

AN ONLINE MODELING LANGUAGE FOR THE LOW-CODE DEVELOPMENT OF BELIEF-DESIRE-INTENTION (BDI) AGENTS

First Author: Burak Çelik

International Computer Institute, Ege University
Izmir, Türkiye
bcburakcelik@gmail.com

Second Author: Geylani Kardas

International Computer Institute, Ege University
Izmir, Türkiye
geylani.kardas@ege.edu.tr

Third Author: Baris Tekin Tezel

Department of Computer Science, Dokuz Eylul University
Izmir, Türkiye
baris.tezel@deu.edu.tr

ABSTRACT: With their autonomy capabilities, agent-based systems have gained prominence in various industrial domains. Belief-Desire-Intention (BDI) architecture, a cornerstone of Agent-Oriented Software Engineering (AOSE), is commonly used to create proactive agents that interact effectively with their environment. Various domain-specific modeling languages (DSMLs) exist for the model-driven development of multi-agent systems (MAS) with a BDI approach. However, they are mostly dependent on the desktop libraries hindering widespread accessibility and online development. In this paper, we introduce a modeling language, called SAML, which enables the creation of BDI agents online. Our goal is to eliminate certain dependencies to the local computing resources and paving the way for a development environment that offers the low-code platform support in conjunction with a Software as a Service (SAAS) infrastructure. Additionally, the platform of SAML, also created in this study, serves as a platform-independent online modeling and development environment. The main objective is to promote the model-driven development of BDI agents using the low-code development techniques, one of the current trends in the software engineering field. By eliminating local dependencies, SAML can become a widely adopted tool for BDI agent development, fostering innovation and agility across various MAS applications. The paper presents a structured approach to achieve these goals, including research on the foundations of SAML, development of user interfaces for upper-level modeling, model-to-model transformation with JSON, and code transformation using JASON standards.

Keywords: AOSE, Domain-specific modeling languages (DSML), Low-code development, Model-driven development, Multi-agent systems (MAS).

1. INTRODUCTION

In the era of Industry 4.0, agent-based systems have emerged as a fundamental paradigm for achieving autonomy and adaptability in various domains. These systems, comprising multi-agent systems (MAS) (Weiss, 2009) with distinct functionalities, are instrumental in scenarios such as logistics, smart grids, and digital transformation initiatives (Leitao and Karnouskos, 2015). One of the pivotal architectural paradigms in Agent-Oriented Software Engineering (AOSE) is the Belief-Desire-Intention (BDI) model (Rao and Georgeff, 1998). This model plays a crucial role in enabling agents to work proactively and interact effectively with their environment, making it a cornerstone of MAS development.

Various domain-specific languages (DSLs) / domain-specific modeling languages (DSMLs) (e.g. Hahn, 2008; Kardas et al., 2010; Ciobanu and Juravle, 2012; Demirkol et al., 2012; Challenger et al., 2014; Goncalves et al., 2015; Bergenti et al., 2017; Kardas et al., 2018; Sredejovic et al., 2018; HoseinDoost et al., 2019) exist for the model-driven development (MDD) of multi-agent systems (MAS) (Alaca et al., 2021) by considering approaches or models such as BDI. However, they are mostly dependent on the desktop libraries hindering widespread accessibility and online development. Hence, in this study, we investigate the applicability of Software as a Service (SaaS) and cloud computing features to promote the usability of MAS DSMLs with online integrated development environments (IDEs). Within this context, we introduce a DSML, called SAML, which enables the creation of BDI agents online. SAML offers capabilities to model BDI agents compatible with the Jason platform (Bordini et al., 2007). SAML, introduced in this paper, is mainly an improved version of its predecessor (Tezel, 2020) having a limited MAS modeling capabilities by relying only on local (desktop) libraries and platforms, such as Eclipse and Java. This new version of SAML enhances accessibility and usability of the language in an online development context. We also believe that the language introduced in this paper helps eliminate the challenges in terms of version compatibility and operating system support. It also supports MAS development with the application of one of the newest and popular programming paradigms, namely low-code development (Cabot, 2020; Sahay et al., 2020; Di Ruscio et al., 2022).

This research paper aims to address the above mentioned limitations and enhance SAML by providing an online platform-independent modeling and development environment. We focus on implementing a new independent low-code development platform (LCDP) to make BDI agent development easier and more accessible. Hence, SAML can become a widely adopted tool for BDI agent development, fostering innovation and agility across various MAS applications.

MAS models designed with SAML can be automatically transformed inside SAML's IDE and source code of the designed system can be generated which is compatible with the Jason platform, a Java-based interpreter for an extended version of a Prolog-like logic programming language, called AgentSpeak (Rao, 1996). Developers can also create JSON (Ecma-404, 2017) based instance MAS models according to the model specifications.

The rest of the paper is organized as follows: Section 2 discusses the metamodel from which the abstract syntax of SAML is derived. SAML's graphical concrete syntax, enabling MAS modeling, is given in Section 3. Section 4 covers the implementation of the online IDE supporting SAML. Section 5 presents a case study to exemplify how SAML and its IDE can be used for MAS development. Section 6 concludes the paper.

2. ABSTRACT SYNTAX

An abstract syntax can be defined as the conceptual representation of the fundamental building blocks and relationships of a modeling language free from the concrete notations. This abstract structure focuses on specifying the language's main entities, their core features, concepts, and relationships. Mostly, an

abstract syntax is defined with a metamodel and is used to describe the meaning and structure of a DSL / DSML (Kardas et al., 2023).

As shown in Figure 1, our metamodel includes SAML meta-entities and their relationship between each other. In addition to the agent internals, the metamodel also includes environmental entities of a MAS other than the autonomous agents. In the following, main entities of the metamodel (written in Italics) are described.

Agents represent autonomous entities in the MAS, capable of sensing their environment, reasoning about their beliefs and desires, and making decisions to achieve their intentions. In a SAML model, agents embody the BDI paradigm, where they have beliefs about the world, desires to achieve certain goals, and intentions that drive their actions.

Environments define the context in which agents operate. They represent the surroundings or the system in which agents interact and perform actions. Environments influence agents' perceptions and may trigger events or conditions that impact their beliefs and influence decision-making.

Capabilities characterize the skills, functionalities, or abilities of agents. They describe what an agent is capable of doing within the system. Capabilities contribute to the agent's set of potential actions and influence the planning and execution of tasks.

Subcapabilities has also same skills with the capabilities. However they are always extended the agent capability by linking the main capability which is connected with the agent.

Plans outline sequences of actions that agents can take to achieve specific goals or desires. They represent strategic courses of action. Plans are part of the intentional aspect, helping agents realize their desires by providing a structured approach to achieving goals.

Beliefs capture the knowledge or information that agents hold about the state of the world, influencing their understanding of the environment. They form the basis for an agent's understanding of its environment. Beliefs drive an agent's decision-making process by influencing the perception of the environment and shaping desires and intentions.

Operations define the actions or functions that an agent can perform within the system. They represent executable behaviours within the system. Operations are instrumental in the implementation of plans and the execution of actions to achieve desired outcomes.

Events represent occurrences or changes in the environment that may trigger actions or affect an agent's beliefs and desires. Events are significant in the perceptual aspect, signaling changes in the environment that agents need to respond to or consider in their decision-making process.

Actions are the executable behaviours that agents can perform in response to their beliefs, desires, and intentions, representing the tangible outcomes of an agent's decision-making process.

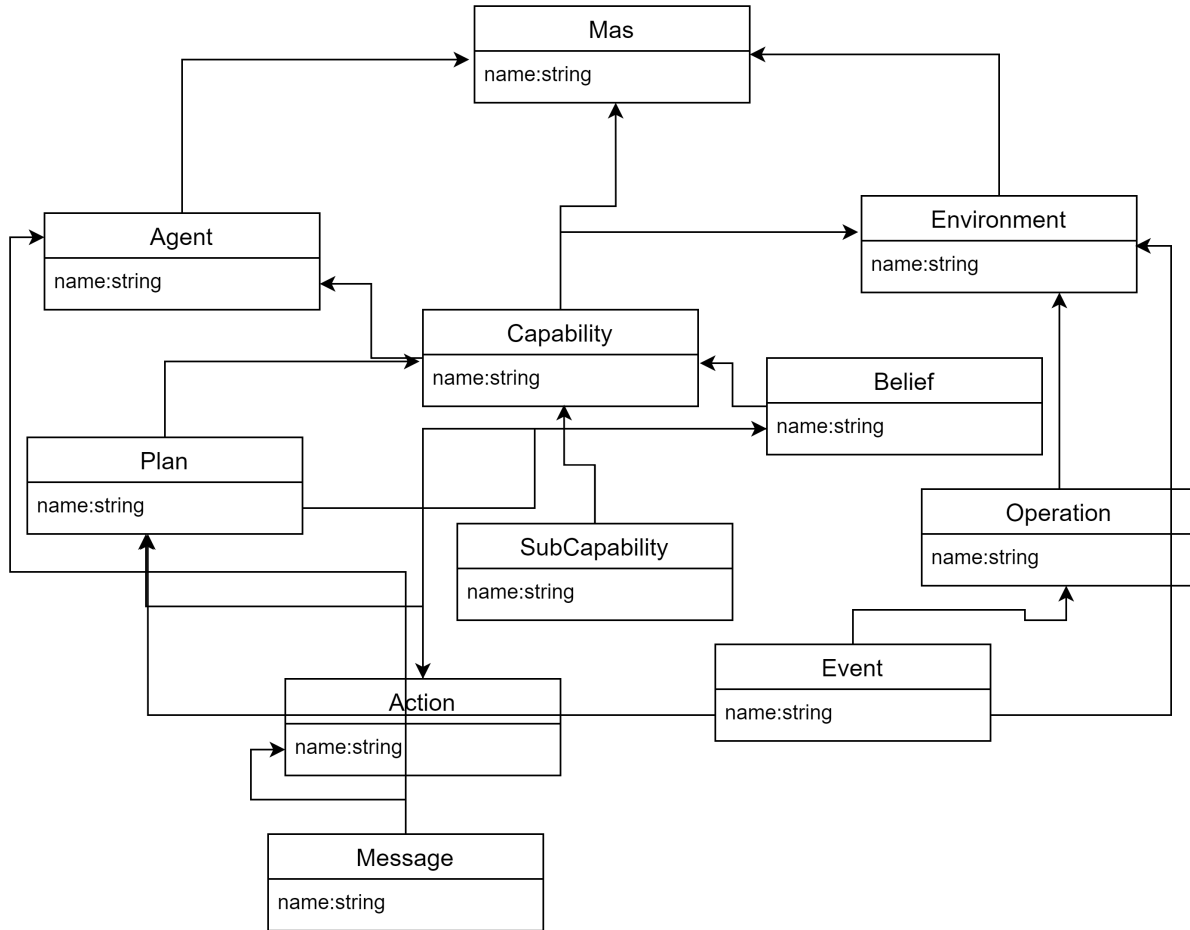


Figure 1. SAML metamodel

3. CONCRETE SYNTAXES


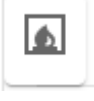








While the delineation of abstract syntax within a metamodel encompasses the fundamental language constructs manifested in the language and the connections among these constructs, the definition of a concrete syntax establishes a connection between meta-elements and their depictions in the instance models. Essentially, concrete syntax constitutes the assortment of symbols and structures that simplifies the exhibition and creation of the language. Therefore we provided a graphical concrete syntax which maps SAML's abstract syntax elements presented in the previous section to their graphical notations

In the process of translating the abstract syntax of SAML into a concrete syntax, we employ the graphical notations outlined in Table 1. As mentioned earlier, these notations enable us to create definitions without any dependencies, facilitating seamless modeling and then code transformations.

The graphical representations allow us to navigate from the conceptual realm of the abstract metamodel to the practical use of a concrete syntax. By adopting these notations, we achieve the transformation of our defined structures into both models and code, emphasizing a versatile and dependency-free approach.

This transition from abstract to concrete, facilitated by the graphical notations, underscores the flexibility and adaptability of our MDD methodology. Following this methodology, a developer may use these graphical notations to formalize the system models of a MAS to be implemented.

Table 1. SAML concepts and their corresponding notations for the graphical concrete syntax.

Concept	Notation	Concept	Notation
Agent		Belief	
Capability		Plan	
Environment		Relation	
Event		Action	
Operation		Message	

The concrete syntax of the language consists of four different diagrams representing four different perspectives. These are MAS, Environment, Capability, and Plan diagrams. The MAS diagram, representing the overall structure of the system, is responsible for modeling the agents, their capabilities, environments, and the interactions between them. The Environment, Capability, and Plan diagrams are respectively responsible for modeling the internal structure of the environment, capabilities, and plans. We will delve into a more detailed examination of these diagrams in Section 5.

4. AN ONLINE IDE FOR MODELING AND IMPLEMENTING BDI AGENTS

The web-based model development environment aims to facilitate model-to-model and code transformations which is compatible with JASON syntax. Our platform not only features robust support for Model-to-Model (M2M) and Model-to-Code (M2C) conversions but also offers a sophisticated Integrated Development Environment (IDE) for model development. This comprehensive suite of capabilities underscores our commitment to facilitating seamless transitions between different modelling levels while simultaneously providing a conducive environment for the efficient creation and enhancement of models.

SAML IDE is currently available at <https://web-dsml.vercel.app/designer>. Access to the IDE requires users to have authorization for the storage of metamodel data, enabling online accessibility from anywhere. In this phase of the study, we leverage the Google authentication feature to empower users to store their data securely and manage authorizations effectively.

The main interface of the application (see Figure 2) includes a left panel containing a "projects" directory, allowing the creation of multiple projects. On the right panel, users find a workspace equipped with components such as "node" and "edge" for modelling. Additionally, the same panel features a button for

converting the design to a JSON model, java-based code conversion, and an area for modifying the selected "node" name.

In the project directory located on the left side of the application, each folder can be created to correspond to the desired new project name. Each file created within the folder directory corresponds to the conceptual definitions for modeling.

Table 2 lists the file names and extensions of four different diagrams represented which we have already discussed in the previous section. It is crucial to note that each concept should be created with the respective file extensions.

Table 2. Required file extensions for the related concepts.

Concept	File Extension
MAS	.mas
Environment	.env
Capability	.cap
Plan	.pln

As we illustrated in Figure 2, each of these file types created within the project folder generates a layout area conducive to designing in the middle section of the IDE. It should be emphasized that the main directories of the created files correspond to a project name, and each project name populates the "Projects" dropdown component on the right side of the IDE.

Within the layout area, to perform modeling, any node icon from the "nodes" section on the right side of the application can be dragged and dropped into the layout area. To connect each node, after selecting the desired edge icon from the "edge" section on the right, a connection can be made between nodes using mouse control. After modeling is completed for each concept, the "save" button initiates the process of storing the models in a dedicated database for developers. Developers can make a selection from the project selection area on the right side of the application and by using the buttons, both code and model transformations will be downloaded as an archived file to the local computer. If the generated model data poses a challenge to code and model transformation, a warning will be issued before the transformation is carried out.

5. CASE STUDY

We consider low-code development of a Garbage Collector MAS which is widely utilized in the field of Agent-Oriented Software Engineering (AOSE) for exemplifying MAS design and implementation. The system enables agents to collectively handle garbage in an environment and consists of two types of agents, Burner and Collector. The primary task of Burner agents is to share the location of waste with Collector agents. Another task of Burner agents is to dispose of the garbage brought by Collector agents. Collector agents, after getting information about the location of waste from Burner agents, go to that location, collect the garbage, and bring it back to the Burner agents. After delivering it, Collector agents must wait for messages from Burner agents about potential new garbage in the environment.

Developers can create a design model by dragging and dropping necessary elements from the palette on the right side of the modeling environment, as explained in the previous section. In this way, the main entities of the system, such as agents, capabilities, and environments, are identified in the MAS diagram.

An example SAML instance model for this case study is shown in Figure 2. As depicted in the figure, the Garbage Collector MAS model includes three capabilities, one of which is a sub-capability of another, an environment, and two agents. Each nodes can be displayed with type of the node, icon and name, also each edge provides information on how the nodes are linked with each other.

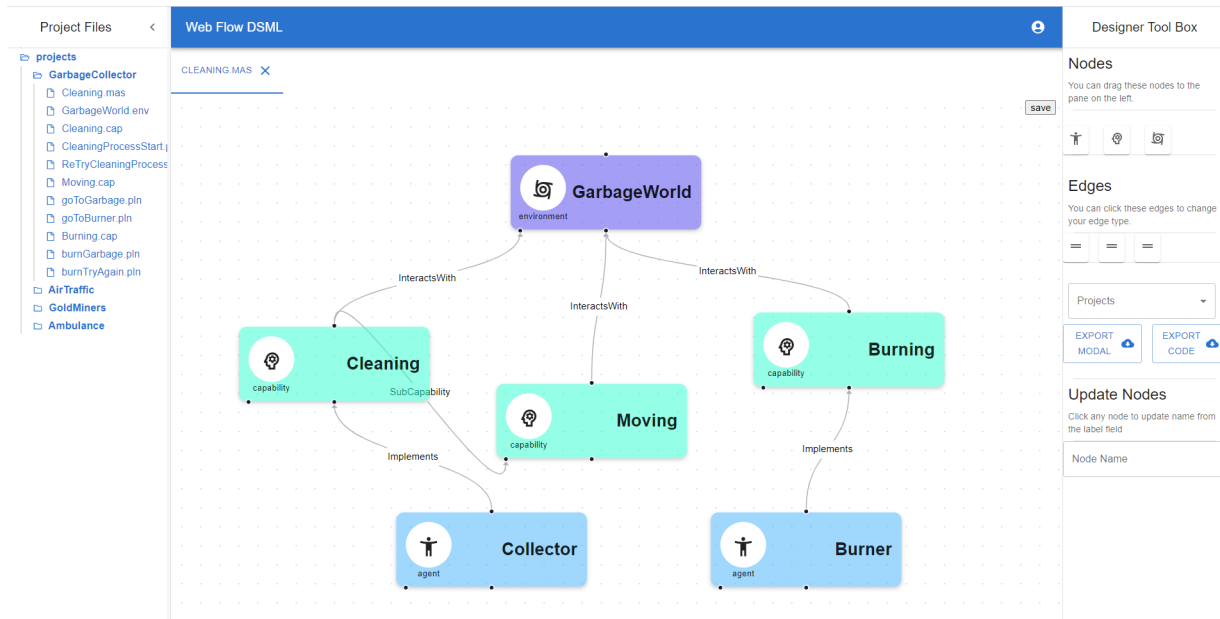


Figure 2. MAS model for Garbage Collector agents

As previously introduced, SAML has four different diagram types for modeling BDI agents and their environments. The instance models of these diagrams for this case study are defined with files under the "GarbageCollector" project directory, similar to the MAS diagram model. Conforming to the file extensions for the corresponding concepts discussed in Section 2, it is worth noting that in the MAS diagram, the model names we create are the same as the file names (e.g., the Cleaning capability has the file name Cleaning.cap for the modeling design file).

Figure 3 and Figure 4 provide short snippets of Java and ASL files which are automatically generated from the instance SAML model and suitable for further JASON programming. It is essential to note that the classes herein are entirely dependent on the file and node names provided within the application.


```
public class GarbageWorld {  
  
    public boolean retryTrash() {  
        //TODO: should be implemented for the retryTrash operation.  
        return true;  
    }  
    public boolean goToGarbage() {  
        //TODO: should be implemented for the goToGarbage operation.  
        return true;  
    }  
    public boolean goToBurner() {  
        //TODO: should be implemented for the goToBurner operation.  
        return true;  
    }  
    public boolean startBurn() {  
        //TODO: should be implemented for the startBurn operation.  
        return true;  
    }  
}
```

Figure 3. An excerpt from the generated GarbageWorld.java class

In Figure 3, we can observe that the class name GarbageWorld.java corresponds to the environment diagram. It is worth noting that the helper methods created within this Java class support the event designs generated during modeling. The concrete classes we will observe in Figure 4 are inherently supportive of the JASON programming language, as mentioned earlier. The concrete files corresponding to each agent model are obtained through the beliefs, rules, and plans data created during modeling.

```
// Agent Collector  
/* Initial beliefs */  
available().  
  
/* Initial rules */  
  
/* Initial goals */  
  
/* Plans */  
@CleaningProcessStart  
+available;trashOccured  
<-goToGarbage,pickGarbage,goToBurner,dropTheGarbage,startBurn  
@ReTryCleaningProcess  
+trashOccured  
<-retry
```

Figure 4. An excerpt from the generated Collector.asl file

For the M2M conversion, our platform provides a robust JSON-based model transformation for M2M conversion, aligning with contemporary usage standards.

```
▼ {nodes: [{width: 76, height: 80, id: "dndnode_0.915490435821867", type: "customNode",...},...],...}
  ▼ edges: [{source: "dndnode_3.483940437708766", sourceHandle: "a", target: "dndnode_1.4839404377087662",...},...]
    ▼ 0: {source: "dndnode_3.483940437708766", sourceHandle: "a", target: "dndnode_1.4839404377087662",...}
      deletable: true
      id: "reactflow__edge-dndnode_3.483940437708766a-dndnode_1.4839404377087662"
      label: "InteractsWith"
      ▶ markerStart: {type: "arrowclosed"}
        source: "dndnode_3.483940437708766"
        sourceHandle: "a"
        target: "dndnode_1.4839404377087662"
        targetHandle: null
        type: "floating"
      ▶ 1: {source: "dndnode_3.483940437708766", sourceHandle: "b", target: "dndnode_4.483940437708766",...}
      ▶ 2: {source: "dndnode_3.483940437708766", sourceHandle: "b", target: "dndnode_1.3645829854726073",...}
      ▶ 3: {source: "dndnode_1.4839404377087662", sourceHandle: "b", target: "dndnode_4.483940437708766",...}
      ▶ 4: {source: "dndnode_4.483940437708766", sourceHandle: "b", target: "dndnode_0.915490435821867",...}
      ▶ 5: {source: "dndnode_1.3645829854726073", sourceHandle: "b", target: "dndnode_0.36458298547260726",...}
    ▼ nodes: [{width: 76, height: 80, id: "dndnode_0.915490435821867", type: "customNode",...},...]
      ▼ 0: {width: 76, height: 80, id: "dndnode_0.915490435821867", type: "customNode",...}
        ▶ data: {label: "Collector", name: "agent", colorCode: "#33B2FF"}
          dragging: false
          height: 80
          id: "dndnode_0.915490435821867"
          ▶ position: {x: -165, y: 315}
          ▶ positionAbsolute: {x: -165, y: 315}
          selected: false
          type: "customNode"
          width: 76
        ▶ 1: {width: 64, height: 80, id: "dndnode_1.4839404377087662", type: "customNode",...}
        ▶ 2: {width: 118, height: 80, id: "dndnode_3.483940437708766", type: "customNode",...}
        ▶ 3: {width: 76, height: 80, id: "dndnode_4.483940437708766", type: "customNode",...}
        ▶ 4: {width: 61, height: 80, id: "dndnode_0.36458298547260726", type: "customNode",...}
        ▶ 5: {width: 68, height: 80, id: "dndnode_1.3645829854726073", type: "customNode",...}
      ▶ viewport: {x: 927.5, y: 116.5, zoom: 2}
```

Figure 5. Json model transformation for MAS model

Figure 5 showcases the distinctions between nodes and edges, illustrating how each item's connections can be established with relevant properties in a MAS model transformation. While this demonstrates only MAS M2M transformation, we provide output as an archived file by performing separate M2M transformations for each diagram.

6. CONCLUSION

A novel DSML, called SAML, has been introduced in this paper. SAML enables online design and implementation of BDI agents without any dependency to the local / desktop software libraries. Modeling both agent internals and the environment of MAS are possible with the graphical concrete syntax of SAML. Instance models conforming to the SAML metamodel can be created within SAML IDE and code from these models for the implementation can be automatically achieved again online. To the best of our knowledge, SAML represents the first DSML to provide low-code development of agents and MAS.

In our future work, we aim at conducting experiments on the assessment of both SAML and its IDE. For this purpose, we plan to utilize evaluation frameworks such as (Challenger et al., 2016; Alaca et al., 2021)

which support the quantitative and qualitative assessment of MAS DSMLs and their artifacts according to the various criteria. The results and feedback gained from these multi-case and multi-user evaluations may lead the improvement of SAML's usability as well as the quality of the produced models and agent programs.

REFERENCES

- Alaca, O. F., Tezel, B. T., Challenger, M., Goulao, M., Amaral, V. and Kardas, G. (2021) "AgentDSM-Eval: A framework for the evaluation of domain-specific modeling languages for multi-agent systems", *Computer Standards & Interfaces*, vol. 76, 103513, pp. 1-20, DOI: 10.1016/j.csi.2021.103513
- Bergenti, F., Iotti, E., Monica, S., Poggi, A. (2017) "Agent-oriented modeldriven development for JADE with the JADEL programming language". *Computer Languages, Systems & Structures*, 50: 142-158
- Bordini, R.H., Hubner, J.F., Wooldridge, M. (2007) "Programming Multi-Agent Systems in AgentSpeak Using Jason", John Wiley & Sons
- Cabot J. (2020) "Positioning of the low-code movement within the field of model-driven engineering", In proc. the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems
- Challenger, M., Demirkol, S., Getir, S., Mernik, M., Kardas, G., Kosar, T. (2014) "On the use of a domain-specific modeling language in the development of multiagent systems". *Engineering Applications of Artificial Intelligence*, 28: 111-141.
- Challenger, M., Kardas, G. and Tekinerdogan, B. (2016) "A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems", *Software Quality Journal* 24(3): 755-795
- Ciobanu, G., Juravle, C. (2012) "Flexible Software Architecture and Language for Mobile Agents". *Concurrency and Computation-Practice & Experience*, 24(6): 559-571.
- Demirkol, S., Challenger, M., Getir, S., Kosar, T., Kardas, G. and Mernik, M. (2012) "SEA_L: A Domain-specific Language for Semantic Web enabled Multi-agent Systems", In proc. 2nd Workshop on Model Driven Approaches in System Development (MDASD 2012), held in conjunction with 2012 Federated Conference on Computer Science and Information Systems (FedCSIS 2012), September 9-12, 2012, Wroclaw, Poland, IEEE Conference Publications, pp. 1373-1380
- Di Ruscio, D., Kolovos, D., Lara, J., Pierantonio, A., Tisi, M., Wimmer, M. (2022) "Low-code development and model-driven engineering: Two sides of the same coin?". *Software and Systems Modeling* 21: 437-446
- Ecma-404(2017) https://www.ecma-international.org/wp-content/uploads/ECMA404_2nd_edition_december_2017.pdf (Last access: November 23, 2023)
- Goncalves, E. J. T., Cortes, M. I., Campos, G. A. L., Lopes, Y. S., Freire, E. S. S., da Silva, V. T., de Oliveira, K. S. F., de Oliveira, M. A. (2015) "MAS-ML2.0: Supporting the modelling of multi-agent systems with different agent architectures". *Journal of Systems and Software* 108: 77-109.
- Hahn, C. (2008) "A Domain Specific Language for Multiagent Systems". In proc. 7th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008), Estoril, Portugal, pp. 233-240.
- Hoseindoost, S., Adamzadeh, T., Zamani, B., Fatemi, A. (2017) "A modeldriven framework for developing multi agent systems in emergency response environments". *Software and Systems Modeling* 18(3): 1985-2012.

- Kardas, G., Tezel, B. T., Challenger, M. (2018) "Domain-specific modelling language for belief-desire-intention software agents". IET Software 12(4): 356-364.
- Kardas, G., Demirezen, Z. and Challenger, M. (2010) "Towards a DSML for Semantic Web enabled Multi-agent Systems", In proc. International Workshop on Formalization of Modeling Languages (FML 2010), held in conjunction with the 24th European Conference on Object-Oriented Programming (ECOOP 2010), June 21-25, 2010, Maribor, Slovenia, ACM Press, pp. 1-5
- Kardas, G., Ciccozzi, F. and Iovino, L. (2023) "Introduction to the special issue on methods, tools and languages for model-driven engineering and low-code development", Journal of Computer Languages 74, 101190, pp. 1-2.
- Leitao, P., Karnouskos, S. (2015) "Industrial Agents: Emerging Applications of Software Agents in Industry". Elsevier Science Publishers, Amsterdam, Netherlands, 476 pages.
- Rao, A. (1996) "AgentSpeak(L): BDI agents speak out in a logical computable language", Lecture Notes in Computer Science 1038: 42-55
- Rao, A., Georgeff, M.P. (1998) "Decision procedures for BDI logics". Journal of Logic and Computation 8(3): 293-343
- Sahay, A., Indamutsa, A., Di Ruscio D., Pierantonio, A. (2020) Supporting the understanding and comparison of low-code development platforms. In proc. 46th Euromicro Conference on Software Engineering and Advanced Applications, pp. 171-178
- Sredejovic, D., Vidakovic, M., Ivanovic, M. (2018) "ALAS: agent-oriented domain-specific language for the development of intelligent distributed non-axiomatic reasoning agents". Enterprise Information Systems, 12 (8-9): 1058-1082.
- Tezel, B. (2020) "Debugging for the domain-specific agent modeling languages of software agents". Phd. Thesis, Ege University, Turkiye
- Weiss, G. (2009) "Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence", The MIT Press, 643 pages