Proceedings of the 19th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 637–642 DOI: 10.15439/2024F6157 ISSN 2300-5963 ACSIS, Vol. 39

# DSML4JaCaMo: A Modelling tool for Multi-agent Programming with JaCaMo

Burak Karaduman*, Baris Tekin Tezel*†, Geylani Kardas‡ and Moharram Challenger*
*Department of Computer Science, University of Antwerp and Flanders Make, Antwerp, Belgium
{burak.karaduman, baristekin.tezel, moharram.challenger}@uantwerpen.be
†Department of Computer Science, Dokuz Eylul University, Izmir, Türkiye
baris.tezel@deu.edu.tr
‡International Computer Institute, Ege University, Izmir, Türkiye
geylani.kardas@ege.edu.tr

***Abstract*—This paper introduces a domain-specific modelling language (DSML) called DSML4JaCaMo to develop belief-desire-intention (BDI) agents. The DSML's design covers aspects of Jason, Cartago, and Moise from viewpoints that follow the meta-modelling approach. In this way, the DSML4JaCaMo enables graphical modelling of JaCaMo's multi-agent systems (MASs), providing comprehensive support for defining agents' beliefs, desires, and intentions (BDI) using Jason, specifying artifacts and their operations with Cartago, and outlining organizational structures and norms via Moise. The DSML's operational semantics ensure seamless integration of these components, facilitating automatic code generation and artifact construction for creating a JaCaMo-based system. The graphical syntax contributes to ease of use, making it accessible for novice and experienced developers. This work aims to enhance the JaCaMo ecosystem by offering a model-driven approach to provide abstraction on MAS development as well as facilitating design and implementation.**

## I. Introduction

IN AGENT-ORIENTED software engineering (AOSE), higher-level abstractions, such as belief-desire-intention (BDI), are used more than object-oriented programming. In addition to agent programming, different dimensions are utilized along with agent development, namely artefacts and organisations [1]. As these concepts provide well-fit advantages on complex scenarios like cyber-physical systems (CPS), Internet-of-Things (IoT), and Industry 4.0 [2], they aid software complexity where it can be addressed using model-driven engineering (MDE) techniques [3], [4] which provides model-level abstraction for representing the system via model entities and relations to generate code and model-to-model transformation [5].

Despite numerous metamodels that exist to describe multi-agent systems (MASs) [6], [7], they are limited in covering agent, artefact, and organisational perspectives at once. Hence, this paper presents our ongoing work on platform-specific modelling of belief-desire-intention (BDI) Jason agents, including Cartago (Artefact) and Moise (Organisation) perspectives [1], [8]. We investigate the creation of a metamodel considering JaCaMo [1] and develop a graphical modelling tool that allows agent developers to model these views according to our proposed metamodel.

The rest of the paper is organized as follows. In section II, related studies are mentioned. Section III introduces the proposed metamodel JaCaMo. Section IV focuses on translational semantics for code generation. Section V includes case studies, model excerpts from the concrete syntax and evaluation. Lastly, section VI concludes the paper.

## II. Related Works

This study enhances the existing literature by introducing a model-driven engineering (MDE) approach for developing BDI agents and an environment and organization based on JaCaMo. To our knowledge, no previous research has specifically addressed the model-driven development of the JaCaMo-based MASs.

As the complexity of MASs increases, researchers in AOSE [9] strive to develop processes, methods, and techniques that enable system developers to address safety, interoperability, and performance effectively. Among these techniques, software modelling and MDE [10], [11], [12], [13], [14] are prominently utilized. MDE approaches allow developers to work at a higher level of abstraction and use component modelling early in the development process, which helps mitigate the complexities associated with MAS implementation [15], [16], [17], [18], [19], [20].

In MDE-based development processes, models are treated as first-class entities [21]. Engineers create models using various modelling languages that represent distinct parts of the system, providing a high level of abstraction. This abstraction enables engineers to concentrate on defining the system's functionality rather than its implementation details. AOSE researchers have defined several agent metamodels [6], [7] to support modelling various aspects of MAS at appropriate abstraction levels. These metamodels are designed to capture agent characteristics such as plans, beliefs, goals, and interactions within MAS organizations.

To effectively implement MDE for MASs, a practical approach involves customizing Domain-Specific Modeling Languages (DSMLs) using integrated development environments (IDEs) that support modelling and code generation for the target system. Proposed MAS DSMLs (e.g., [22], [23], [24], [25],

[26], [27]) are based on the aforementioned agent metamodels and offer various abstract syntaxes. These DSMLs facilitate modelling both static and dynamic aspects of agent software from different perspectives within the MAS domain, including internal agent behaviour, interactions with other agents, and environmental entities.

Our research has focused on developing a platform-dependent modelling language that supports MASs, environment, and organization. We have developed a meta-model for JaCaMo that also leads to creating a syntax of a DSML for MAS development. To demonstrate its effectiveness, we have carried out qualitative evaluations.

## III. The Syntax of the language

This section introduces the metamodel of DSML4JaCaMo, representing the domain-specific language's abstract syntax. Generally, a metamodel outlines the system elements, their relationships, and cardinality constraints. It may also include attributes and operations for these elements. The metamodel is implemented using the EMF Ecore framework. Figure 1 illustrates the DSML4JaCaMo metamodel, describing the key meta-elements briefly. Detailed information regarding the JaCaMo components represented by the meta-elements can be reached by [1].

Beginning with Agent perspective, *Agent* defines the Jason BDI agent, which has *Plan*, *Belief*, *Rule*, *Goal*. A *Plan* element consists of *BodyTerm* and *Action*. As a *Plan* consists of a set of actions, a self-reference, namely *nextAction*, is created to model the action chain. An action can be an *InternalAction* and *ExternalAction*. *Message* element refers to agent communication. Lastly, *TriggeringEvent* meta-element is included as events are realized as consequences of beliefs or goals change in any Jason agent's mind.

In the Artifact dimension, a *Workspace* defines all contained artefacts and AbstractOperation E-Class, namely *AbsOperation* that the agents can utilize. *Port* element is used to link two artifacts within artifacts dimension. Since an *Artifact* element can have multiple *ObsProperty* elements that can be used in the operations such *LinkedOperation*, *InternalOperation*, *GuardOperation* and *Operation*.

On the Organisational perspective side, the Moise platform uses *XML* configuration comprising normative, structural and functional specifications. *NormativeSpecification* contains *Norm* elements which normalise the *Mission* elements. The *StructuralSpecification* has *Group* and *Role* EClasses in which subgroups and role extensions can be created. Moreover, *FormationConstaints* defines the features such as formation between roles and group scope. *Link* determines the link and their types among the roles. Lastly, *FunctionalSpecification* composes *Scheme* and *Scheme* creates *Mission* elements where each mission contains goals that agents achieve. In addition to *Scheme*, we also defined *OPlan* and *OGoal* EClasses, which allows us to model the layered goal and plan composition structure of Moise, which uses XML configuration. This composable layered structure via *OGoal* and *OPlan* elements are visualized in detail in section V.

The concrete syntax of a language encompasses the set of notations responsible for its presentation and construction. In the context of DSML specifications, the concrete syntax primarily facilitates mapping between meta-elements and their representations within instance models of the meta-model. Consequently, we developed a graphical concrete syntax that aligns the abstract syntax elements of DSML4BDI with their corresponding graphical notations. To achieve this, we leveraged the features of the Sirius [1] modelling environment. Sirius provides tools for creating a graphical editor from an Ecore metamodel and allows for the definition of specialized editors—including diagrams, tables, and trees—based on a viewpoint approach. This functionality enabled us to build the DSML4JaCaMo graphical modelling toolset within the Sirius environment for this study. Due to page limitations, we cannot provide the concrete system syntax as a table separately. However, you will see some examples in the case study graphs.

## IV. Code Generation: Translational Semantics

A comprehensive definition of a DSML cannot be achieved solely by specifying the notations and their representations. It also necessitates providing the semantics of language concepts, typically regarding the meanings of already established concepts. In this study, therefore, the metamodel elements are mapped to the concepts within the JaCaMo framework. This mapping between the metamodel and JaCaMo entities facilitates a series of model-to-text (M2T) transformations, constructing the DSML4JaCaMo's semantics within the JaCaMo framework. To this end, model-to-code transformation is used. Some excerpts of the generation rules are given and discussed below.

Listing 1: Excerpt from Acceleo rules for creating JaCaMo files

```
[template public generateElement(aMAS : MAS) ]
[comment @main/]
[for (ag : Agent | aMAS.agent)]
[ file (ag.Name.concat('. asl ' ), false , 'UTF−8')]
    ................
[for (p : Plan | ag.plan)]
[ if (p. asBeliefAddition =' true ' )]+[ else ]+![/ if ][p.Name/]
    <− [p.hasContext.Expression /]
[ if (p.hasBody. firstAction −>size()>0)]
[ p.hasAction.thePlanSeq(p.hasAction, p.hasBody.
    firstAction )−>asOrderedSet().Expression /]
[/ if ]
[/ for ]
[/ file ]
[/ for ]
[for (wp : Workspace | aMAS.workspace)]
[for ( art : Artifact | wp. artifact )]
[ file ( art .className.concat( ' . java ' ), false , 'UTF−8')]
import cartago .∗;
public class [ art .className/] extends Artifact {
    ................
[for (Op: AbsOperation | art . operation )]
[ if (Op.eClass() .instanceTypeName.equalsIgnoreCase('
    dSML4JaCaMo.Operation'))]
```

[1] ‘Sirius modeling tool’. Available at https://eclipse.org/sirius/, accessed May 2024

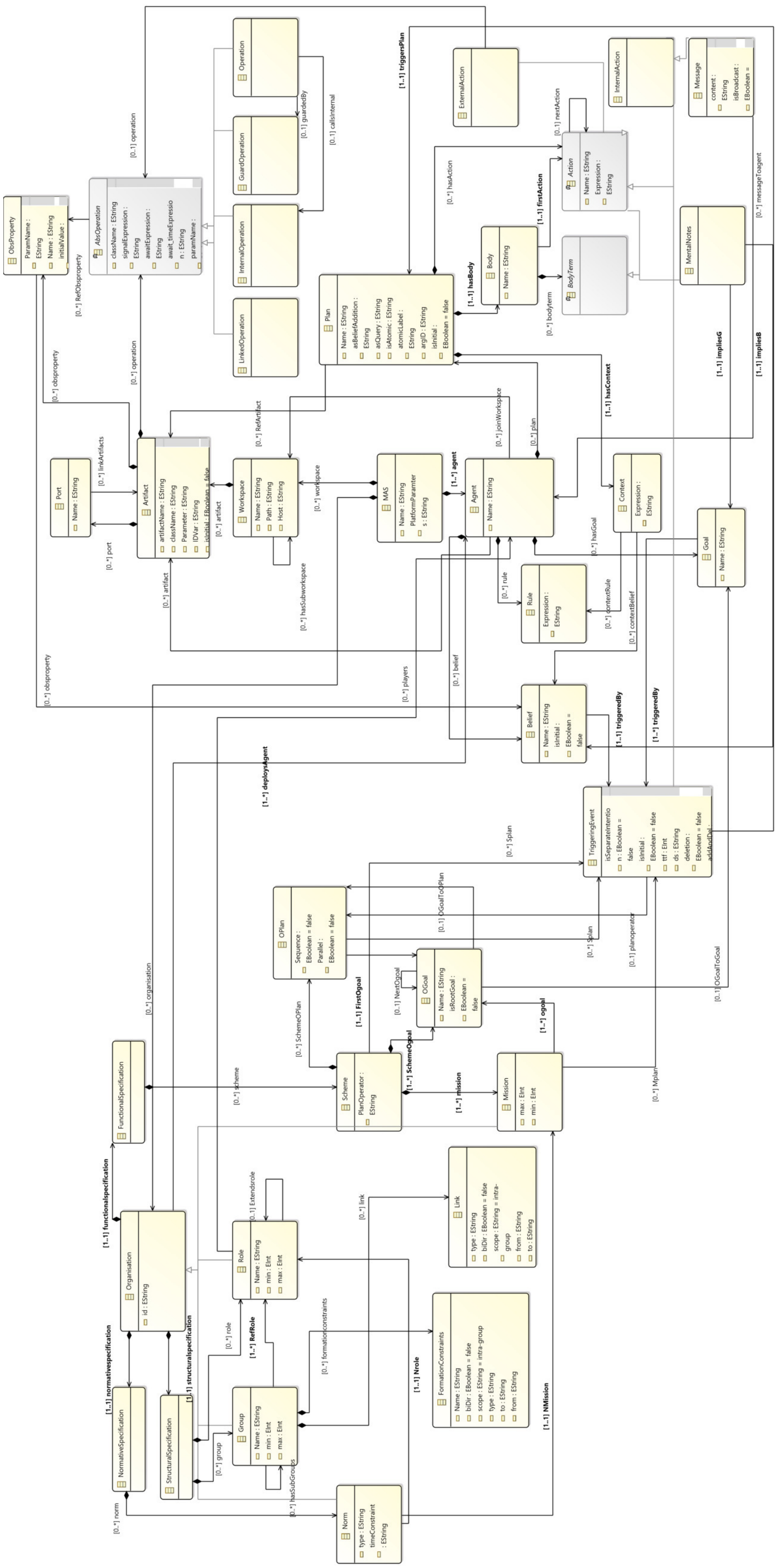

Fig. 1: DSML4JaCaMO Meta-model.

```
22 @OPERATION void [Op.className/]
23 {
24 ObsProperty prop = getObsProperty(" ");
25 prop.updateValue( );
26 signal("tick");
27 }
28 [/ if ]
29     ................
30 [/ if ]
31 [/ for ]
32 }
33 [/ file ]
34 [/ for ]
35 [/ for ]
36 [for (org : Organisation | aMAS.organisation)]
37 [ file (org.id.concat('.xml'), false, 'UTF−8')]
38 <?xml version="1.0" encoding="UTF−8"?>
39 <?xml−stylesheet href="http :// moise.sourceforge.net/xml/
       os.xsl" type="text/xsl" ?>
40 < organisational − specification id="[org.id/]"
41 os−version="0.8" xmlns='http :// moise.sourceforge.net/os'
42           xmlns:xsi='http :// www.w3.org/2001/XMLSchema−
                instance'
43           xsi:schemaLocation='http :// moise.sourceforge.net/
                os
44                http :// moise.sourceforge.net/xml/os.xsd'>
45 [for (ss: StructuralSpecification | org.
        structuralspecification )]
46 < structural − specification >
47     ................
48 </ structural − specification >
49 [/ for ]
50 < functional − specification >
51 [for (fs: FunctionalSpecification | org.
        functionalspecification )]
52 [for (sch: Scheme| fs.scheme)]
53 <scheme id="[sch.id/]">
54 [for (Og: OGoal | sch.SchemeOgoal ) ]
55 [ if (Og.isRootGoal)]
56 <goal id="[Og.Name/]" ttf="5 seconds">
57 [for (Op: OPlan | Og.OGoalToOPlan )]
58 <plan operator="[ if (Op.Parallel )] parallel [ else ]sequence
       [/ if ]">
59 [for (FOg: OGoal | Op.FirstOgoal )]
60 <goal id="[FOg.Name/]" ds="">
61     ................
62 </goal>
63 [/ if ]
64     ................
65 [/ for ]
66 [/ for ]
67 </ functional − specification >
68 [for (ns: NormativeSpecification | org.
        normativespecification )]
69 [for (norm: Norm | ns.norm )]
70 <normative− specification >
71     ................
72 </normative− specification >
73 [/ for ]
74 [/ for ]
75 [/ file ]
76 [/ for ]
77 [/ template ]
```

Within the JaCaMo framework, each agent is represented by an ASL file, which includes code written in the AgentSpeak language, designed to outline the internal structure of the agent according to the BDI architecture. Fundamentally, an AgentSpeak agent is characterized by a set of beliefs, rules, and plans. Beliefs denote the initial knowledge possessed by the agent. Rules are logical expressions or mathematical equations that guide the agent's reasoning. Plans consist of the actions and/or subgoals the agent employs to achieve its current objectives. Each plan within an AgentSpeak agent comprises a triggering event, a context, and a body element. The triggering event indicates the circumstances under which the plan is appropriate. The context determines the plan's applicability based on the agent's beliefs. The body is a sequence of basic actions and/or subgoals the agent will execute.

Listing 1 includes an excerpt from the Acceleo rules to generate all Agent ASL, Artifact Java and Organisation XML files. Initially, in lines 3 to 16, an ASL file is created for each agent. Lines 6 to 11 are responsible for generating each agent's plans, plans' contexts and the actions in these plans, including their triggering events and goals.

Lines 14 to 35 are responsible for writing the necessary Java code for the artefacts of each workspace in the environment. Each Artefact Java file includes Cartago library and Artefact naming, which extends the Artifact class. The generator then synthesis operations that can be *Operation*, *Linked Operation*, *Guard Operation* and *Internal Operation*. The generator checks the EClass of these elements to generate the necessary type. Lastly, observable properties are also generated within the corresponding operation types.

The remaining lines, defined between 36 and 76, are responsible for creating the organization file, including the agents' roles in the system, their responsibilities, and the organizational goals and plans, aforementioned as *OGoal* and *OPlan* elements. The code generation for organisational dimension also considers specifications which are *Functional*, *Structural* and *Normative*. Specifically, an excerpt from *StructuralSpecification* is defined lines between 45-49, and *FunctionalSpecification* is defined between 50 and 67, and lines 68-77 scopes the *NormativeSpecification*. As *StructuralSpecification* creates goal composition and mapping relations such as *OGoalToOPlan* and *FirstOGoal* is designed to preserve this structure at the modelling level as well. This structure is exemplified in Figure 2. Due to size and space limitations, a brief description of the transformation rules is mentioned.

## V. Case studies & Evaluation

Two case studies are planned and implemented with DSML4JaCaMo to validate the system and assess its code-generating capability. The case study models are displayed.

Figure 4 shows how the DSML builds the code artifacts based on the intended models and how the delta codes are incorporated to create the final code.

Our modelling language is evaluated by examining its code-generation capabilities. This involves comparing the lines of generated code with the final code, which includes the additional delta code.

### A. *Case 1: Writing Paper*

In this case study, a set of agents coordinate to mimic the process of writing a paper. This is achieved with the help of assistant agents following a structured plan and having different roles and missions. There are roles such as writer, editor and manager. The writing goal is divided into multiple goals achieved via agents' plans. In addition, an agent uses a checkList artifact to mark the finalisation of paper writing. Figure 2 depicts the organisational structure, i.e., the scheme of the case study.

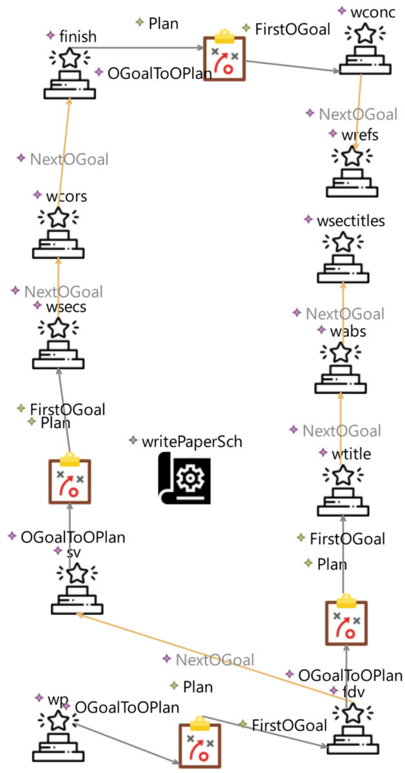


Fig. 2: Scheme design based on OGoals and OPlans composition.

### B. *Case 2: Harvest Process for Pizza*

In this case study, agents coordinate to mimic the pizza process, from wheat harvesting to dough preparation to having a pizza. The pizza-making goal is broken down into multiple goals. In addition, an agent uses an oven artifact to mimic the cooking process based on/off control. Table I shows the code generation performance for Jason, Cartago and Moise platforms based on lines of code. In each scenario, 3 agents for Jason, 1 Scheme for Moise and 1 Artifact for Cartago were used. Figure 3 illustrates these three agents connected to a Cartago Workspace and joined the organisation, namely *Org 1*. It uses a scheme called *harvestForPizza*, where three norms and three roles are defined. In addition, Figure 4 shows the Agent viewpoint that contains the Plans and their corresponding contexts.

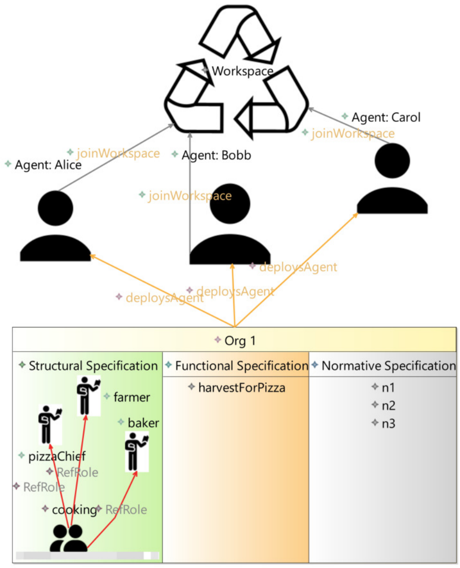


Fig. 3: The MAS viewpoint of the harvest case study.

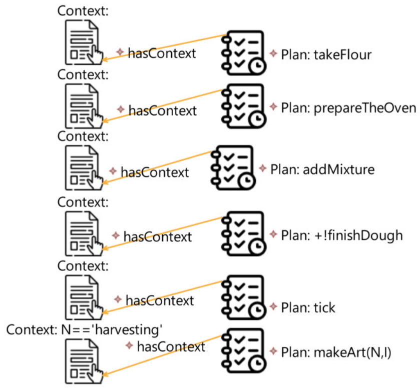


Fig. 4: Agent viewpoint of the harvest case study.

Eventually, based on model-to-text transformation, we assessed the code generation performance using two case studies. Although this facilitates the system's development, another MDE approach, the model-to-model transformation, is also required to verify the JaCaMo. Specifically, the designed Jason agents and Moise organisation need static analyses before the system's deployment. In this regard, Coloured Petri-nets (CPN) are a suitable paradigm to transform the JaCaMo model to CPN representations to achieve analyses [5], [28] using a non-deterministic domain. Another approach is that Moise's

TABLE I: Evaluation results for two case studies based on lines of code.

| | Agent 1 | Agent 2 | Agent 3 | Agent Total | 1 Scheme | 1 Artifact |
|---|---|---|---|---|---|---|
| Case1 Normal | 6 | 35 | 3 | 42 | 77 | 11 |
| Case 1 Generated | 6 | 28 | 3 | 37 | 60 | 9 |
| Case 2 Normal | 12 | 38 | 3 | 53 | 74 | 23 |
| Case 2 Generated | 12 | 24 | 3 | 39 | 44 | 21 |

Scheme design needs a proper formalism for representation. For this purpose, Statecharts can be used as there is quite a similarity between deterministic OGoal and OPlan structures. In the next section the paper is concluded.

## VI. Conclusion

This study presents a DSML called DSML4JaCaMo, detailing its abstract and concrete syntaxes with translational semantics. The DSML is evaluated through two case studies, showing that 76% of the total system code is generated automatically. Our study offers an abstraction to simplify complexity and alleviate difficulties. We use a set of graphical notations and domain constraints to create a graphical editor for the DSML. The effectiveness of the DSML's generation capability is assessed through two case studies. In future work, we aim to provide a more comprehensive evaluation of the language and its tool by following the approach for the multi-case systematic evaluation of MAS DSMLS introduced in [19]. That evaluation will provide the quantitative measurement of the language's usability as well as the assessment of the language features according to a series of well-defined quality characteristics for MAS development.

## References


[1] O. Boissier, R. H. Bordini, J. Hubner, and A. Ricci, *Multi-agent oriented programming: programming multi-agent systems using JaCaMo*. Mit Press, 2020. ISBN 9780262044578

[2] P. Leitao, S. Karnouskos, L. Ribeiro, J. Lee, T. Strasser, and A. W. Colombo, "Smart agents in industrial cyber–physical systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1086–1101, 2016. doi: 10.1109/JPROC.2016.2521931

[3] B. Karaduman, I. David, and M. Challenger, "Modeling the engineering process of an agent-based production system: An exemplar study," in *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2021. doi: 10.1109/MODELS-C53483.2021.00051 pp. 296–305.

[4] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis lectures on software engineering*, vol. 3, no. 1, 2017. doi: https://doi.org/10.1007/978-3-031-02549-5

[5] B. Karaduman, B. T. Tezel, and M. Challenger, "Towards static analysis of bdi agents on cps using petri nets and model-driven engineering," in *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer (in press), 2024.

[6] C. Hahn, C. Madrigal-Mora, and K. Fischer, "A platform-independent metamodel for multiagent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 18, no. 2, pp. 239–266, 2009.

[7] B. T. Tezel, M. Challenger, and G. Kardas, "A metamodel for Jason BDI agents," in *5th Symposium on Languages, Applications and Technologies (SLATE'16)*, vol. 51, 2016. doi: 10.4230/OASIcs.SLATE.2016.8 pp. 8:1—-8:9.

[8] O. Boissier, R. H. Bordini, J. F. Hübner, and A. Ricci, "Dimensions in programming multi-agent systems," *The Knowledge Engineering Review*, vol. 34, p. e2, 2019. doi: 10.1017/S026988891800005X

[9] O. Shehory and A. Sturm, *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*. Springer-Verlag Berlin Heidelberg, 2014.

[10] B. Lelandais, M.-P. Oudot, and B. Combemale, "Applying model-driven engineering to high-performance computing: Experience report, lessons learned, and remaining challenges," *Journal of Computer Languages*, vol. 55, p. 100919, 2019. doi: https://doi.org/10.1016/j.cola.2019.100919

[11] A. Bucchiarone, J. Cabot, R. F. Paige, and A. Pierantonio, "Grand challenges in model-driven engineering: an analysis of the state of the research," *Software and Systems Modeling*, vol. 19, no. 1, pp. 5–13, 2020. doi: https://doi.org/10.1007/s10270-019-00773-6

[12] C. Verbruggen and M. Snoeck, "Model-driven engineering: A state of affairs and research agenda," *Enterprise, business-process and information systems modeling*, 2021. doi: 10.1007/978-3-030-79186-5_22

[13] E. de Araújo Silva, E. Valentin, J. R. H. Carvalho, and R. da Silva Barreto, "A survey of model driven engineering in robotics," *Journal of Computer Languages*, vol. 62, 2021. doi: https://doi.org/10.1016/j.cola.2020.101021

[14] D. Di Ruscio, D. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, and M. Wimmer, "Low-code development and model-driven engineering: Two sides of the same coin?" *Software and Systems Modeling*, vol. 21, no. 2, 2022. doi: https://doi.org/10.1007/s10270-021-00970-2

[15] Y. E. Cakmaz, O. F. Alaca, C. Durmaz, B. Akdal, B. Tezel, M. Challenger, and G. Kardas, "Engineering a bdi agent-based semantic e-barter system," in *2017 International Conference on Computer Science and Engineering (UBMK)*. IEEE, 2017. doi: 10.1109/UBMK.2017.8093474 pp. 1072–1077.

[16] B. T. Tezel, M. Challenger, and G. Kardas, "Dsml4bdi: A modeling tool for bdi agent development," in *12th turkish national software engineering symposium (uyms 2018)*, 2018, pp. 1–8.

[17] V. Mascardi, D. Weyns, A. Ricci, C. B. Earle, A. Casals, M. Challenger, A. Chopra, A. Ciortea, L. A. Dennis, Á. F. Díaz *et al.*, "Engineering multi-agent systems: State of affairs and the road ahead," *ACM SIGSOFT Software Engineering Notes*, vol. 44, no. 1, pp. 18–28, 2019. doi: https://doi.org/10.1145/3310013.3322175

[18] M. Challenger, B. T. Tezel, V. Amaral, M. Goulao, and G. Kardas, "Agent-based cyber-physical system development with sea_ml++," in *Multi-Paradigm Modelling Approaches for Cyber-Physical Systems*. Elsevier, 2021, pp. 195–219.

[19] O. F. Alaca, B. T. Tezel, M. Challenger, M. Goulão, V. Amaral, and G. Kardas, "Agentdsm-eval: A framework for the evaluation of domain-specific modeling languages for multi-agent systems," *Computer Standards & Interfaces*, vol. 76, p. 103513, 2021. doi: https://doi.org/10.1016/j.csi.2021.103513

[20] B. Karaduman, B. T. Tezel, and M. Challenger, "Rational software agents with the bdi reasoning model for cyber–physical systems," *Engineering Applications of Artificial Intelligence*, vol. 123, p. 106478, 2023. doi: https://doi.org/10.1016/j.engappai.2023.106478

[21] G. Kardas, F. Ciccozzi, and L. Iovino, "Introduction to the special issue on methods, tools and languages for model-driven engineering and low-code development," *Journal of Computer Languages*, vol. 74, 2023. doi: https://doi.org/10.1016/j.cola.2022.101190

[22] 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," *Engineering Applications of Artificial Intelligence*, vol. 28, pp. 111–141, 2014. doi: https://doi.org/10.1016/j.engappai.2013.11.012

[23] E. J. T. Gonçalves, M. I. Cortés, G. A. L. Campos, Y. S. Lopes, E. S. Freire, V. T. da Silva, K. S. F. de Oliveira, and M. A. de Oliveira, "Masml 2.0: Supporting the modelling of multi-agent systems with different agent architectures," *Journal of Systems and Software*, vol. 108, pp. 77–109, 2015. doi: https://doi.org/10.1016/j.jss.2015.06.008

[24] F. Bergenti, E. Iotti, S. Monica, and A. Poggi, "Agent-oriented model-driven development for JADE with the JADEL programming language," *Computer Languages, Systems & Structures*, vol. 50, pp. 142–158, 2017. doi: 10.1016/j.cl.2017.06.001

[25] G. Kardas, B. T. Tezel, and M. Challenger, "Domain-specific modelling language for belief–desire–intention software agents," *IET Software*, vol. 12, no. 4, pp. 356–364, 2018. doi: https://doi.org/10.1049/iet-sen.2017.0094

[26] D. Sredojević, M. Vidaković, and M. Ivanović, "Alas: agent-oriented domain-specific language for the development of intelligent distributed non-axiomatic reasoning agents," *Enterprise Information Systems*, vol. 12, no. 8-9, pp. 1058–1082, 2018. doi: https://doi.org/10.1080/17517575.2018.1482567

[27] A. Siabdelhadi, A. Chadli, H. Cherroun, A. Ouared, and H. Sahraoui, "Motrans-bdi: Leveraging the beliefs-desires-intentions agent architecture for collaborative model transformation by example," *Journal of Computer Languages*, vol. 74, p. 101174, 2023. doi: https://doi.org/10.1016/j.cola.2022.101174

[28] B. Karaduman, M. Challenger, R. Eslampanah, J. Denil, and H. Vangheluwe, "Analyzing WSN-based IoT Systems using MDE Techniques and Petri-net Models," in *4th International Workshop on Model-Driven Engineering for the Internet-of-Things (MDE4IoT), Co-Located With Software Technologies: Applications and Foundations (STAF 2020), Virtual Event, Norway*, 2020, pp. 35–46.