Agent-oriented Methodologies*. Idea Group Publishing, 2005.
- 15 Geylani Kardas. Model-driven development of multiagent systems: a survey and evaluation. *The Knowledge Engineering Review*, 28(4):479–503, 2013.
- 16 Geylani Kardas, Erdem Eser Ekinici, Bekir Afsar, Oguz Dikenelli, and N. Yasemin Topaloglu. Modeling tools for platform specific design of multi-agent systems. In Lars Braubach,

- Wiebe van der Hoek, Paolo Petta, and Alexander Pokahr, editors, *Multiagent System Technologies: 7th German Conference, MATES 2009*, pages 202–207. Springer, Berlin, Heidelberg, 2009. doi:10.1007/978-3-642-04143-3_20.
- 17 Uirá Kulesza, Alessandro Garcia, Carlos Lucena, and Paulo Alencar. A generative approach for multi-agent system development. In Ricardo Choren, Alessandro Garcia, Carlos Lucena, and Alexander Romanovsky, editors, *Software Engineering for Multi-Agent Systems III: Research Issues and Practical Applications*, pages 52–69. Springer, Berlin, Heidelberg, 2005. doi:10.1007/978-3-540-31846-0_4.
 - 18 Ambra Molesini, Enrico Denti, and Andrea Omicini. MAS meta-models on test: UML vs. OPM in the SODA case study. In *Proceedings of the 4th International Central and Eastern European Conference on Multi-Agent Systems and Applications*, pages 163–172, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11559221_17.
 - 19 James Odell, Marian Nodine, and Renato Levy. A metamodel for agents, roles, and groups. In James Odell, Paolo Giorgini, and Jörg P. Müller, editors, *Agent-Oriented Software Engineering V*, pages 78–92. Springer, Berlin, Heidelberg, 2005. doi:10.1007/978-3-540-30578-1_6.
 - 20 Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.
 - 21 Anand Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Eindhoven, The Netherlands, Lecture Notes in Computer Science*, volume 1038, pages 42–55, 1996.
 - 22 Anand S. Rao and Michael P. Georgeff. Decision procedures for BDI logics. *Journal of Logic and Computation*, 8(3):293–343, 1998.