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/OASIcs.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

implementation platforms such as JACK\(^2\), JADE\(^3\) or JADEX\(^4\). 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 platform\(^5\) 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


\(^3\) JAVA Agent DEvelopment Framework, http://jade.tilab.com/

\(^4\) JADEX Active Components, https://www.activecomponents.org/#/project/news

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 AgentSpeak 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 EuGENia\textsuperscript{6} 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 modeling. 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

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.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Notation</th>
<th>Concept</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td>🌐</td>
<td>Rule</td>
<td>📝</td>
</tr>
<tr>
<td>Belief Base</td>
<td>📜</td>
<td>Plan</td>
<td>🕺</td>
</tr>
<tr>
<td>Event Set</td>
<td>📜</td>
<td>Context</td>
<td>🕺</td>
</tr>
<tr>
<td>Rule Set</td>
<td>📜</td>
<td>Body</td>
<td>🕺</td>
</tr>
<tr>
<td>Plan Library</td>
<td>🌐</td>
<td>Internal Action</td>
<td>🕺</td>
</tr>
<tr>
<td>Action Set</td>
<td>🌐</td>
<td>External Action</td>
<td>🕺</td>
</tr>
<tr>
<td>Goal Set</td>
<td>🌐</td>
<td>Message</td>
<td>🕺</td>
</tr>
<tr>
<td>Belief</td>
<td>🌐</td>
<td>Goal</td>
<td>🕺</td>
</tr>
<tr>
<td>Triggering Event</td>
<td>🌐</td>
<td>Mental Note</td>
<td>🕺</td>
</tr>
</tbody>
</table>

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


