

Accepted Manuscript

Domain-specific Modelling Language for Belief-Desire-Intention Software Agents

Geylani Kardas, Baris Tekin Tezel, Moharram Challenger

DOI: [10.1049/iet-sen.2017.0094](https://doi.org/10.1049/iet-sen.2017.0094)



To appear in: *IET Software*

Published online: 4 April 2018

Please cite this article as: Geylani Kardas, Baris Tekin Tezel, Moharram Challenger, Domain-specific Modelling Language for Belief-Desire-Intention Software Agents, IET Software, doi: [10.1049/iet-sen.2017.0094](https://doi.org/10.1049/iet-sen.2017.0094).

This is a PDF file of an unedited manuscript that has been accepted for publication. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

Domain-specific Modelling Language for Belief-Desire-Intention Software Agents

Geylani Kardas¹, Baris Tekin Tezel², Moharram Challenger¹

¹International Computer Institute, Ege University, 35100, Izmir, Turkey

²Computer Science Department, Dokuz Eylul University, 35160, Buca, Izmir, Turkey

geylani.kardas@ege.edu.tr, baris.tezel@deu.edu.tr, moharram.challenger@ege.edu.tr

ABSTRACT

Development of software agents according to Belief-Desire-Intention (BDI) model usually becomes challenging due to autonomy, distributedness and openness of multi-agent systems (MAS). Hence, in this paper, a domain-specific modeling language (DSML), called DSML4BDI, is introduced to support development of BDI agents. The syntax of the language provides the design of agent components required for the construction of the system according to the specifications of BDI architecture. The implementation of designed MAS on Jason BDI platform is also possible via model-to-text transformations built in the DSML. The comparative evaluation results showed that a significant amount of artifacts required for the exact MAS implementation can be automatically achieved by employing DSML4BDI. Moreover, time needed for developing a BDI agent system from scratch can be reduced to one-third in case of using DSML4BDI. Finally, qualitative assessment, based on the developers' feedback, exposed how DSML4BDI facilitates development of BDI agents.

1. INTRODUCTION

Design and implementation of software agents and multi-agent systems (MAS) according to Belief-Desire-Intention (BDI) model [1] are performed in many agent-oriented software engineering (AOSE) applications (e.g. [2-4]). In BDI architecture, agents constantly monitor their environment and respond 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 [5].

Although BDI model enables representation of agent internals and supports the composition of reactive or proactive agent behaviors, development of software required for constructing BDI agents usually becomes challenging and time-consuming due to autonomy, distributedness and openness of MAS. Working in a higher abstraction layer and modeling BDI components within a model-driven development (MDD) process before going into depths of implementation may help building such agent systems. Hence, in this paper, a domain-specific modeling language (DSML), called DSML4BDI, is introduced to support MDD of BDI agents. Modeling all BDI components and relations is possible by using the graphical syntax of DSML4BDI. Furthermore, the language is also capable of automatic code generation, which significantly facilitates implementing BDI agents. In this paper, we also discuss how DSML4BDI enhances development of software agents within the scope of a comparative language evaluation in which both quantitative and qualitative analyses take place.

The rest of the paper is organized as follows: Syntax descriptions of DSML4BDI are discussed in Section 2. Section 3 covers operational semantics and code generation facilities of the language. Assessment of DSML4BDI and achieved results are discussed in Section 4. Related work is given in Section 5 and Section 6 concludes the paper.

2. ABSTRACT AND CONCRETE SYNTAXES

DSML4BDI's abstract syntax is based on a metamodel which provides the meta-entities and their relations for modeling both internal architecture of BDI agents and the organization of MAS. The metamodel given here is an improved version of the metamodel we introduced in [5]. Not only providing an all-embracing model of BDI architecture with new entities, the improved metamodel also elaborates each entity with its inner attributes / properties which are not included in [5]. Fig. 1 shows the Ecore [6] encoded DSML4BDI metamodel in which essential meta-entities are coloured. During the discussion below, all entities are given in the text with italic font.

Although a MAS is mainly composed of agents, DSML4BDI metamodel includes nine different entities other than the *Agent* entity, namely *LogicalExpressionSet*, *RuleSet*, *PlanLibrary*, *ActionSet*, *GoalSet*, *EventSet*, *BeliefBase*, *MentalNoteSet* and *FormulaSet* both for the complete representation of the MAS and internal logic structure of individual agents.

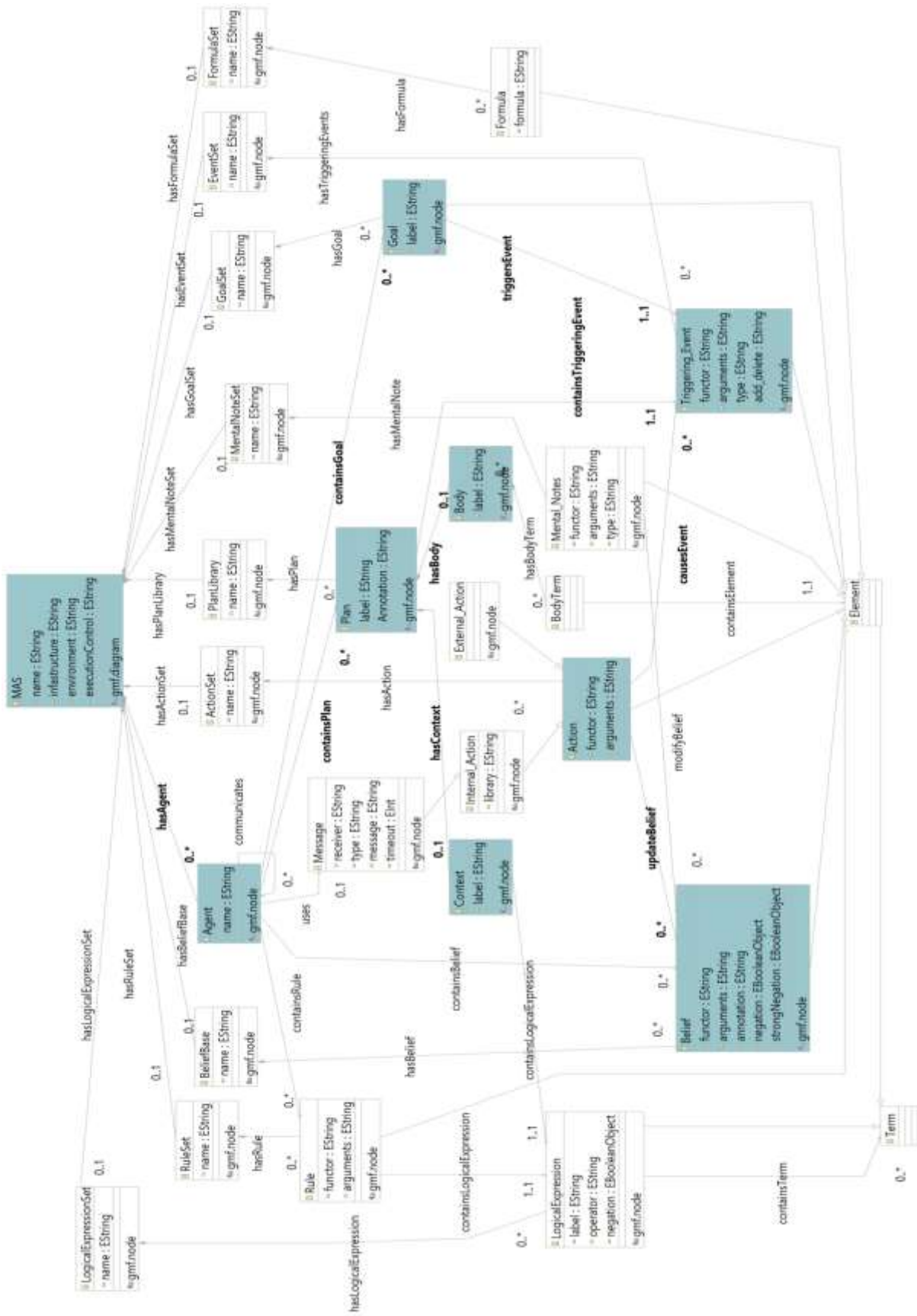


Fig. 1: The metamodel of DSML4BDI modeling language

BeliefBase is consisted of possible beliefs which may be adopted by agents. *BeliefBase* of a MAS can be different from *BeliefBase* of individual agents. It mostly has a static structure while *BeliefBase* of an agent can be changed during the execution of reasoning cycles. In addition, each *Belief* represents knowledge about an agent or the environment.

EventSet contains events within the environment. In our approach, each event is represented by a triggering event which is also a meta-element (*Triggering_Event*) included in the DSML4BDI metamodel. *Belief* or goal changes inside an agent's mind or an action performed by an agent may cause happening an event. *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 agent plans.

ActionSet stores actions that can be used by all agents. *Action* meta-entity represents what an agent can perform. The metamodel also supports two specializations of the actions: internal actions and external actions. Internal actions (represented by the *Internal_Action* meta-entity) are executed inside an agent's mind. Therefore, they cannot directly change the environment. Communication actions of an agent, called as Messages in the metamodel, are also internal actions. However, external actions (represented by the *External_Action* meta-entity) directly change the environment.

GoalSet brings together all candidate *Goals* which will be achieved by each agent of a MAS. In fact, goals are the desired states reached by the execution of agent plans. There are two types of goals, including achievement goals and test goals. An achievement goal represents a state of the environment, that 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. Each goal has to be related with a certain triggering event. Hence, these goal types are included in the metamodel as being attributes of *Triggering_Event* associated with the goal of interest.

Reactions of an agent against events are represented by plans in DSML4BDI language. A *Plan*, which constitutes 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 *LogicalExpressions* which allow deducing based on the knowledge. The *Body* of a plan contains a list of actions. In addition, a plan also has sub-goals and *Mental_Notes* which modify *BeliefBase* of an agent. Hence, the body of a plan is represented as being a composition of all these related *BodyTerms*. Besides, mental notes pertaining to plans are stored in *MentalNoteSets*. All possible plans which can be used by an agent, are covered in a *PlanLibrary* meta-element.

Finally, *LogicalExpressionSet* is composed of *LogicalExpressions* that are to be interpreted as being either true or false. A *LogicalExpression* is formed by *Terms* which can be simple *Elements* or again *LogicalExpressions*. Terms contained in *LogicalExpression* instances are connected to each other by logical operators which are the attributes of *LogicalExpressions*. On the other hand, a *LogicalExpression* has a Boolean negation attribute. This attribute represents the complement in logic. If it takes true value, then *LogicalExpression* produces logical complement of its own value. *FormulaSet* keeps *Formulas* which represent pure mathematical expressions. These can be used in *LogicalExpressions* too.

























The concrete syntax of a language is the set of notations responsible for the presentation and construction of the language. Taking into account DSML specifications, the concrete syntax mainly provides a mapping between the meta-elements and their representations for the instance models of meta-model. Hence, we also provided a graphical concrete syntax which maps DSML4BDI's abstract syntax elements to their graphical notations. In order to construct DSML4BDI's concrete syntax, we benefited from the features of Sirius [7] modeling environment. Both providing a tool for implementing a graphical editor from an Ecore metamodel and allowing one to define dedicated editors including diagrams, tables or trees based on a viewpoint approach caused us to build DSML4BDI's graphical modeling toolset on Sirius environment in this study.

Four diagram types, named as MAS, Agent, Plan and Logical Expression, can be used for reflecting different perspectives of a DSML4BDI instance. MAS diagram is the main diagram of the tool. In this diagram, MAS organization of BDI agents is expressed with including main elements and their relationship of the modeled MAS. Agent diagrams show internal agent structure which is composed of plans, beliefs, rules and goals. Plan diagrams are needed for designing plans of the agents in the MAS organization. Finally, logical expressions, which might be used in any agent plan or rule, are created in Logical Expression diagrams. Some significant graphical notations pertaining to the abstract syntax elements, covered inside the DSML4BDI diagram types, are listed in Table 1.

A screenshot from Sirius-based IDE of DSML4BDI is given in Fig. 2. Developers can create BDI models of agent systems conforming to DSML4BDI specifications by simply drag-and-dropping required items from the palette residing at the right-side of the modeling environment. Constraint-checks and any other static semantics controls are automatically made inside the environment. Fig. 2 also contains a part of an instance DSML4BDI model designed for Garbage Collector MAS which is well-known in AOSE field. The system has two types of agents, called destructor and collector. The first task of destructor agents is to report the location of the garbage in the environment to collector agents. The other task of destructor agents is to burn garbage brought by collector agents. Collector agents go to

the location of garbage which is told by destructor agents, take the garbage and bring it back to destructor agents. After the garbage is delivered, collector agents should wait for the new messages.

Table 1 Some of the concepts and their notations provided for DSML4BDI's graphical concrete syntax

Concept	Notation	Concept	Notation
Agent		MAS	
Action		Mental Note	
Action Set		Message	
Belief		Plan	
Belief Base		Plan Library	
Body		Rule	
Context		Rule Set	
Event		Formula	
Event Set		Formula Set	
External Action		Logical Expression	
Internal Action		Logical Expression Set	
Goal		Goal Set	

3. OPERATIONAL SEMANTICS

A complete DSML definition cannot be made only by specifying the notions and their representations. It also requires that one provides semantics of language concepts generally in terms of the meanings of other concepts which are already established. Hence, in this study, the elements of DSML4BDI's abstract syntax are mapped into the concepts of Jason [2], which is an interpreter for an extended version of a Prolog-like logic programming language for BDI agents, called AgentSpeak. With providing a Java-based interpreter, Jason extends the expressiveness of AgentSpeak during implementation of cognitive agents. Mapping between DSML4BDI and Jason entities leads a series of model-to-text (M2T) transformations, which enables the construction of DSML4BDI semantics on Jason platform and hence automatic generation of executable Jason codes for the corresponding DSML4BDI model instances is possible.

We used Aceleo [8], a well-known implementation of Object Management Group's (OMG) Model to Text Language (MTL) standard, for the implementation of the required M2T transformations. Having both a simple syntax and an appropriate IDE and the interoperability with Sirius are the main features caused us to choose Aceleo as the code generator. In order to automatically generate

Jason codes for the real implementation of DSML4BDI models, M2T rules are executed on Ecore representations of MAS model instances created inside DSML4BDI's modeling environment.

When considering Jason platform, each agent is represented with an ASL file including codes written in AgentSpeak language for the inner structure of the related agent according to BDI architecture. 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 events for which the plan is suitable. The context represents whether the plan is applicable according to beliefs of the agent. Body is a sequence of basic actions and/or subgoals.

Fig. 3 includes an excerpt from written Acceleo rules which is perhaps the most important part of the transformation phase since all agent plans in an ASL file are generated by these rules. Between lines 4 and 6, the triggering event of DSML4BDI agent plan is created and added to ASL file. In line 7, the context part of the plan is generated if it exists in the DSML4BDI model instance. Remaining Acceleo codes (starting from line 9) complete the generation of the body of agent plan with including all modeled body components such as internal actions, external actions, goals and mental notes.

In addition, Jason provides MAS specification files and fundamental Java classes overridden to define "behaviour" of complete MAS environment. Acceleo templates for the automatic generation of such specification files from DSML4BDI models are also prepared in our study. However, they are not discussed here due to space limitations.

In Fig. 4, a snippet from auto-generated Jason ASL files achieved as the result of executing abovementioned M2T transformations is given. It includes codes for the collector agent modeled in Fig. 2.

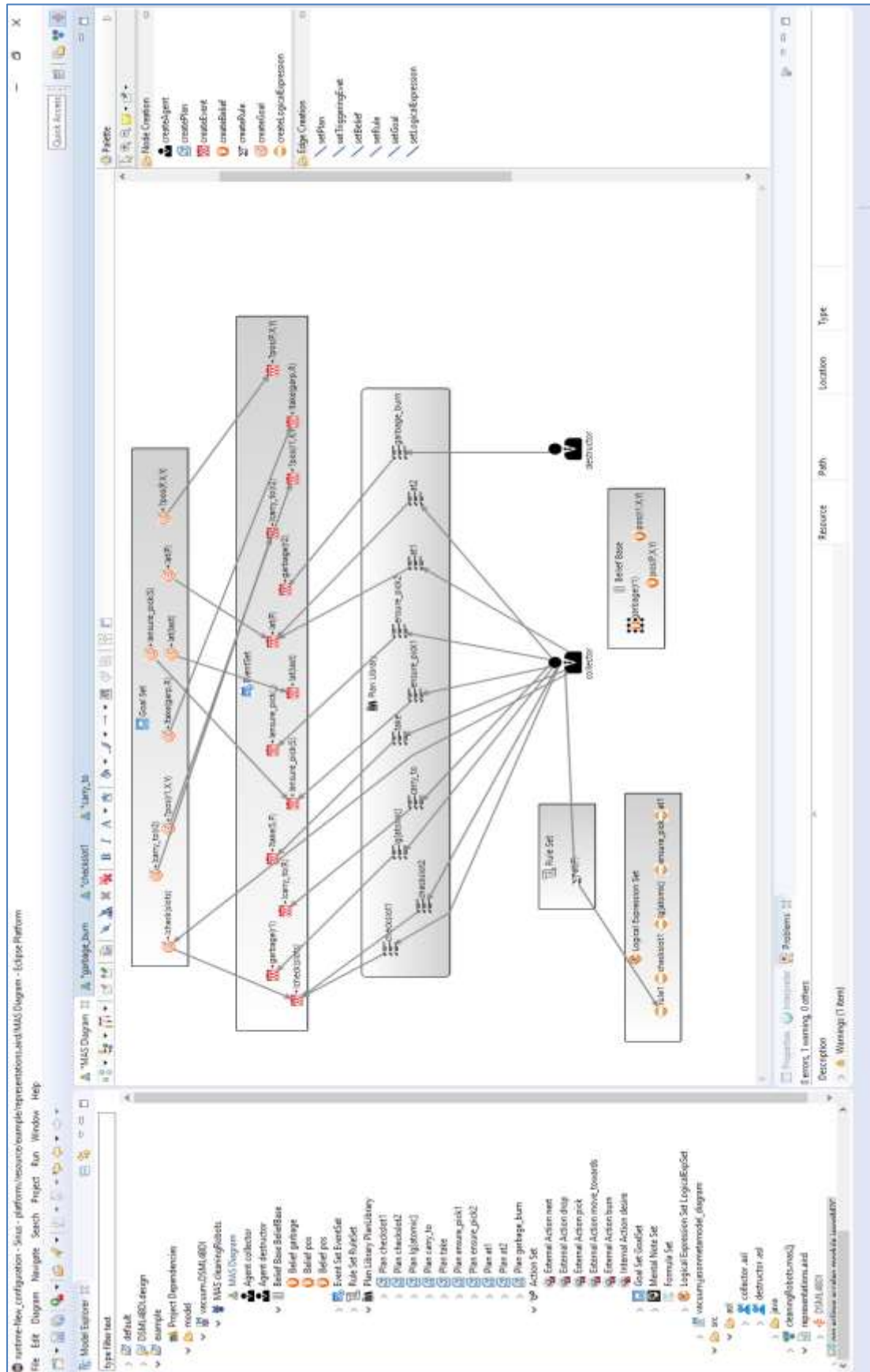


Fig. 2: DSML4BDI's IDE for MDD of BDI agents

```

1. /* Plans */
2. [for (p : Plan | self.containsPlan)]
3. @[p.label]
4. +[p.containsTriggeringEvent.type]
5. [p.containsTriggeringEvent.functor][[for(a: String |
6. p.containsTriggeringEvent.arguments) separator(',') ][a][/for]]
7. [if (p.hasContext.containsLogicalExpression->null)]
8. :[writeLogicalExpression(p.hasContext.containsLogicalExpression)]
9. [if][if (p.hasBody.hasBodyTerm->notEmpty())<-
10. [for (it : BodyTerm | p.hasBody.hasBodyTerm)separator(',') ]
11. [let bT : Element = it.containsElement
12. [if(bT.ocIsKindOf(External_Action))]
13. [bT.ocIsType(External_Action).functor]
14. ([[for(ar:String|bT.ocIsType(External_Action).arguments)
15. separator(',')][ar][/for]][/if][if(bT.ocIsKindOf(Internal_Action))]
16. [if(bT.ocIsType(Internal_Action).library->null)]
17. [bT.ocIsType(Internal_Action).library]
18. [if][bT.ocIsType(Internal_Action).functor]
19. ([[for(ar:String|bT.ocIsType(Internal_Action).arguments)
20. separator(',')][ar][/for]][/if][if(bT.ocIsKindOf(Goal))]
21. [bT.ocIsType(Goal).triggersEvent.type]
22. [bT.ocIsType(Goal).triggersEvent.functor]
23. ([[for(ar:String|bT.ocIsType(Goal).triggersEvent.arguments)
24. separator(',')][ar][/for]][/if][if(bT.ocIsKindOf(Mental_Notes))]
25. [bT.ocIsType(Mental_Notes).type]
26. [bT.ocIsType(Mental_Notes).functor]
27. ([[for(ar:String|bT.ocIsType(Mental_Notes).arguments)separator(',') ]
28. [ar][/for]][/if][if][/if][/if][/if]
29. [/for]

```

Fig. 3: An excerpt from Aceleo rules for creating plans in ASL files

```

// Agent collector
/* Initial beliefs */
/* Initial rules */
at(P):-{pos(P,X,Y)&pos(r1,X,Y)}.
/* Initial goals */
!check(slots).
/* Plans */
@checkslot1
+!check(slots)
: not (garbage(r1))<-next(slot);!check(slots).
@checkslot2
+!check(slots)
.
@lg[atomic]
+garbage(r1)
: not (.desire(carry_to(r2)))<-!carry_to(r2).
@carry_to
+!carry_to(R)
<-?pos(r1,X,Y);+pos(last,X,Y);!take(garp,R);!at(last);!check(slots).
@take
+!take(5,P)
<-!ensure_pick(S);!at(P);drop(S).
@ensure_pick1
+!ensure_pick(S)
:(garbage(r1))<-pick(garb);!ensure_pick(S).
@ensure_pick2
+!ensure_pick(_)
.
@at1
+!at(P)
:(at(P)).
@at2
+!at(P)
<-?pos(P,X,Y);move_towards(X,Y);!at(P).

```

Fig. 4: Auto-generated Jason ASL codes of the collector agent modeled in Fig. 2

4. EVALUATION

An evaluation of DSML4BDI was performed in this study by considering MAS developers' perspective. We believe that some kind of comparative evaluation may help clarifying the feasibility of choosing DSML4BDI for the development of BDI agent systems instead of the classical code-centric way of development. For this purpose, we adopted the evaluation framework proposed in [9] which provides the systematic assessment of both the language constructs and the use of agent DSMLs according to various dimensions and criteria. To the best of our knowledge, it is currently the only evaluation framework specific to the MAS DSMLs and guides the assessment of model-driven agent development methodologies.

4.1. Overview of the evaluation approach

Considering the evaluation dimensions and criteria given in [9], the scope of our evaluation in here mainly covers Development Sub-dimension (under Execution Dimension) and User Perspective Sub-dimension (under Quality Dimension). Therefore, the evaluation criteria pertaining to these dimensions, called Output Performance (Generation Performance), Development Time Performance and Qualitative Analysis by a Questionnaire are taken into consideration. We revised these dimensions and criteria to make them more meaningful and appropriate for our quantitative evaluation. Also, the evaluation was separated into two parts: 1) quantitative analysis including generation performance evaluation and development time evaluation 2) qualitative analysis including user perspective by using a questionnaire.

The evaluation performed in this study was realized by two groups of evaluators each having 4 software agent developers. In each group, we had 2 Ph.D. candidates and 2 M.Sc. students. All the evaluators were students of computer related fields and passed graduate courses called Advanced Software Engineering, Agent-oriented Software Development and Multi-agent Systems, taught in Computer Engineering Department and International Computer Institute of Ege University. They had at least 2 years MAS design and implementation experience covering the application of AOSE methodologies and using some agent development APIs like JADE and JACK. In addition, all evaluators were familiar with software engineering methodologies, mostly based on UML and having at least 5 years' experience on using various IDEs such as Eclipse, NetBeans and IntelliJ IDEA. Four evaluators were also working in industry at the time of this evaluation performed and possessed the experience of developing software in industrial scale (2 years on the average). The evaluators were assigned into groups in a way that the groups are balanced in terms of programming experience level of each group. First group, called Group A, utilized DSML4BDI during the development of a MAS provided within a use case study. Second group, called Group B, did not use any domain-specific

modeling tool including DSML4BDI; instead, they followed a code-centric approach and used generic software development tools and technologies for the development of the same MAS. Inside the same use case study, evaluation results were achieved from both groups and their analysis are reported in this section.

Considering the execution of the case study, first, both teams received a review on Jason as the BDI agent programming language with including an example for demonstration. This step ensures countervailing their level of familiarity to the target language and removes the threat of validity regarding evaluators. Next, Group A received an introduction to DSML4BDI and its IDE for modelling and code generation. The time spent for this step is considered as an overhead for this group in the analysis phase. This overhead is ignorable comparing to the whole process since this language introduction is performed only once for whole team before the beginning of MAS development. Then, a briefing to the case study and its requirements was made for the both groups. Finally, the evaluators realized MAS development by going through steps of analyzing the use case, designing/modelling the system, implementing/generation of code and testing software. Elapsed times were recorded for each developer and each step, which are analyzed in the next sub-section. Also, generated and/or developed artifacts of both groups are evaluated to find out especially the generation performance of DSML4BDI. Development of software required for Garbage Collector MAS described in Section 2 is taken into account as the case study during our evaluation.

4.2. Results and Discussion

Quantitative analysis of the assessment results according to generation performance and development time criteria is given in Section 4.2.1. while qualitative findings on questionnaire-based analysis are discussed in Section 4.2.2.

4.2.1. Quantitative Analysis

We analyzed DSML4BDI's performance quantitatively by assessing both its power on the generation of artifacts (called generation performance), and usability of the tools through the time saved for developers during whole development process (called development time performance).

Generation Performance:

Considering the generative aspect of DSML4BDI, the essential artifacts including ASL and MAS2J files are generated with their architectural code. Specifically, the interaction between BDI agents is taken into consideration. To calculate the generation performance, we made a comparison between the automatically generated code and the delta code added by Group A to complete the code.

Performance evaluation is fulfilled by comparing the percentage of artifacts automatically generated and manually developed. These artifacts are number of files, Lines of Code (LoC) of ASL files, and LoC of MAS2J files. The average of two LoC is calculated to give the overall ratio of the generated LoC and the cumulative average of all artifacts is calculated to show the development performance considering whole system. These results are presented in Fig. 5 to ease their comparison and analysis.

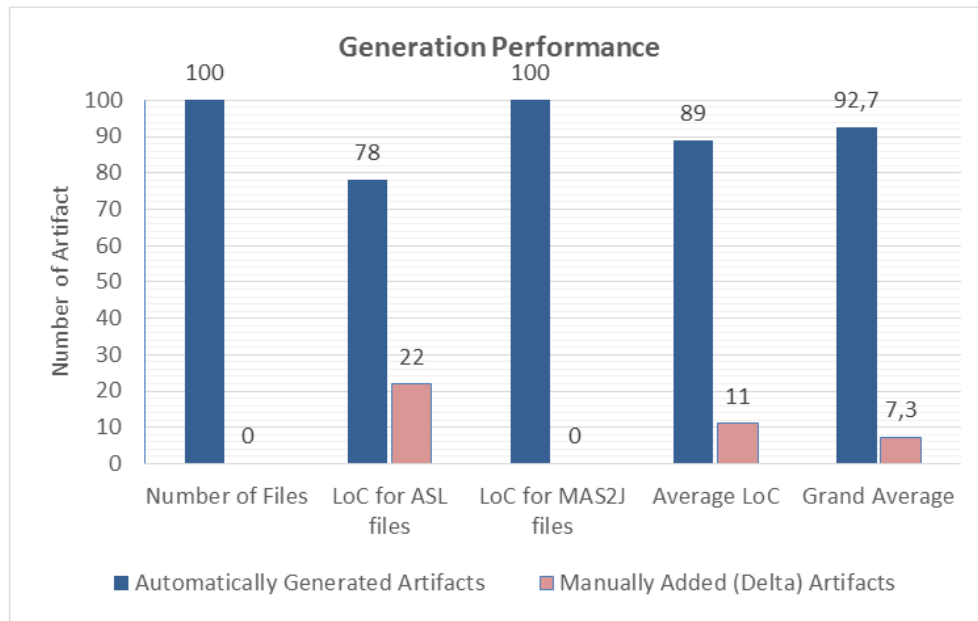


Fig. 5: Generative performance of DSML4BDI

As seen in Fig. 5, all files needed for garbage collector MAS are generated. The production of all files is crucial in terms of executable code generation. Thus, developers do not need to add new files to the system. Moreover, all necessary interconnections between these files are provided. The reason why the generation of all files is possible is that in Jason structure, each agent needs at least one ASL file and there is need for exactly one MAS2J file for each MAS. Therefore, DSML4BDI model can easily generate these files for instantiated elements in MAS. This leads to a fruitful generation of number of files for the system.

In addition, all required MAS2J files are successfully generated from designed models. This is possible because MAS modelling elements in DSML4BDI represent the MAS structure of BDI agents with including structural information like: MAS properties, number of agents in the MAS, the properties of underlying infrastructure, and environment properties.

According to the results in Fig. 5, the average generated LoC rate is 89%. However, it is worth indicating that generation rate for the developer in Group A, who is expert on the tool and BDI agents, was 100% in this study.

On the other hand, the developer with the lowest LoC rate is 75%. The difference here comes from the quality of the model created by the developers in DSML4BDI, depending on their familiarity with BDI MAS modelling. Therefore, it is expected that with more practice in some other use cases, the developer will be able to provide more qualitative model leading to more generated code at the beginning. Finally, we can see from Fig. 5 that 92.7% of the whole artifacts are generated for average developers and only 7.3% needed to be developed by these developers to achieve a fully functional system.

Development Time Performance:

To evaluate DSML4BDI with its supporting tool by considering the time it can save for developers, we compared and analyzed the times which were recorded during the development efforts of Group A (DSML4BDI users) and Group B (users who do not benefit from any domain-specific tool). It is worth stating that Group B evaluators could use any general modelling tools such as UML. Times recorded for the developers include all steps of MAS development discussed in Section 4.1. The average time for each of these phases were calculated separately for both groups. Obtained results are presented in Fig. 6 to facilitate comparing peer phases for both groups. Based on the results given in Fig. 6, the followings can be deduced for each of the development steps:

- Analysis of the use case is part of the development. It is not depended on any tool, so, both teams have almost similar analysis time. In fact, this step is only dependent on the complexity of the problem and is independent of the development language. Therefore, the difference between times can be ignored.
- The modeling time for Group A is close to the design time for Group B, but the one for Group A is slightly more than of Group B. Although both DSML4BDI (used by Group A) and UML (used by Group B) provide drag-and-drop utility, DSML4BDI works with concepts specific to agent domain in design time with including more detailed elements and their attributes which will be used during generation. Also, DSML4BDI has static semantics checks enabling designers to avoid semantical errors which probably take more time to correct them later. Thus, modeling with DSML4BDI's tool extends the length of the design time slightly. This kind of detailed modelling was not considered by Group B developers since they mostly created the models as documentation artifacts. However, that elapsed time is an investment by

DSML4BDI users to gain more detailed and accurate codes which will be generated automatically.

- Perhaps, the implementation is the most important step to compare the results of two groups. While many files and codes were automatically generated for Group A, Group B wrote required codes manually based on their design. Because DSML4BDI succeeded in automatic generation of agent codes by utilizing the products acquired from detailed modeling. However, Group A developers still needed to complete the generated code by adding delta codes to have a fully executable program. Consequently, the implementation time required by Group B is about 9 times more than generation and implementation time elapsed for Group A's efforts.
- In the test/error detection step, both groups tried to find syntactic and semantic errors of the programs. While Group B developers needed to find errors in the entire code, Group A developers only dealt with delta code added, which is much less than the generated code. Because generated code constitutes most of the final code and consists of error-free and almost-ready architectural code that does not need to be re-validated since it is verified by DSML4BDI constraint checks. Hence, the time required to find errors for Group A is four times less than for Group B.
- Finally, considering the total development time for both groups, Group A completed the whole development process about 3 times faster than Group B due to using DSML4BDI and its IDE.

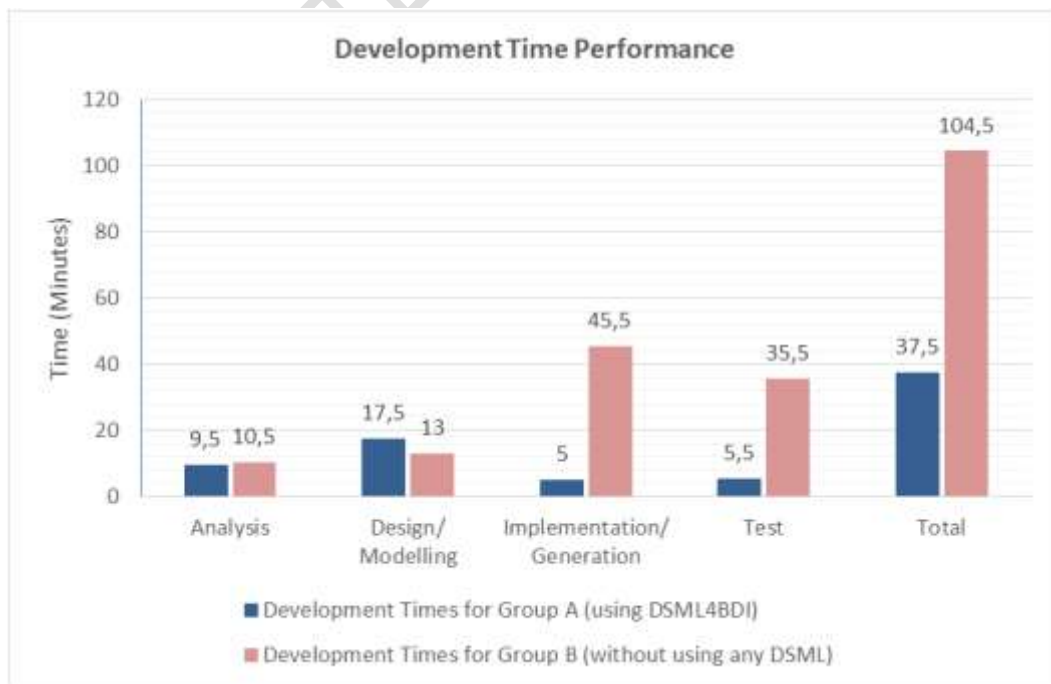


Fig. 6: Development time performance of DSML4BDI

4.2.2. Qualitative Analysis

To evaluate DSML4BDI qualitatively, a questionnaire was provided, including five open-ended questions, which were answered by the evaluators of Group A, who used DSML4BDI in their agent development process. These questions are:

1. How does DSML4BDI make MAS development easier?
2. How is DSML4BDI appropriate for BDI agent development?
3. Do you think DSML4BDI is powerful enough to model the general MAS structure? Why?
4. Do you feel DSML4BDI's IDE easy to use? How?
5. Did you encounter any difficulties while using DSML4BDI? If any, please provide your suggestions to resolve them.

Most of the evaluators answered the first question by indicating that designing inside a domain-specific graphical editor helped them for determining and visualizing the details of the system from scratch. One evaluator also stated that DSML4BDI indeed facilitates the construction of agents by enabling checking the domain rules on the designed models and automatically generating codes with the required agent interconnections. Another evaluator found code generation feature satisfying since very little programming is needed to add delta codes to achieve fully functional MAS programs. One added that the capability of reusing previously modeled elements (e.g. agent plans) during modeling with DSML4BDI accelerates the development process.

Regarding the responses for the second question, the evaluators agreed on the appropriateness of DSML4BDI for BDI agent development by mostly indicating that the language provides an all-embracing model of fundamental BDI elements (e.g. plan, goal, belief, event) and their relations. One evaluator added that the defined syntax and semantics fully supports the construction of BDI agent systems with varying sizes. Another evaluator stated that the graphical syntax especially helped him to the composition of heavy logical and mathematical rule expressions required for BDI agents. The evaluators also acknowledged DSML4BDI's support for Jason, one of the well-known BDI agent frameworks. Moreover, one evaluator confirmed DSML4BDI's appropriateness in his reply by stating that the auto-generated code is complete and there is no need for any delta code (especially for BDI agent plans) in most situations.

The evaluators accredited DSML4BDI's modeling capability for the general MAS structure with emphasizing that the modeling language enhanced the system analysis and design by providing all needed MAS domain concepts and their relations as the first class elements. Two evaluators also

added that both agent internals and interactions between agents can be adequately modeled and implemented using the language.

The IDE of DSML4BDI was found easy-to-use in general. One evaluator supported that view by accentuating how the drag-and-drop feature facilitates MAS modeling. Another evaluator stated that the developers can instantly see the effect of adding or removing an element to/from a diagram in the remaining diagrams due to the synchronization between DSML4BDI diagrams and, that definitely makes the system modelling easier inside several viewpoints. One evaluator had the opinion that the DSML4BDI's user-friendly interface empowers the modelling process by guiding the user with the help of semantic controls. Another one found using the modeling palette in the IDE comfortable since icons for the modeling elements are understandable and denotative for MAS concepts.

Taking into account the answers received for the last question, some new agent relations and plan attributes, found missing by the developers, were added into the language's metamodel. Corresponding syntax and semantic modifications were realized for the improved version of DSML4BDI, which is presented in this paper. Visual organization of DSML4BDI concepts and relations inside the palette of the modeling editor was also re-arranged according to the feedbacks gained from the evaluators since some of the evaluators found the arrangement of these components inside the IDE a bit complicated. Some evaluators also admitted that using the IDE is slightly confusing at the beginning. However, they also confirmed such confusion mainly originates from evaluators' unfamiliarity to a BDI agent development tool like DSML4BDI. It was the first time for evaluators to experience such a DSML for MAS.

Finally, as it is the case in any evaluation study, there are also some risks and threats to the validity of the performed evaluation. First, qualitative evaluation of DSML4BDI, discussed above, was naturally subjective and mostly based on each evaluator's own comparison between methods using and without using DSML4BDI. However, we think that such self-assessment is convenient and acceptable since Group A evaluators possess significant experience on designing agent systems by using classical software development approaches and implementing MASs by utilizing only general purpose languages.

In addition, we preferred to use two different groups instead of a single group for the evaluation, which may also be a threat to validity. Using a single developer group for MAS development with or without DSML4BDI provides the advantage of having exactly the same group of developers and knowledge. However, we think that if the same group were employed for both assessments, e.g. they first developed the MAS with their usual approaches and then developed the same MAS using DSML4BDI (or vice versa), that would probably affect the result of evaluation since they unavoidably

would reflect their experience on the first evaluation to the second. Using two groups eliminated this risk in our study, but this time we needed to keep the level of knowledge and experience of two different groups the same. For this purpose, we strongly cared on forming the groups having almost the same educational background and experience on software engineering and MAS development.

5. RELATED WORK

As indicated in [10], application of MDD and development of DSMLs for MAS emerged in AOSE field especially for the last decade. Among these studies, Agent-DSL [11] is used for modeling agent features, like knowledge, interaction and autonomy by presenting a metamodel. Two agent modeling languages are introduced in [12] considering syntax definitions rather than operational language semantics. Studies like [13,14] also discuss MDD of agent systems by introducing a series of transformations on MAS metamodels in different abstractions. Although those transformations may guide to construct some sort of semantics, related studies describe MAS development methodologies instead of specifying a complete DSML. In addition, there exist MAS metamodel proposals (e.g. [15-17]) from which abstract syntaxes of MAS DSMLs can originate. In our work, a MAS metamodel is also presented similar to abovementioned studies. However, more than providing just an abstract syntax based on this metamodel, DSML4BDI is a full-fledged modeling language with including all syntax and semantics constructs required for MDD of agents according to well-known BDI principles.

Taking into consideration the work in agent DSMLs, the language in [18] enables MAS specification by having an abstract syntax structured into several aspects each focusing on a specific viewpoint of a MAS. Another DSML is provided for MASs in [19] based on EMF [6]. The language supports modeling of agents according to one of the specific MAS methodologies called Prometheus. A similar study [20] proposes an MDD technique for the definition of agent-oriented engineering process models according to another specific MAS development methodology called INGENIAS.

The language described in [21] provides a way of code generation from textual agent descriptions for mobile agent systems. The work conducted in [22] aims at creating a UML-based agent modeling language, which is able to model various types of agent internal architectures. However, the current version of the language does not support any code generation, which prevents the execution of modeled agent systems. SEA_ML language introduced in [23] supports the execution of modeled agents over a series of model-to-code transformations enabling the construction of interactions between agents and semantic web services.

In a recent work [24], how a model-driven framework can be constructed to develop BDI agents by proposing strategic, tactical and operational views is investigated. Although it is possible to convert generated dependencies to BDI agents, the implementation of the required transformations and code generation are not included in the study. The work introduced in [25] aims at modeling Jason BDI agents. However, the work only consists of presenting a metamodel, not a complete language implementation like DSML4BDI and does not support the reusability of same concepts (e.g. beliefs, plans, rules) for different agents of a MAS as is the case with DSML4BDI.

Differentiating from abovementioned MAS DSML studies [18-23], DSML4BDI especially facilitates both modeling and creation of agent BDI rules and enables the achievement of all artifacts required for the implementation. Although visual MAS modeling and code generation are also provided in other DSMLs at some degree, both the quantitative and qualitative assessment of these language features are not taken into account which make difficult to determine the fruitfulness of using those DSMLs. Unlike the previous studies, the work herein covers an assessment of the proposed language to infer on its usability and help adoption of the language by clearly showing its generative performance and effect on reducing the time needed for MAS development.

6. CONCLUSION

A DSML, called DSML4BDI, for developing BDI agent systems has been introduced in this paper with including its graphical modeling and code generation features. A comparative evaluation was performed which revealed the efficiency of employing DSML4BDI. The quantitative analysis on generation performance showed that developers could achieve the significant amount of the artifacts required for exact BDI implementations automatically by just using the IDE of DSML4BDI. The evaluation was also beneficial to determine whether the use of DSML4BDI shortens MAS development time. Although the evaluators spent a little more time on design during MAS development by using DSML4BDI, implementation and test of the system were almost 5 times faster on average for the conducted case study. Taking into account the overall development process, the evaluation showed that the time needed for developing a BDI agent system from scratch can be reduced to one-third when DSML4BDI and its IDE are used.

Finally, the qualitative analysis part of the evaluation allowed us to receive feedbacks of agent developers on using DSML4BDI. Evaluators, who experienced using DSML4BDI, shared their opinions on DSML4BDI's convenience and appropriateness for agent development. They mostly confirmed that the language significantly facilitates the development of BDI agents. Such a qualitative evaluation

based on the user experiences also guided for eliminating some deficiencies (e.g. adding previously missing agent relations and plan attributes) and improving the language's IDE (e.g. reorganization of visual components for handy usage).

Although many efforts were given on AOSE field for providing new agent software architectures, methodologies, programming languages, platforms and tools, currently the adoption of software agents in the real system implementations for various industrial domains is not at the desired level. That most likely originates from the difficulties encountered on programming agents with general-purpose languages since such languages do not have built-in constructs, e.g. BDI elements, especially for goal-oriented intelligent agents. As previously indicated, languages and platforms like AgentSpeak and Jason are developed for this purpose. However, many agent developers hesitate on using such languages and platforms due to their high discrimination from well-known languages and developers do not prefer dealing with hardcoding the required logical and mathematical BDI constructs mandatory, e.g. in AgentSpeak and Jason. Hence, taking into consideration all features and benefits of DSML4BDI within this context, we strongly believe that DSML4BDI may also contribute to the widespread embracement of agent technologies by supporting easy and efficient implementation of BDI agents for the industry.

DSML4BDI, its modeling and code generation tools and example models are all available online with including required installation and configuration instructions at: <http://serlab.ube.ege.edu.tr/Bundles/dsml4bdi.zip>.

Our future work aims at extending the platform support of DSML4BDI by enabling MAS implementation in different agent platforms. For this purpose, interoperability between DSML4BDI and other existing MAS DSMLs can be established via horizontal model transformations.

7. ACKNOWLEDGEMENT

This work is funded by the Scientific and Technological Research Council of Turkey (TUBITAK) under grant 115E591.

8. REFERENCES

- [1] Rao, A.S., Georgeff, M.P.: "Decision procedures for BDI logics", *Journal of Logic and Computation*, 1998, 8(3):293-343
- [2] Bordini, R.H., Hubner, J.F., Wooldridge, M.: "Programming Multi-Agent Systems in AgentSpeak Using Jason" (John Wiley & Sons, 2007)
- [3] Teket, K.D., Sayit, M., Kardas, G.: "Software agents for peer-to-peer video streaming", *IET Software*, 2014, 8(4):184-192
- [4] Leito, P., Karnouskos, S.: "Industrial Agents: Emerging Applications of Software Agents in Industry" (Elsevier Science Publishers, 2015)
- [5] Tezel, B.T., Challenger, M., Kardas, G.: "A Metamodel for Jason BDI Agents". *Proc. 5th Symp. Languages, Applications and Technologies*, 2016, pp.8:1-8:9
- [6] "Eclipse Modeling Framework", <http://www.eclipse.org/modeling/emf/>, accessed March 2018
- [7] "Sirius Modeling Tool", <https://eclipse.org/sirius/>, accessed March 2018
- [8] "Acceleo Code Generator", <https://www.eclipse.org/acceleo/>, accessed March 2018
- [9] Challenger, M., Kardas, G., Tekinerdogan, B.: "A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems", *Software Quality Journal*, 2016, 24(3):755-795
- [10] Kardas, G., Gomez-Sanz, J.J. "Special issue on model-driven engineering of multi-agent systems in theory and practice", *Computer Languages, Systems & Structures*, 2017, 50:140-141
- [11] Kulesza, U., Garcia, A., Lucena, C., Alencar, P.: "A generative approach for multi-agent system development", *Lecture Notes in Computer Science*, 2005, 3390:52-69
- [12] Rougemaille, S., Migeon, F., Maurel, C., Gleizes, M-P. "Model Driven Engineering for Designing Adaptive Multi-agent Systems", *Lecture Notes in Artificial Intelligence*, 2007, 4995:318-333
- [13] Pavon, J., Gomez-Sanz, J.J., Fuentes, R.: "Model driven development of multi-agent systems". *Lecture Notes in Computer Science*, 2006, 4066:284-298
- [14] Hahn, C., Madrigal-Mora, C., Fischer, K.: "A Platform-Independent Metamodel for Multiagent Systems", *Autonomous Agents and Multi-Agent Systems*, 2009, 18(2):239-266
- [15] Omicini, A., Ricci, A., Viroli, M.: "Artifacts in the A&A meta-model for multi-agent systems", *Autonomous Agents and Multi-Agent Systems*, 2008, 17(3):432-456
- [16] Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J.J., Pavon, J., Gonzalez-Perez, C.: "FAML: A Generic Metamodel for MAS Development", *IEEE Transactions on Software Engineering*, 2009, 35(6):841-863
- [17] Garcia-Magarino, I.: "Towards the integration of the agent-oriented modeling diversity with a powertype-based language", *Computer Standards & Interfaces*, 2014, 36:941-952
- [18] Hahn, C.: "A Domain Specific Modeling Language for Multiagent Systems", *Proc. 7th Int. Conf. Autonomous Agents and Multi-Agent Systems*, 2008, pp.233-240
- [19] Gascuena, J.M., Navarro, E., Fernandez-Caballero, A. "Model-Driven Engineering Techniques for the Development of Multi-agent Systems", *Engineering Applications of Artificial Intelligence*, 2012, 25(1):159-173
- [20] Fuentes-Fernandez, R., Garcia-Magarino, L., Gomez-Rodriguez, A.M., Gonzalez-Moreno, J.C.: "A technique for defining agent-oriented engineering processes with tool support", *Engineering Applications of Artificial Intelligence*, 2010, 23(3):432-444
- [21] Ciobanu, G., Juravle, C.: "Flexible Software Architecture and Language for Mobile Agents", *Concurrency and Computation-Practice & Experience*, 2012, 24(6):559-571
- [22] Goncalves, E.J.T., Cortes, M.I., Campos, G.A.L., Lopes, Y.S., Freire, E.S.S., daSilva, V.T., deOliveira, K.S.F., deOliveira, M.A. "MAS-ML2.0: Supporting the modelling of multi-agent systems with different agent architectures", *Journal of Systems and Software*, 2015, 108:77-109

- [23] Challenger, M., Demirkol, S., Getir, S., Mernik, M., Kardas, G. and Kosar, T.: "On the use of a domain-specific modeling language in the development of multiagent systems", *Engineering Applications of Artificial Intelligence*, 2014, 28:111-141
- [24] Wautelet, Y., Kolp, M.: "Business and model-driven development of BDI multi-agent systems", *Neurocomputing*, 2016, 182:304-321
- [25] Cossentino, M., Chella, A., Lodato, C., Lopes, S., Ribino, P., Seidita, V.: "A notation for modeling Jason-like BDI agents". *Proc. Sixth Int. Conf. Complex, Intelligent and Software Intensive Systems*, 2012, pp.12-19

ACCEPTED MANUSCRIPT