AgentDSM Eval : a framework for the evaluation of domain specific modeling languages for multi agent systems

AgentDSM-Eval: A Framework for the Evaluation of Domain-specific Modeling Languages for Multi-agent Systems

Omer Faruk Alaca[1], Baris Tekin Tezel[1,2], Moharram Challenger[1,3], Miguel Goulão[4], Vasco Amaral[4], Geylani Kardas[1]

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

[2]Department of Computer Science, Dokuz Eylul University, Izmir, Turkey

[3]Department of Computer Science, University of Antwerp and Flanders Make, Belgium

[4]Universidade NOVA de Lisboa, NOVA LINCS, DI, FCT, Lisboa, Portugal

omerfarukalaca@gmail.com, baris.tezel@deu.edu.tr, moharram.challenger@uantwerpen.be, {vma, mgoul}@fct.unl.pt, geylani.kardas@ege.edu.tr

Abstract

Software development required for constructing multi-agent systems (MAS) usually becomes challenging and time-consuming due to the properties of autonomy, distributedness, and openness of these systems in addition to the complicated nature of internal agent behaviors and agent interactions. To facilitate MAS development, the researchers propose various domain-specific modeling languages (DSMLs) by enriching MAS metamodels with a defined syntax and semantics. Although the descriptions of these languages are given in the related studies with the examples of their use, unfortunately, many are not evaluated in terms of either the usability (being hard to learn, understand and use) or the quality of the generated artifacts. Hence, in this paper, we introduce an evaluation framework, called AgentDSM-Eval, with its supporting tool which can be used to evaluate MAS DSMLs systematically according to various quantitative and qualitative aspects of agent software development. The empirical evaluation, presented by the AgentDSM-Eval framework, was successfully applied for one of the well-known MAS DSMLs. The assessment showed that both MAS domain coverage of DSMLs and the agent developers' adoption of modeling elements can be determined with this framework. Moreover, the tool's quantitative results can assess MAS DSML's performance on the development time and throughput. AgentDSM-Eval also enables the qualitative assessment of MAS DSML features according to novel quality characteristics and measures, which it defines specifically for the MAS domain.

Keywords: Multi-agent system, Domain-specific modeling language, Agent-oriented software engineering, Evaluation framework, Human factors in Modelling, Usability Evaluation, AgentDSM-Eval

1. Introduction

In a Multi-agent System (MAS), autonomous software agents interact with each other competitively or collaboratively to perform tasks and execute plans through a set of reactive and/or proactive behaviors (Weiss, 2016). MAS are recognized as both useful abstraction and effective technologies for modeling and building autonomous, complex, and distributed systems in various industrial fields, e.g., industrial automation, real-time adaptive resource management, large-scale network and service management, integrating quality and process control in production lines, fast deployment of evolvable systems and predictive analysis for business management (Leitao and Karnouskos, 2015; Liang et al., 2019). However, the development of software required for constructing MAS usually becomes challenging and time-consuming due to the properties autonomy, distributedness, and openness of these systems, in addition to the complicated nature of internal agent behaviors and agent interactions (Challenger et al., 2016a; Mascardi et al., 2019).

To minimize the abovementioned problems of MAS development, researchers in agent-oriented software engineering (AOSE) field (Shehory and Sturm, 2014) define various agent metamodels (e.g., Bernon et al., 2005; Omicini et al., 2008; Beydoun et al., 2009; Hahn et al., 2009; Challenger et al., 2011; Garcia-Magarino, 2014; Tezel et al., 2016), which include fundamental MAS entities and relations. Furthermore, model-driven agent development approaches (Kardas, 2013) are provided, and the researchers propose using domain-specific languages (DSLs) / domain-specific modeling languages (DSMLs) (Mernik et al., 2005; Kelly and Tolvanen, 2008; Kosar et al., 2016) to facilitate design and implementation of software agents by enriching MAS metamodels with some defined syntax and semantics.

In AOSE, perhaps the most popular way of applying model-driven engineering (MDE) techniques for MAS development, is based on creating DSMLs with appropriate integrated development environments (IDEs) in which both modeling and code generation for MAS-to-be-developed can be adequately performed (Kardas and Gomez-Sanz, 2017). Proposed MAS DSMLs (e.g., Hahn, 2008; Ciobanu and Juravle, 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) usually support modeling both the static and the dynamic aspects of agent software from different MAS viewpoints including agent internal behavior model, interaction with other agents, use of resources and other environmental entities. Although the descriptions of these languages are given in these studies mostly including some examples of how they can be utilized during MAS development, unfortunately, many do not consider an evaluation of the proposed language, i.e., evaluating the usability of the language and the efficiency of the generated artifacts.

Usability plays an essential role in the adoption of DSLs (Barišić et al., 2018). If some kind of systematic evaluation for the usability of MAS DSMLs can be provided, this may lead the agent developers to infer on whether a MAS DSML is suitable for the needs of agent design and implementation, and so, it facilitates the MAS development. Quality assessment of the MDE processes (Goulao et al., 2016) brought by using the MAS DSMLs compared to the conventional MAS development approaches can also be possible with these evaluations. Moreover, throughput performance (e.g., the generalization of agent components) of the language and saving on the development time can be analyzed if an evaluation produces some quantitative results. DSML developers, who implement these languages, may also benefit from such assessment to improve their languages, e.g., according to the developers' feedback. Hence, in this paper, we introduce an evaluation framework, called AgentDSM-Eval, with its supporting tool which can be used to evaluate MAS DSMLs according to various quantitative and qualitative aspects of agent software development. The main contributions of AgentDSM-Eval can be listed as follows:

* A general framework that can be applied for the systematic evaluation of any MAS DSML.

* Semi-automatic comparison of DSML metamodels with a reference MAS metamodel to quantitatively determine the DSML's coverage on MAS domain concepts.

* Qualitative and quantitative evaluation of MAS DSMLs and their features with computer-aided automation.

* Support on both conducting multi-case empirical evaluation of MAS DSMLs and online analysis of the results achieved.

* Automatic analysis of designed models to infer which meta-entities and/or viewpoints of a MAS DSML are mostly adopted by the agent developers.

* Qualitative assessment of MAS DSMLs according to novel quality characteristics and measures specific to the MAS domain leading to comparable results.

The multi-case empirical evaluation, presented by the AgentDSM-Eval framework, was successfully applied for one of the well-known MAS DSMLs. This evaluation's results are given in this paper as well as the discussion about the features of AgentDSM-Eval and its tool. The use of AgentDSM-Eval's qualitative characteristics in the comparative evaluation of different MAS DSMLs is also exemplified.

The paper's remainder is organized as follows: the AgentDSM-Eval framework and its features are discussed in Section 2. Section 3 introduces the online tool that supports MAS DSML evaluations and analyzes the results according to AgentDSM-Eval specifications. Section 4 both demonstrates how AgentDSM-Eval can be used to assess a MAS DSML and discusses the achieved results. Section 5 includes the related work on MAS DSMLs and their evaluation. Finally, we conclude the paper with Section 6.

2. Evaluation Framework

The AgentDSM-Eval framework enables both the evaluation and the comparison of MAS DSMLs by considering the various features such as ease of use, MAS domain coverage, richness and efficiency of the supported toolsets, and finally, productivity on generating agent software components. To provide both quantitative analysis and qualitative evaluation of MAS DSMLs, the framework adopts the multi-case study approach which we first introduced in (Challenger et al., 2016b) for evaluating a MAS development language, called SEA_ML (Challenger et al., 2014; Challenger et al., 2018). According to this approach, using a DSML is evaluated within the scope of many use cases, each covering the design and implementation of agent systems for different business domains with varying complexities. As we will discuss shortly, AgentDSM-Eval improves this approach by adding new metrics, specially to facilitate the qualitative assessment of language features on MAS development. This improvement comes in the sequence of our previous experiences (Kardas et al., 2017; Kardas et al., 2018) that showed the qualitative assessment brought in (Challenger et al., 2016b), which is composed of only answering three open-ended questions, limits the appropriate procurement of evaluator feedback. Moreover, AgentDSM-Eval introduces the comparison of language syntaxes quantitatively using a reference MAS metamodel, which is also not supported in the previous work (Challenger et al., 2016b).

Both MAS DSML developers and users may benefit from the proposed framework. Figure 2.1. portrays how this tool-assisted framework can be used during the evaluation of MAS DSMLs within the scope of various quantitative and qualitative aspects. In the following subsections, we first describe the phases of the multi-case study protocol applied during the evaluation of MAS DSMLs. Then, we will discuss both the execution of the AgentDSM-Eval processes (shown in Figure 2.1) and the language features' utilization on a quantitative and qualitative evaluation of MAS DSMLs and their artifacts.

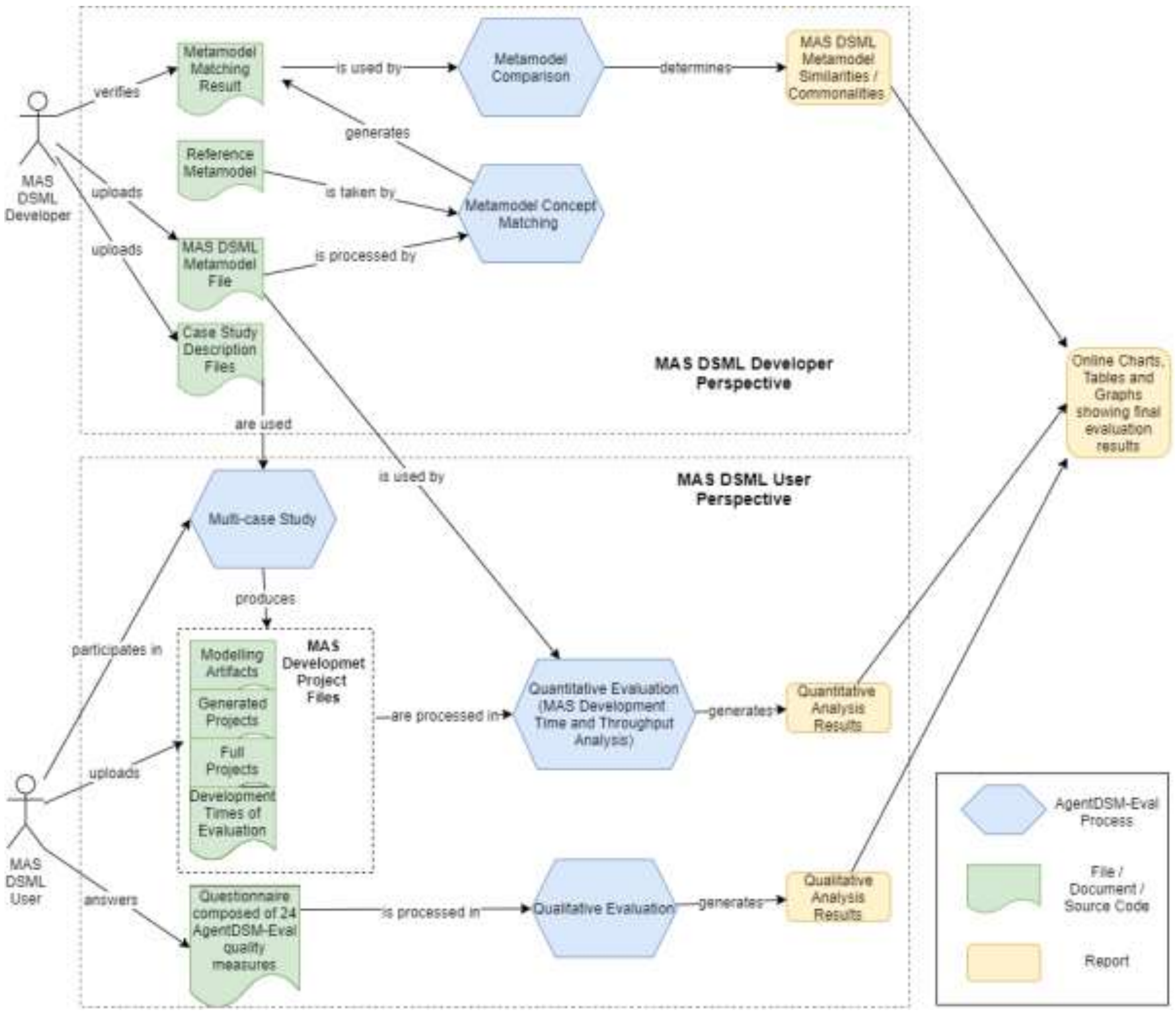

MAS DSML Developer
verifies
uploads
uploads
Metamodel Matching Result
is used by
Metamodel Comparison
determines
MAS DSML Metamodel Similarities / Commonalities
generates
Reference Metamodel
is taken by
Metamodel Concept Matching
MAS DSML Metamodel File
is processed by
Case Study Description Files
MAS DSML Developer Perspective
Online Charts, Tables and Graphs showing final evaluation results
are used
is used by
MAS DSML User Perspective
Multi-case Study
produces
participates in
Modelling Artifacts
Generated Projects
Full Projects
Development Times of Evaluation
MAS Developmet Project Files
are processed in
Quantitative Evaluation (MAS Development Time and Throughput Analysis)
generates
Quantitative Analysis Results
uploads
MAS DSML User
answers
Questionnaire composed of 24 AgentDSM-Eval quality measures
is processed in
Qualitative Evaluation
generates
Qualitative Analysis Results
AgentDSM-Eval Process
File / Document / Source Code
Report

Figure 2.1: Use of AgentDSM-Eval framework for MAS DSML evaluation.

2.1 The Structure of the Multi-case Study

The Multi-case Study in AgentDSM-Eval is conducted in accordance with a protocol consisting of three phases, namely preparation, execution, and analysis.

Each of the case study scenarios, consisting of various agents' design and implementation, is described in the preparation phase. The operational features of each MAS DSML to be evaluated are also collected via these case studies. Besides, the MAS developers who actively participate in the language assessment studies should also be determined in this phase. If needed, the training on the MAS DSMLs, which will be evaluated, is given to this team to keep the level of knowledge on using these MAS DSMLs the same required for the whole team.

During the execution phase, an opening briefing is first presented to introduce both the case study scenarios and the online tool for AgentDSM-Eval. Then, the evaluator teams, formed by the agent software developers, design and implement the MAS that meets the related case scenario's requirements. After finishing each MAS development, all developers are requested to input data for the analysis by utilizing the AgentDSM-Eval web tool.

Finally, the analysis of the evaluation data is performed. This phase is automated inside the AgentDSM-Eval tool (will be discussed in the next section) and does not need any human intervention. Upon completion of data input, the web application automatically analyzes these data and reports the quantitative and qualitative results pertaining to the evaluation of the MAS DSML in question.

2.2 Quantitative Evaluation

The quantitative evaluation process inside AgentDSM-Eval consists of 3 main sections. First, the abstract syntax of the MAS DSML, which is being evaluated, is compared with a reference MAS metamodel to determine the comprehensiveness level of the language on agent concepts. Then, the analysis of the development costs (using the elapsed time for MAS development) and development outputs (artifacts) in a comparative way.

2.2.1 Comparison with a Reference Metamodel

Almost all MAS DSML studies (e.g., Hahn 2008, Gascuena 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) provide definitions of agent components including agent internals, plans, roles, goals, agent behavior models, and the relationship between agents within a MAS. These definitions are mostly formalized with a metamodel description leading to various language syntaxes for these DSMLs. However, very few of these studies discuss how the proposed metamodels support MAS domain concepts. Moreover, they only show the derived syntax utilization for a minimal number of MAS development examples. The AgentDSM-Eval framework can measure each MAS DSML's comprehensiveness on agent domain concepts and their relations by comparing these languages' metamodels with a reference MAS metamodel representing an all-embracing model of agent components. In addition to determining these metmodels' domain coverage, the related analysis also enables the MAS developers to compare MAS DSMLs. Thus, after each DSML's metamodel is evaluated according to this reference model, it is possible to provide quantitative information to MAS developers to infer which MAS DSML provides certain agent concepts and / or relationships more extensively. For instance, two MAS DSMLs can be compared through their support on Belief-Desire-Intention (BDI) agent models (Rao and Georgeff, 1998) when these metamodels are first compared with the reference metamodel. They receive a compatibility degree, and finally, these degrees are compared with each other to give an idea on BDI coverage of these languages.

Many agent metamodel proposals (e.g., Bernon et al., 2005; Omicini et al., 2008; Beydoun et al., 2009; Hahn et al., 2009; Challenger et al., 2011; Garcia-Magarino, 2014; Tezel et al., 2016) exist in AOSE to model agent internals, communication between the agents and interaction of agents within MAS environments. However, it is difficult to use most of these metamodels for modeling wide-ranging agent systems since they include the description of agent meta-entities only for a specific MAS platform and/or an AOSE methodology such as Gaia (Zambonelli et al., 2003), INGENIAS (Pavon et al., 2005), Prometheus (Padgham and Winikoff, 2005) and Tropos (Bresciani et al., 2004). Some of these metamodels lack support on modeling runtime components critical to implement agent plans and agent interactions. Among all of these existing MAS metamodels, the FAML metamodel (Beydoun et al., 2009) was chosen as the reference metamodel, and utilized inside the AgentDSM-Eval framework. This metamodel is derived from the synthesis of the MAS concepts introduced with many relevant AOSE methodologies. It provides an inclusive definition of all fundamental agent concepts and their relations from both static and dynamic MAS modeling aspects.

The FAML metamodel consists of 2 layers, namely Design Time and Runtime. Definitions of entities such as agents, organizations, resources, interaction protocols, environment statements, tasks, goals, and message schemas are given in the design-time layer, while entities like beliefs, roles, plans, events, message actions are defined in the runtime layer. Also, each of these layers has two scopes, agent-external, and agent-internal. The FAML metamodel presents four different views to group classes into these four other areas of concern. Twenty-six concepts for design time and twenty-one concepts for runtime are defined in this metamodel including the relations between them.

In AgentDSM-Eval, FAML concepts are matched with the abstract syntax of a MAS DSML, which is being evaluated. To realize this matching, meta-entities defined inside the related MAS DSML's metamodel should be determined. This process is manual inside the online AgentDSM-Eval tool when a serialized version (e.g., in XMI, Ecore, or JSON) of the MAS DSML's metamodel is not available. However, it becomes semi-automatic when the serialized version of the DSML metamodel is available. The tool generates the definition of the meta-entities from the metamodel files and prepares them to match FAML concepts. If meta-entities named such as Agent, Belief, Goal, Plan are determined during the automatic generation of the definitions from the MAS metamodel, these entities are also automatically matched with the corresponding FAML concepts as the result of a keyword-based search. Other remaining meta-entities need to be checked manually. In all cases, the matching results need to be verified by a user who plays administrator roles in the AgentDSM-Eval tool before these results are used in the conducted multi-case studies. It is worth indicating that the MAS developers participating in the evaluation of MAS DSMLs do not need to concern with this matching process, as it is isolated from them. Only users of the AgentDSM-Eval online tool with administrator rights deal with this matching process and make the results of comparison with FAML ready for all remaining evaluations. Moreover, this comparison with FAML is required to be made just once for a specific MAS DSML, which means the administrators do not need to repeat this process as long as the MAS DSML does not change.

Upon completing the semi-automatic matching between FAML concepts and a MAS DSML metamodel, the similarities / commonalities between the FAML and the related DSML are determined automatically by the AgentDSM-Eval tool. Hence, the overall metamodel comparison is semi-automatic in which the matching process is manual in some situations (as discussed above), while the similarity / commonality determination is always performed automatically by the tool. After the comparison is completed, each MAS DSML receives a quantitative comparison degree. These degrees can be used to evaluate different MAS DSMLs and help compare these DSMLs with each other.

2.2.2 Case Study Analysis

The case study analysis in the AgentDSM-Eval framework provides the assessment of results from conducting a series of MAS development studies (each named as a case study) in which a specific MAS DSML is utilized during the design and implementation of agent systems with varying complexities and business domains. The time elapsed for developing a MAS (Development Time) and the quantity of the produced outputs (Development Throughput) are considered during the case study analysis.

Development Time refers to the time an evaluator group spends for the problem analysis, modeling / design, development, and testing steps for a specified case study. As the number of evaluators (agent developers) increases, these measured times provide data suitable for the assessment of using the MAS DSML within each development step.

MAS DSMLs are expected to shorten development time, especially by reducing design time and implementation time. Design time can be shortened by providing an understandable DSML interface that facilitates MAS modeling via a graphical editor. A code generation utility can also enable developers to write less delta code by providing as many case study requirements as possible. From this perspective, the relationship between modeling / design time and implementation time can be analyzed. In the evaluation process, the development times can be analyzed specifically for the relevant case studies and can be used to analyze the average times elapsed for all case studies. As shown in the next section, AgentDSM-Eval presents a suitable environment to realize the above analysis on the development time by automatically processing the case study results, making the required calculations and automatically producing comparison tables. In these tables, it is also possible to examine whether using a MAS DSML shortens the development time by comparing times elapsed for developing the same agent systems without using this DSML.

The analysis of the *Development Throughput* is carried out from two different perspectives. Firstly, by examining the modeling outputs achieved from each language evaluator for each case study, the usage frequency of modeling elements belonging to a MAS DSML is determined. This assessment is important since it helps to infer which of the developers mostly adopt meta-entities and/or viewpoints of the MAS DSML. It may also give clues on improving MAS DSMLs in terms of their language elements, e.g., less commonly used or unused language elements can be eliminated or replaced in the future versions of the language. The related assessment can be performed for a specific case study, or an average result covering all case studies can also be achieved within AgentDSM-Eval. To the best of our knowledge, such kind of MAS DSML evaluation according to usage frequencies of modeling elements was not previously performed in any AOSE study.

MAS DSMLs generally enable automatic generation of code from MAS models. Although various code generation approaches (e.g., visitor-based, meta-programming, in-line generation, code annotations, template-based) are followed in MDE of different software systems (Sebastián et al., 2020), we see that current MAS DSMLs mostly use template-based code generation (Syriani et al., 2018) techniques during the creation of artifacts such as source code for the implementation of MAS. For this purpose, the serialized model of a MAS is parsed by a template engine, and a code template is filled with a textual definition conforming to each agent component identified in the model. However, the assessment with AgentDSM-Eval is, in fact, independent from the type of code generation used inside a MAS DSML. It only considers the performance evaluation of the applied code generation within this context, i.e., the amount of the generated code is measured. This performance measurement does not deal with the way of how the code is generated. The high performance of code generation allows the evaluator to add less delta code for MAS's full implementation So, the code generation performance of a MAS DSML stands out as a relevant parameter to be measured. The second perspective of the Development Throughput analysis considers the required evaluation. Based on the conducted MAS development case studies, the code production performance of a MAS DSML is analyzed automatically inside AgentDSM-Eval by comparing the generated code with the final code. Like the previous aspects of the case study analysis, it is possible to realize this development throughput analysis again for a specific MAS development case study. It may also be performed to provide an average result covering all case studies.

2.3 Qualitative Evaluation

The qualitative evaluation of MAS DSMLs inside the AgentDSM-Eval framework is performed by receiving feedback from the developers who experienced using the MAS DSML being evaluated. For this purpose, the method of collecting data from users through a questionnaire is preferred. To formalize the required questionnaire, we benefited from the FQAD framework (Kahraman and Bilgen, 2015), which is proposed for the qualitative assessment of DSLs with a list of quality characteristics. Within FQAD, a number of characteristics, which can be accepted as DSL assessment criteria, were defined. Each identified characteristic owns the description of sub-characteristics with quality measures. According to any FQAD characteristic, the evaluation of a DSL depends on evaluating this language employing the quality measures of this FQAD characteristic.

FQAD characteristics, named Functional Suitability, Usability, Reliability, Expressiveness, and Compatibility, are adopted in AgentDSM-Eval. However, both these characteristics and their definitions, which are initially derived to evaluate DSLs in general terms, are re-engineered and specialized in AgentDSM-Eval for the needs of assessing MAS DSMLs. In addition, AgentDSM-Eval defines a new quality assessment characteristic, called MAS Development with including five new quality measure descriptions. For instance, these new quality measures aim at receiving user's feedback on whether the DSML facilitates MAS development or reporting the DSML's coverage on concepts and relations required to design a MAS software.

As a result of re-engineering FQAD characteristics and adding the new MAS Development characteristic, the qualitative evaluation of MAS DSMLs is carried out in terms of 6 major characteristics and their quality measure descriptions. These six characteristics and a total of twenty-four quality measures are listed in Table 2.1. Upon completion of using a specific MAS DSML inside the multi-case studies, agent developers (evaluators) are requested to assess their experience on using this DSML. They score each quality measure of each characteristic between 1 and 5, where 1 means the worst point and 5 means the best point, for the related DSML feature.

Table 2.1: Quality characteristics and quality measures used inside AgentDSM-Eval.

Quality Characteristic	Quality Measure No	Quality Measure
Functional Suitability	1	All concepts and scenarios of the domain can be expressed in the MAS DSML (completeness).
	2	The MAS DSML is appropriate for the specific applications of the domain (e.g. to express an interaction between two agents) (appropriateness).
Usability	3	The required amount of effort for understanding the MAS DSML is small (comprehensibility).
	4	The MAS DSML's concepts and symbols are easy to learn and remember (learnability).
	5	The MAS DSML has capability to help users achieving their tasks in a minimum number of steps.
	6	The MAS DSML is appropriate for the needs of agent developers (likeability, user perception).
	7	Both operating and controlling the language are facilitated by the MAS DSML's features (operability).
	8	The MAS DSML owns user-friendly graphical notations (attractiveness).
	9	The language provides mechanisms for the compactness of the representation of the program (compactness).
Reliability	10	The MAS DSML protects users against making errors and provides model checking.
	11	The MAS DSML prevents the construction of wrong relations between language elements (correctness).
Expressiveness	12	It is easy to reflect a MAS design into an agent program easily with the DSML.
	13	The MAS DSML provides one and only one good way to express every MAS concept (unique).
	14	Each MAS DSML construct is used to represent exactly one distinct concept in the agent domain (orthogonal).
	15	The language constructs correspond only to necessary agent domain concepts.
	16	The MAS DSML does not contain any conflicting elements.
	17	The abstraction level of the MAS DSML is satisfactory for general modeling of MAS, i.e. it is free from specific definitions of agent deployment platforms such as JADE or JACK.
Compatibility	18	Using the MAS DSML to develop agent models fits in the general development process of MAS. The language can be used as part of an AOSE methodology with process phases and roles.
	19	The MAS DSML is compatible with the MAS domain.
MAS Development	20	The MAS DSML makes MAS development easier.
	21	The MAS DSML is appropriate for the construction of specific agent architectures and/or autonomous agent planning models such as reactive agents or Belief-Desire-Intention (BDI) models.
	22	IDE of the MAS DSML is easy to use and provides a handy interface for software development from different MAS perspectives.
	23	The MAS DSML is powerful enough to implement the general MAS structure with including the construction of agent plans, agent internals, communication between agents and agent interactions with the resources residing in the MAS environment.

	24	The MAS DSML enables graphical modeling of both static and runtime aspects of agents and assists the implementation of agent components with sufficient code generation.

3. AgentDSM-Eval Tool

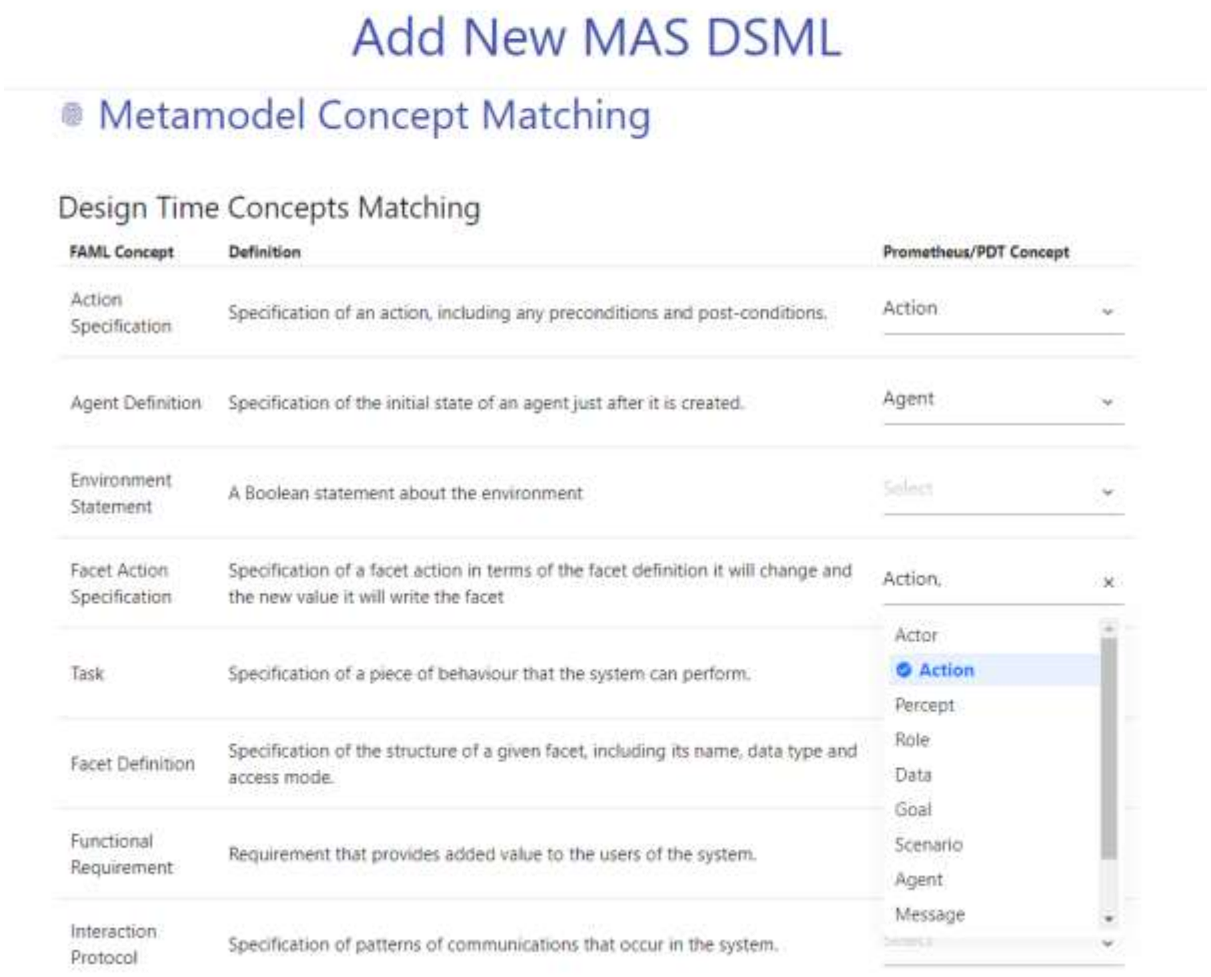
This section introduces the online tool that supports the assessment of MAS DSMLs following the evaluation framework discussed in the previous section. Having the same name as the evaluation framework, the AgentDSM-Eval tool has been designed and implemented as a web application that is publicly available (AgentDSM-Eval, 2019). AgentDSM-Eval tool is implemented based on the Google Firebase platform and hence empowered with Firebase Cloud functions leading to a serverless architecture. Vue.js Javascript library is used for the construction of AgentDSM-Eval's interface. Also, we benefited from ApexCharts.js library for the visualization of the interactive evaluation graphs inside the tool.

The AgentDSM-Eval tool presents a user-friendly GUI to evaluate a MAS DSML from the quantitative and qualitative aspects of the AgentDSM-Eval framework. The tool can be used by both MAS DSML developers and MAS DSML users. MAS DSML developers can create profiles for their DSMLs inside the tool, including the languages' introductory data and metadata definitions. Once a profile is created for a MAS DSML, it is possible to analyze and assess the results of using this DSML inside a multi-case study, performed by various MAS developers who play the evaluator role during the case studies. Evaluation results are stored in the system's repository and can be examined with appropriate graphs, charts, etc. by any users at any time. In the following subsections, the tool's features are discussed within two defined user perspectives: MAS DSML developers and MAS DSML users.

3.1 MAS DSML Developer Perspective

Before evaluating a MAS DSML, a profile for this language needs to be created inside the tool to add this language's specifications. For this purpose, an owner of a MAS DSML, who is mostly a MAS DSML developer from the developer team of this language, first enters the general information about the DSML, including its name, version, owner, and distribution URL, via an online form inside the tool. He/she then uploads the documents for the DSML's metamodel definition and the multi-case study descriptions from the same interface.

The expected format for the MAS DSML metamodel definition is JSON. As discussed previously in subsection 2.2.1, this metamodel file is parsed by the AgentDSM-Eval tool. The meta-entities defined in this metamodel are automatically derived from this file to match them with the entities of the reference MAS metamodel (FAML). In case the DSML's metamodel definition is not available in JSON, it is also possible to manually enter concepts of the related DSML, select them from the given combo boxes and match them again with the reference metamodel entities. Figure 3.1 shows an example from this entity matching screen.


Add New MAS DSML

Metamodel Concept Matching

Design Time Concepts Matching

FAML Concept	Definition	Prometheus/PDT Concept
Action Specification	Specification of an action, including any preconditions and post-conditions.	Action
Agent Definition	Specification of the initial state of an agent just after it is created.	Agent
Environment Statement	A Boolean statement about the environment	Select
Facet Action Specification	Specification of a facet action in terms of the facet definition it will change and the new value it will write the facet	Action,
Task	Specification of a piece of behaviour that the system can perform.	
Facet Definition	Specification of the structure of a given facet, including its name, data type and access mode.	
Functional Requirement	Requirement that provides added value to the users of the system.	
Interaction Protocol	Specification of patterns of communications that occur in the system.	Select

Figure 3.1: The interface for the matching between FAML reference metamodel entities and a MAS DSML's concepts

Definitions of the multi-case studies guiding to evaluate a MAS DSML are given in PDF documents. Each case study's name is entered, and the definition file for this case study is uploaded to the tool. During the execution of the case studies, the evaluators (MAS developers) will benefit from these definitions to develop agent systems according to the agent system requirements given in these files.

3.2 MAS DSML User Perspective

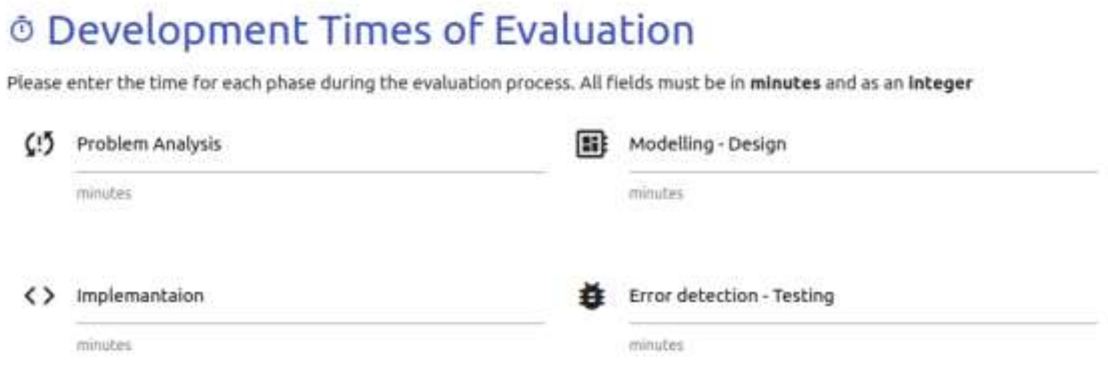
Software developers, who are the end-users of MAS DSMLs, benefit from the online AgentDSM-Eval tool mainly for 1) participating in the multi-case evaluation of a specific MAS (as being an evaluator) and 2) examining the evaluation results of MAS DSMLs (possibly before choosing a DSML to develop a MAS).

Developers participate in the evaluation processes in AgentDSM-Eval by joining an evaluator group, performing MAS design and implementation activities within the multi-case studies and then giving feedback covering their experience on using DSMLs. Upon completion of the multi-case studies, the developers can use the tool to enter the data and upload files required for the quantitative evaluation and also answer the survey for the qualitative assessment.

Before using the evaluation forms of the AgentDSM-Eval tool, developers, who will participate in the study as the evaluators, should first read and approve the online consent letter which confirms their participation is entirely voluntary and they are free to refuse to answer the survey. Moreover, by approving this letter, the developers also accept that their evaluation data can be a part of public research reports but their names and personal data will be hidden and remaining data they entered will be anonymized in these reports.

The approval of the consent letter is followed by the section where the participants enter their personal data including e-mail address, gender, age, field of study (e.g. computer science, software engineering, electrical engineering), completed education (e.g. B.Sc., M.Sc., Ph.D.), knowledge and experience on MAS and current occupation (e.g. student, researcher, worker in the industry).

After obtaining the personal information from each evaluator, the necessary data for quantitative and qualitative analysis are entered. First of all, an evaluator is required to enter the time he/she spent while developing a MAS in each case study. Elapsed time is entered in minutes individually for each step of MAS development, namely problem analysis, system modeling / design, implementation and testing (see Figure 3.2). Only numeric data can be entered and cannot be left blank for these measured times.

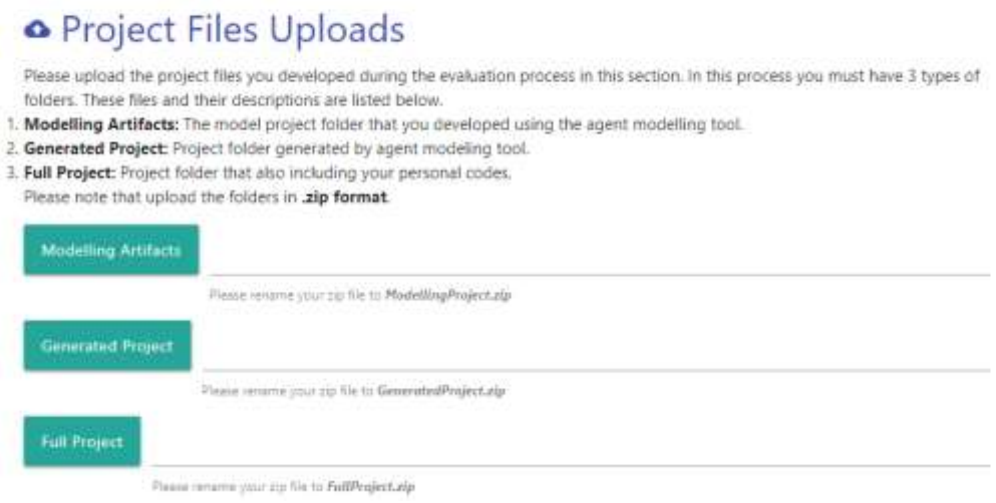

Development Times of Evaluation

Please enter the time for each phase during the evaluation process. All fields must be in **minutes** and as an **integer**

Problem Analysis	Modelling - Design
minutes	minutes
Implemantaion	Error detection - Testing
minutes	minutes

Figure 3.2: Input screen for the times elapsed for MAS development

In addition to the elapsed times, the evaluator is also requested to upload three main compressed files achieved at the end of each MAS development case study. These files should be uploaded in .zip format via the interface shown in Figure 3.3. The first compressed file covers all software model artifacts created by using the DSML being evaluated. The second compressed file includes all code and any other system documents which are auto-generated according to the MAS models designed by this evaluator. Finally, the full project is uploaded as the third compressed file in which delta code written by this evaluator for the full MAS implementation are also included.


Project Files Uploads

Please upload the project files you developed during the evaluation process in this section. In this process you must have 3 types of folders. These files and their descriptions are listed below.

1. **Modelling Artifacts:** The model project folder that you developed using the agent modelling tool.
2. **Generated Project:** Project folder generated by agent modeling tool.
3. **Full Project:** Project folder that also including your personal codes.

Please note that upload the folders in **.zip format**.

Please rename your zip file to ***ModellingProject.zip***

Please rename your zip file to ***GeneratedProject.zip***

Please rename your zip file to ***FullProject.zip***

Figure 3.3: Upload screen for the project files pertaining to each MAS development case study

In the last section, the evaluators answer an online questionnaire to obtain their feedback on using the MAS DSML being evaluated. The questionnaire covers all quality measures of 6 quality characteristics we defined for assessing the DSMLs from the user perspective. Each of these 24 quality measures (previously listed in Table 2.1) is shown along with six radio buttons. The evaluator scores each quality measure by clicking one of these buttons representing points from 1 to 5 on a Likert scale where one means "Very Bad" and five means "Very Good". When the evaluator clicks the sixth button (N/A), it means he/she prefers not scoring the DSML for the related quality measure, i.e., he/she thinks this quality measure is not applicable for the DSML being evaluated. In this case, this scoring will be omitted while calculating the final average point of the DSML for this quality measure achieved in the whole evaluator group.

After completing all input sections discussed above, the system generates a PDF document containing the evaluator's answers. This document is automatically downloaded to the evaluator's computer for backup purposes.

When data from the evaluators' multi-case studies are obtained and the questionnaire is answered by all evaluators as described above, AgentDSM-Eval tool synthesizes all data, automatically processes them to generate quantitative and qualitative evaluation results and present these results to all interested parties including MAS developers and DSML implementers. These online results are always available and may be updated automatically as additional multi-case studies will be performed in the future for the same MAS DSML.

The evaluation results for each MAS DSML are shown to the users in 5 sections covering general information about the language, comparison with the reference model, development time performance, development throughput performance, and questionnaire-based quality assessment.

General information (e.g. name, version, access link) describing the evaluated MAS DSML is shown first. There is also the opportunity to switch into the evaluation interface. For instance, a user can proceed to the evaluation stage by selecting any case study. Conforming to the AgentDSM-Eval analysis specifications previously discussed in Sect. 2.2.1, results pertaining to the comparison of the language's metamodel with the reference metamodel are shown in two different viewpoints in terms of FAML's design time and run time perspectives. Firstly, the average result of the comparison from these two perspectives is shown by a donut graph. The percentage of supported and unsupported FAML concepts inside this MAS DSML is summarized. In the dropdown menus, comparison results are given in detail, i.e. each MAS DSML meta-entity is shown in green when a counterpart is found in FAML or shown in red otherwise. During the presentation of these comparison results, a user can also see the definition of each FAML concept in the related dropdown list.

Development time performance of the DSML, determined according to the case study analysis discussed in Sect. 2.2.2, is shown with the horizontal bar graphs. Average of the times elapsed during each stage (e.g. modeling, implementation) of MAS development with using the MAS DSML inside the multi-case studies are shown in these graphs. The interface provided by the AgentDSM-Eval tool also enables adding / removing the results of a specific case study to / from this comparison graph. Moreover, total average results achieved from the collection of all conducted case studies can be analyzed too.

Taking into account the analysis of development throughput (see Sect. 2.2.2), results showing the output performance of the MAS DSML are classified in two categories inside the tool. The first one presents the usage frequencies of modeling elements belonging to a MAS DSML. These results are achieved as the tool processes all MAS model instances created by all evaluators inside the multi-case studies. In addition to the aggregate frequencies, the comparison of usage frequencies of each modeling element can also be seen separately for each individual case study on the bar graphs.

In the second category, code generation performance of the MAS DSML is displayed. The AgentDSM-Eval tool automatically processes MAS development project files uploaded by each evaluator to the system and it calculates the overall code generation performance. With the provided horizontal bar graphs, it is possible to examine and compare the percentage of the generated code inside the complete code required for the full implementation of the MAS for each case study. The percentages here represent the code generation capability on the average, i.e. the average size of the code the DSML auto-generates from MAS models created by all evaluators. However, it is also possible to see the code generation performance per each conducted case study. Result graphs visualize the ratio of the size of the generated code over the size of the generated plus written code required for the full MAS implementation.

Finally, the results of the questionnaire-based quality assessment of the MAS DSML are shown to the users via two different interfaces. The tool processes all answers given by all evaluators for the AgentDSM-Eval quality characteristics (discussed previously in Sect. 2.3) and presents the average scores in radar graphs given in the first interface. Hence, both the total average score and the average scores specific for 6 different quality characteristics are shown in these graphs. In the second interface, the detailed analyses of the given scores are possible. For each quality characteristic, the distribution of the scores are shown in bar charts. Distribution is given in both total and case study bases. Furthermore, the distribution of the scores given for each specific quality measure (see Table 2.1) of each quality characteristic is also shown to the users, hence the prominent measures per quality characteristics can be easily determined. Apart from showing the overall distribution, these charts again show score distribution of the quality measures for each specific case study.

All above discussed interfaces and the comparison diagrams of the AgentDSM-Eval for the evaluation of MAS DSMLs will be exemplified in the next section.

4. Evaluation of PDT using AgentDSM-Eval Framework

To give some flavor of utilizing the AgentDSM-Eval framework and its tool, the evaluation of one of the well-known MAS DSMLs, called PDT is discussed in this section. First, PDT is briefly introduced and then we discuss how the multi-case study method was applied. Finally, the evaluation results are presented.

4.1 PDT

Prometheus (Padgham and Winikoff, 2005) is an AOSE methodology which aims at simplifying the development process of intelligent agent systems. It specifically focuses on designing agent goals and plans mostly according to the BDI agent model (Rao and Georgeff, 1998). Three fundamental processes inside the Prometheus methodology are system specification, architectural design and detailed design.

During system specification, the objective of the system, usage scenarios and functionality of the system are shaped. Based on the fact that the agents are proactive and target oriented, the process starts by setting goals. Each identified goal facilitates setting subgoals and identification of use cases. After the system features are identified, the architectural design process starts. The aim of this process is to decide the types of agents in the MAS, clarifying the communication protocols between the agents and determining the general system structure. In Prometheus methodology, one or more functionalities are combined to form an agent type. The resulting agent types are modelled using the coupling diagrams and agent acquaintance diagrams. In addition, the characteristics of the communication between two agents are clarified by using the interaction diagrams and interaction protocols. Finally, the general system structure diagram is created in which the types of agents and the interfaces and the limits of the system are described in terms of actions and perceptions. The final process considers the detailed design where the internal design of the agents in terms of capabilities, the processes and the events for interaction protocols and the details of the capabilities in terms of plans and data are determined. Agent overview diagrams and capability description diagrams are used to model the internal architectures of the agents. Capability diagrams are created for elaborating the capabilities.

Prometheus methodology is supported by a software tool, called "Prometheus Design Tool" (Prometheus/PDT), hereafter we shortly call PDT (Thangarajah et al., 2005; PDT, 2011), which is a product of Intelligent Agents Research Group at RMIT University, Australia. PDT also presents a DSML whose graphical concrete syntax enables users to visually model e.g. agents, goals, plans and communications based on the descriptions of the above mentioned Prometheus diagram types. PDT is publicly available as an Eclipse plugin and the developers may create MAS models conforming to Prometheus specifications inside its IDE by drag and drop techniques. Figure 4.1 includes a screenshot from PDT IDE which depicts how a visual model of Prometheus system roles can be created by using the modeling palette at the right.

In addition to supporting the fundamental process of Prometheus, PDT also includes a code generator to achieve implementation of the modeled agents and plans for the JACK agent execution platform (JACK, 2001). JACK is a MAS platform built on the Java programming language. BDI agents can be implemented using JACK API as being the extensions of predefined Java classes. It is a product of AOS Group and used in the development of many industrial autonomous agent applications. Although JACK is a commercial product, it also provides an academic license.

Taking into consideration all the above features, PDT was chosen to demonstrate the use of AgentDSM-Eval framework in this study. Evaluators used PDT to develop agent systems and achieved outputs were evaluated according to the AgentDSM-Eval criteria.

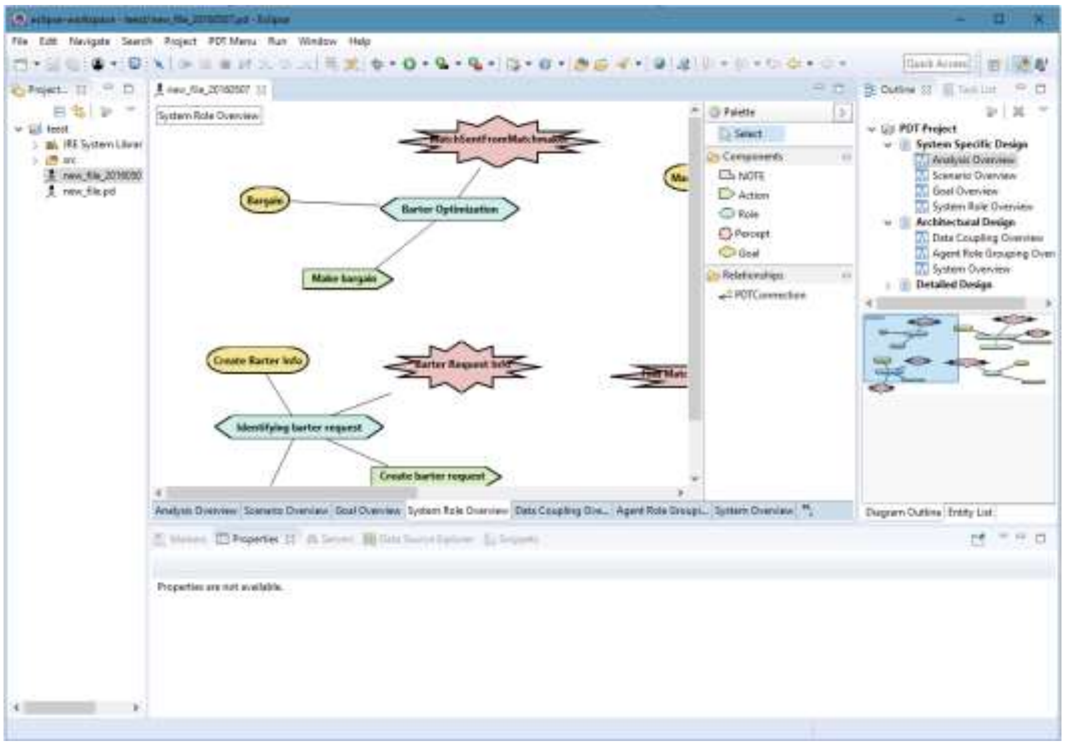
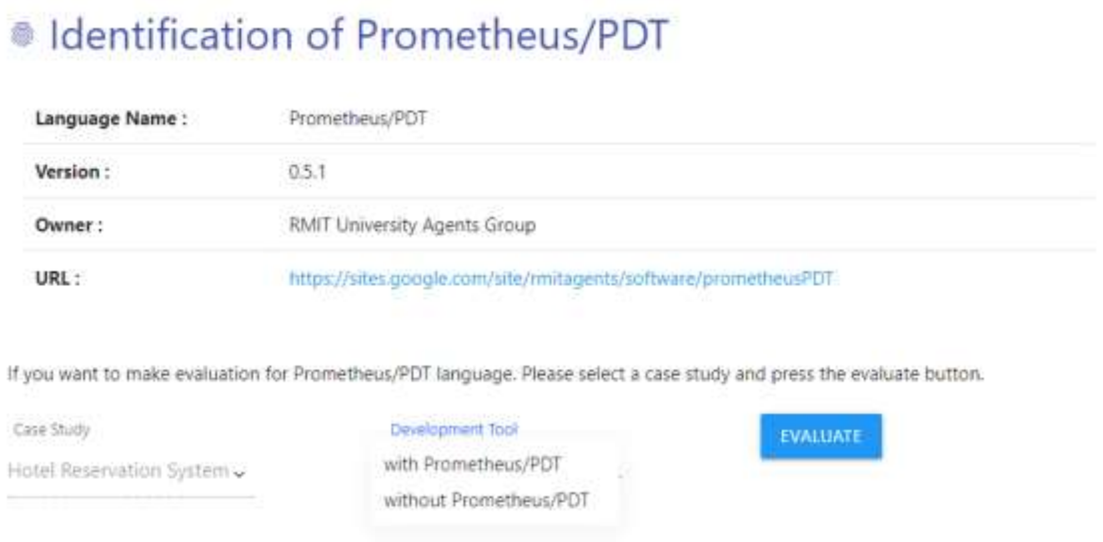


Figure 4.1: A screenshot from PDT IDE (taken from (Padgham and Winikoff, 2005))

4.2 Metamodel Analysis

Before evaluating PDT DSML, its language profile was created inside the AgentDSM-Eval tool by entering its name, version and other identifying information as described in Sect. 3.1. Then, this language's metamodel definition was converted to JSON format and matching its meta-entities (including Actor, Action, Percept, Role, Data, Goal, Scenario, Agent, Message, Capability and Plan) with FAML reference metamodel was completed inside the tool. These matchings were controlled by a MAS DSML developer before the AgentDSM-Eval tool processed them and produced the comparison results. Achieved results can be seen by any user via the provided online interface.

The results, originating from the comparison of PDT metamodel with the design time and runtime concepts of FAML, will be discussed later in Sect. 4.4. When the general information on this DSML was entered and its metamodel comparison with FAML was completed, the interface for the MAS developers to evaluate PDT was ready as seen in Figure 4.2.



Identification of Prometheus/PDT

Language Name :	Prometheus/PDT
Version :	0.5.1
Owner :	RMIT University Agents Group
URL :	https://sites.google.com/site/rmitagents/software/prometheusPDT

If you want to make evaluation for Prometheus/PDT language. Please select a case study and press the evaluate button.

Case Study: Hotel Reservation System

Development Tool: with Prometheus/PDT / without Prometheus/PDT

EVALUATE

Figure 4.2: Welcome screen to start case study based evaluation of PDT

4.3 Evaluation Process

The evaluation of PDT was performed within the context of developing two different MAS. Sixteen agent developers voluntarily participated in this study as evaluators. All 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 the Computer Engineering Department and International Computer Institute (ICI) of Ege University. Six of the evaluators were Ph.D. candidates while the remaining were M.Sc. students. The ages of ten evaluators were between 22 and 25, and the rest were older than 26.

On the average, the evaluators had at least 1.5 years MAS design and implementation experience covering the application of AOSE methodologies and using some agent development APIs like JADE (JADE, 2000) and JACK (JACK, 2001). 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. 60% of the evaluators were also working in the industry at the time of this evaluation performed and they possessed the experience of developing software in industrial scale (4 years on average).

Before the evaluation process, a review on PDT was given to this evaluators group, and then, they developed a mini project to familiarize with PDT IDE and its modeling features. This step ensures countervailing their level of familiarity to the target language and minimizes the threat of validity regarding evaluators.

4.3.1 Selected Case Studies

Evaluators developed two different MAS inside two case studies. The first case consists of the design and implementation of an autonomous agent-based garbage collector system while the second is the realization of a hotel reservation scenario in which customer agents reserve rooms on behalf of its users. Each case study serves for a different MAS application domain including various features of agents. Moreover, modeling and implementation complexities vary in each case study where the developers face the changing difficulties of developing both agent internals and MAS organizations. As can be seen from the following descriptions of the case studies, the implementation of the case study 1 is expected to be relatively easy when we consider the number of agent types and the coverage of each agent internal structure including agent beliefs, goals and plans. However, the case study 2 requires the design and implementation of more complex agent internals as well as complicated agent interactions inside the MAS.

In the following, the description of each case study is given including the system requirements.

1. ***Case Study 1 - Garbage Collector System:*** You are requested to analyze, design and implement a multi-agent system aimed at collecting garbage in different types in a specific environment. There are 2 types of agents in the system. The Manager agent will send a message to the Collector agents. The message is about garbage (there will be 3 types of garbage: plastic, paper and glass) in the environment. Collector agents will reply with a message to the Manager agent indicating whether the relevant garbage can be collected or cannot be collected, according to the belief which is expressing the type of garbage it can collect. The answers about rejection or acceptance of collecting the garbage will be sent to the Manager with two separate plans. If a message, saying a garbage can be collected, is received (the case where the collector agent confirms the garbage collection), then the Manager agent will assume that the garbage has been collected. The garbage in the environment can be created statically during the initialization of the Manager agent.

2. ***Case Study 2 - Hotel Reservation System:*** You are requested to analyze, design and implement a multi-agent hotel reservation system. In the system there should be

> Customer Agents to represent the users seeking hotel rooms, Hotel Agents to represent the hotels and Matchmaker Agents to mediate the interaction between customer and hotel agents. A Customer Agent, who wants to reserve hotel rooms on behalf of a user, first asks for the appropriate hotel agents to a Matchmaker Agent who registers Hotel Agents representing real hotels in this scenario. Following their initialization, each Hotel Agent should send its communication address and the location of the hotel it represents (city name) to the Matchmaker Agent. A Customer Agent requesting a room reservation, searches for Hotel Agents by communicating with the Matchmaker Agent by giving the desired location. Matchmaker Agent responds back with the addresses of the appropriate hotel agents. Upon receiving the addresses of the hotel agents, the Customer Agent immediately sends query messages to all hotel agents indicating the hotel rank (e.g. five-star) and the room price. Conforming to the Contract Net Protocol, the hotel agents may or may not answer to this query within a predefined deadline (e.g. 30 seconds). Hotel agents can randomly decide whether to reply or not. Customer agent receives the replies and chooses one of the replying agents to make the reservation. If just one hotel agent replies positively, the reservation will automatically be made on this hotel. If multiple proposals are received, then the Customer Agent's decision should be based on the lowest-price or first come first served basis when more than one lowest prices exist. Location, rank and price information for each Hotel Agent can be given during the initialization of these agents.

Above descriptions of the case studies were given to the evaluators and they were requested to develop the required agent systems with or without using PDT. Each case study was performed in two sessions. In the first session, the evaluators developed the requested MAS by using PDT, completing the auto-generated JACK code for full implementation and testing the system. In the second session, they developed the whole system without using PDT, i.e. they can apply any AOSE methodology but they should code the system from scratch. During each session, the evaluators saved the times they spent for each stage of MAS development. The evaluators approved the online consent letter and filled the personal information form. Then they entered all measured times and uploaded project files to the AgentDSM-Eval tool via the interface previously shown in Figure 3.3. All sessions were held in the software research laboratory of ICI at Ege University. Upon completion of the MAS development sessions, all evaluators answered the online questionnaire to give their feedback on using PDT DSML.

4.4 Evaluation Results

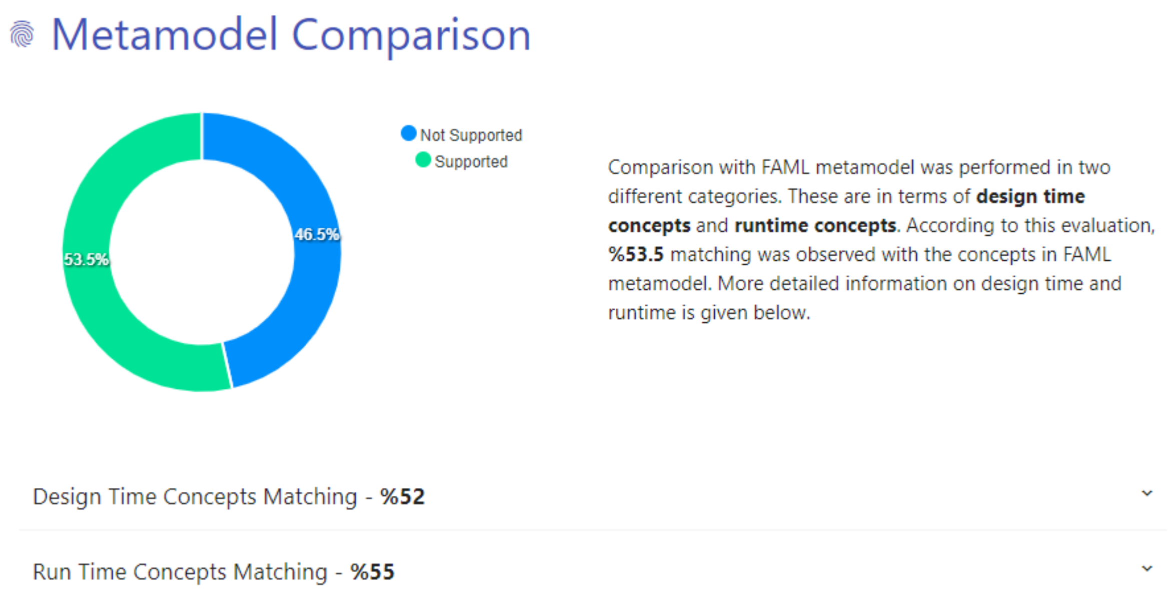
Based on the evaluators' outputs achieved from the multi-case MAS development studies, the AgentDSM-Eval tool generated the results for evaluating PDT. In the following subsections, results on the comparison with the reference model, development time performance, development throughput performance and questionnaire-based quality assessment are given in two main titles as quantitative evaluation and qualitative evaluation.

4.4.1 Quantitative Evaluation Results

As discussed in Sect. 2.2, the quantitative evaluation covers the analysis on the comparison with the reference metamodel, the development time and development outputs.

4.4.1.1 Results of comparing PDT metamodel with the MAS Reference Metamodel

The AgentDSM-Eval tool processed the serialized metamodel of PDT, given in JSON and compared it with the FAML metamodel to generate the ratio of compatibility level. Figure 4.3 shows the results. As can be seen from this figure, PDT supports 53.5% of both design time and runtime components of FAML. That can be interpreted as PDT partially enabling the general modeling of agents and the interaction between the agents when we consider the FAML specifications for MAS design. Moreover, PDT's support on MAS runtime concepts seems better than design time, i.e. it is capable of modeling agent goals, beliefs and executed plans more than the specification of interaction protocols, MAS organization ontologies and resource specifications.



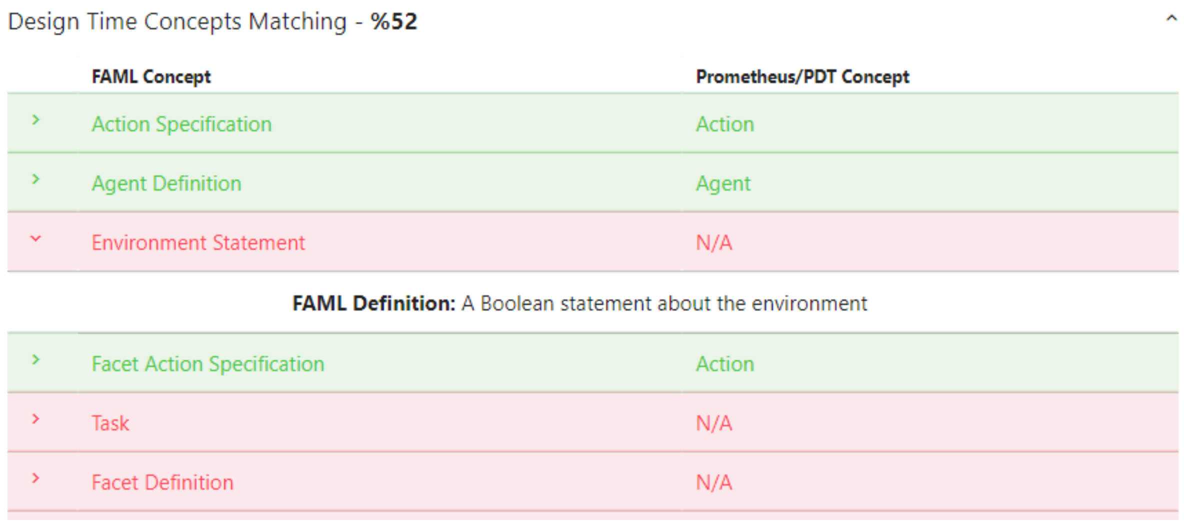
Comparison with FAML metamodel was performed in two different categories. These are in terms of **design time concepts** and **runtime concepts**. According to this evaluation, **%53.5** matching was observed with the concepts in FAML metamodel. More detailed information on design time and runtime is given below.

Design Time Concepts Matching - **%52**

Run Time Concepts Matching - **%55**

Figure 4.3: Comparison results of PDT metamodel with the reference metamodel

In Figure 4.3., there are dropdown menus (below the donut chart) from which the details of metamodel matching are available. The users can benefit from these menus to see which FAML concepts are represented in PDT from both design time and run time perspectives. Each concept, which has a match in PDT DSML, is shown in green color while the others are marked with red (see Figure 4.4) inside these menus of AgentDSM-Eval tool. In addition, when a matching line is clicked, the definition of each concept of the reference model is shown to guide users during examination of these results.



Design Time Concepts Matching - **%52**

FAML Concept	Prometheus/PDT Concept
Action Specification	Action
Agent Definition	Agent
Environment Statement	N/A
FAML Definition: A Boolean statement about the environment	
Facet Action Specification	Action
Task	N/A
Facet Definition	N/A

Figure 4.4: Detailed list of matching between PDT and FAML concepts

4.4.1.2 Results of Case Study Analysis

According to data obtained from the evaluators' MAS development sessions, AgentDSM-Eval tool generated the results on PDT's performance on development time and development throughput. The screenshot given in Figure 4.5 displays the horizontal bar graph of the elapsed times in each case study. Measured times are given in minutes for each stage of system development. Average of the case studies is given in the orange bar named "All Case Studies". It is worth indicating that this graph is dynamically generated by the AgentDSM-Eval tool. It is possible to add or remove any case study from this comparison via using the menu at the right side of the graph. Hence, when a new case study is performed for the evaluation of a MAS DSML, AgentDSM-Eval integrates measured times for this case study to the existing results for this MAS DSML without any human intervention.

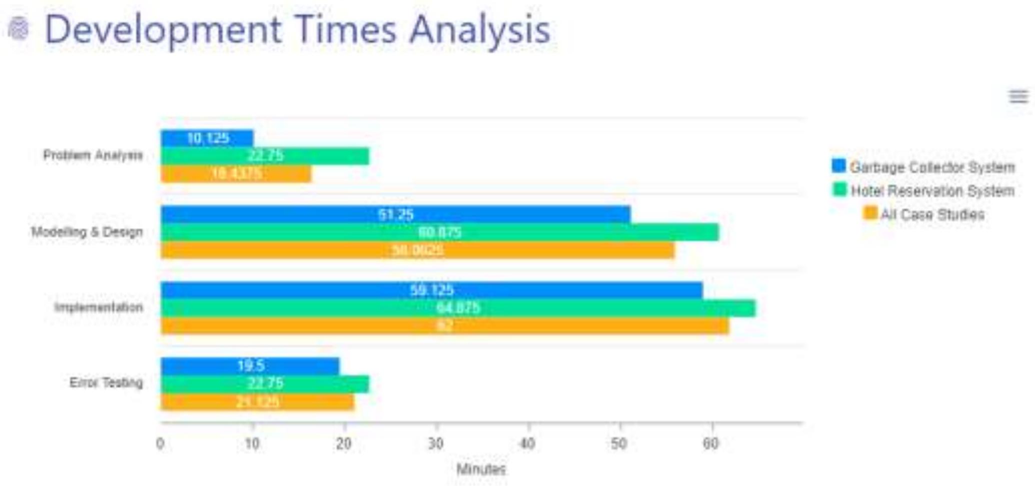


Figure 4.5: Comparison results of the development times elapsed for the case studies using PDT

The graph in Figure 4.5 shows that in every stage of MAS development, the developers (evaluators) spent more time while developing the Hotel Reservation System (Case Study 2) in comparison with the Garbage Collector System (Case Study 1). This confirms that the complexity of both implementing internal architectures of each agent and constructing the messaging between the agents was higher in the second case study. One interesting result originates from comparing modeling and implementation times. Evaluators spent similar times for modeling the requested MAS with PDT and completing the generated JACK code to achieve the full implementation. Although the modeling and programming capabilities of the evaluators have the effect on this result, one can deduce the utilization of PDT's modeling IDE is more challenging than the expected and/or PDT's translational semantics lead to the generation of a limited number of artefacts. However, we need to consider the development of the same systems without using PDT to evaluate both PDT's development time performance in general and compare its effect on modeling / design and implementation stages. Once a user needs to reach the related results, AgentDSM-Eval also produces the comparisons between the times elapsed for implementing MAS with using a MAS DSML and without using this DSML. During the execution of the multi-case studies, our evaluators participated in developing the same agent systems without using PDT in additional case study sessions as discussed in Sect. 4.3.1. Hence, AgentDSM-Eval tool also processed these results and created the graphs for comparing times elapsed for each MAS development with and without using PDT. Screenshots given in Figure 4.6 and Figure 4.7 include these comparisons.

The comparison within the scope of each specific multi-case study is available as shown in Figure 4.6. In this figure, we can see the comparison of times elapsed for each stage of MAS development during the second case study, Hotel Reservation MAS. Based on these results, it is shown that the evaluators spent similar times during problem analysis (with or without using PDT). This is expected since the problem analysis depends on the complexity of the MAS and is independent of the development language. So, use of PDT had a minor effect on the problem analysis. On the other hand, MAS modeling took more time during development using PDT. The main reason is the developers designed the agent models of the Hotel Reservation System in various PDT graph types (see Sect. 4.1) and static semantic controls were made on these models before code generation. Thus, modeling with PDT extended the length of the design time. However, that sacrifice in design time brought a significant gain during the implementation of the MAS. On the average, the evaluators spent almost twice as much time during implementation without using PDT. Auto-code generation plus code completion took approximately 65 min. while preparing all required JACK code from scratch took 132 min. Use of PDT also decreased the testing of the related MAS implementation.

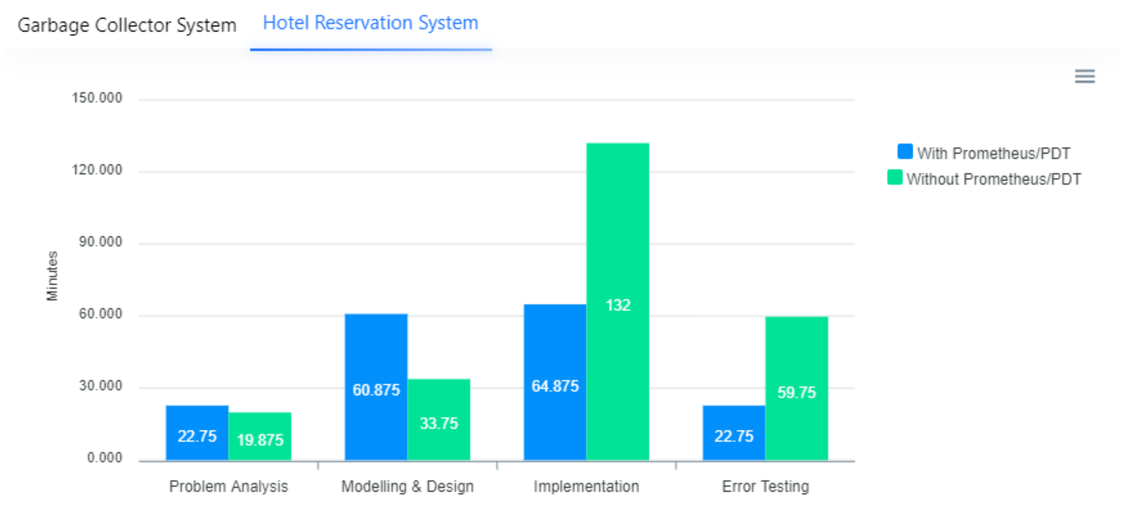


Figure 4.6: Comparison of times elapsed for developing Hotel Reservation MAS with and without using PDT

Development time comparison from the averages of all conducted case studies is also available from the interface provided by the AgentDSM-Eval tool. In Figure 4.7, the comparison of average times of MAS development with and without using PDT is shown. The results show that evaluators saved time mostly in the implementation stage when they used PDT. In general, use of PDT reduced the total MAS development time to approximately three quarters. Addition of the measurements from new case studies may naturally affect these results. Evaluators may easily enter these new measurements into AgentDSM-Tool and the tool automatically updates the existing comparison results by integrating these new ones.

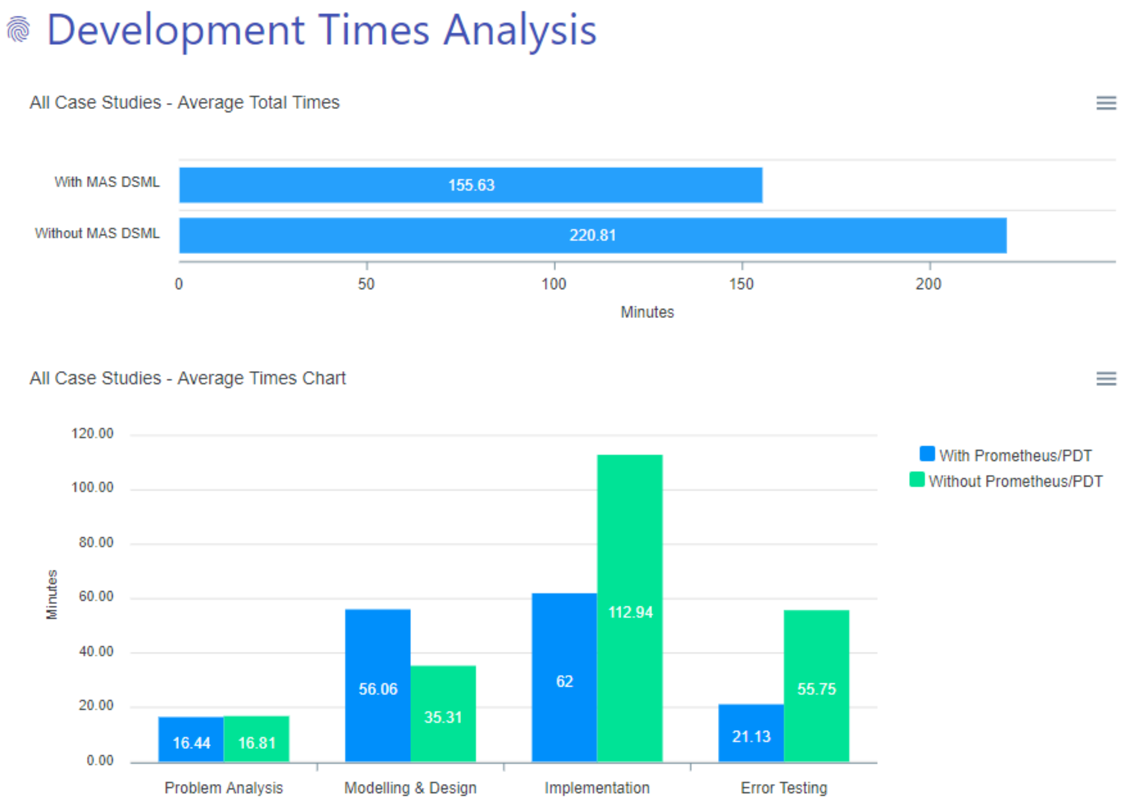


Figure 4.7: Comparison of times elapsed for developing MAS with and without using PDT

The analysis according to the development throughput starts with the comparison of the usage frequencies of PDT meta-entities. As previously discussed in Sect. 2.2.2, one of the novel features of the AgentDSM-Eval framework is to provide users which meta-entities of a MAS DSML are mostly adopted and used by the agent developers during MAS implementations. For this purpose, the tool processes all instance models created by the agent developers, determines the frequencies of each modeling element (instances of the language's meta-entities) and generates graphs reflecting these frequencies. The whole process is automatic. During the evaluation of PDT, the same process was executed and the tool created the comparison graph shown in Figure 4.8. In this graph, each horizontal bar shows the number of occurrences of a modeling entity for each case study. Moreover, next to individual case study results, the graph also shows the total number of times a given modeling entity is used (in orange-colored bars). By using the menu at the right, the results pertaining to a specific case study can be added or removed for the comparison.

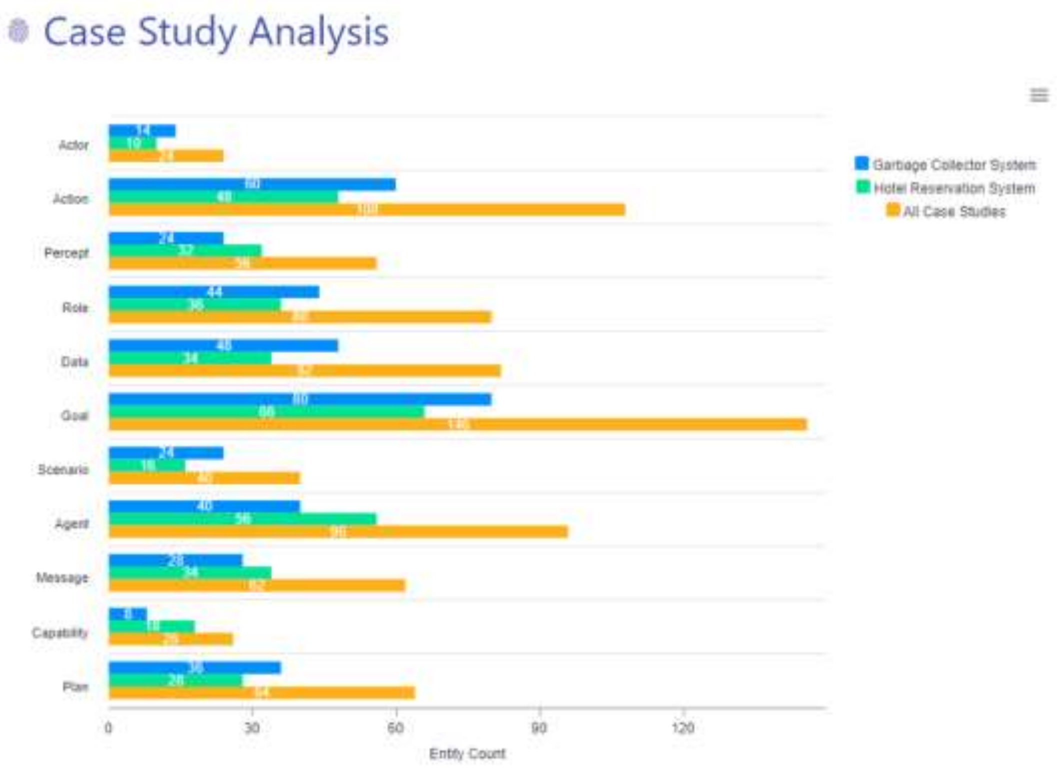


Figure 4.8: The usage frequencies of PDT language elements during multi-case studies

According to these results, the developers used mostly the instances of Goal and Action meta-entities during modeling the Garbage Collector MAS while Capability and Actor elements were used the least. For the Hotel Reservation MAS, again Goal and Action entities were used as well as Agent instances while Actor, Scenario and Capability entities were less frequently used. Originating from the aggregate results, one can say that the developers mostly preferred using PDT's Action, Goal and Agent elements during modeling whereas Capability and Actor elements were used rarely. The results should be interpreted by also taking into account both the complexities of each case study and appropriateness of each model element to the specific needs of these case studies. Hence, the results may change with the inclusion of new MAS implementations. Based on the current results, designers of PDT may think of re-engineering the formulation of Capability and Actor models inside PDT metamodel definition.

Code generation performance of PDT was also evaluated during the development throughput analysis. For this purpose, AgentDSM-Eval tool automatically parsed project files uploaded by all evaluators, processed them to determine the ratio of the auto-generated lines of code (LoC) inside the whole implementation and displayed these ratios for individual case studies as well as the average of these case studies. Figure 4.9 is the screenshot taken from the AgentDSM-

Eval tool which shows PDT's code generation performance. According to this graph, we can see that 39% of the code required for implementing the Garbage Collector MAS were automatically generated by only modeling with PDT. That percentage is the average of all evaluators. Similarly, an evaluator needed to complete 64% of the code to implement the Hotel Reservation MAS on the average. Based on these measurements, approximately 38% of a MAS implementation can be achieved automatically by just using PDT.

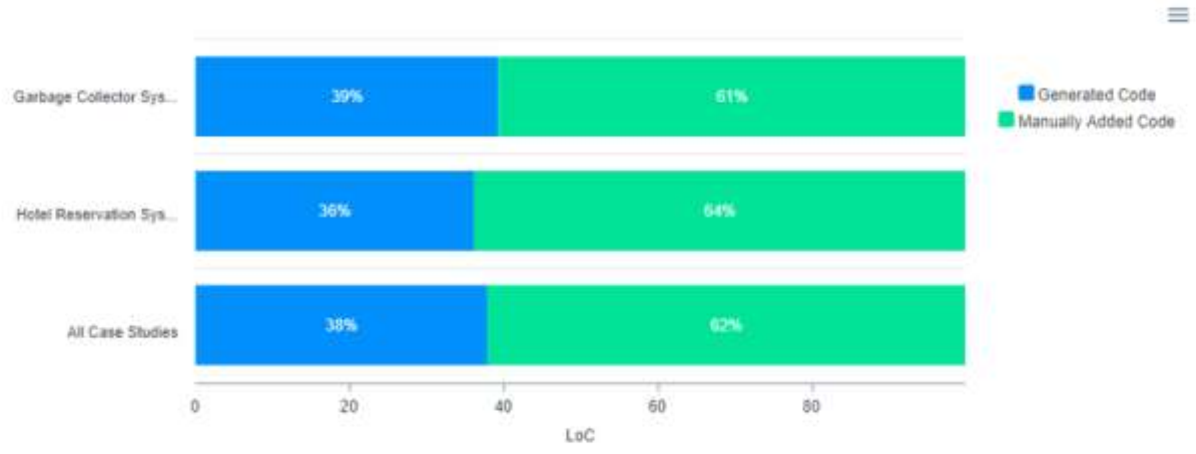


Figure 4.9: PDT language's code generation performance

4.4.2 Qualitative Evaluation Results

Use of PDT was assessed qualitatively according to the AgentDSM-Eval framework's quality characteristics and their quality measures (introduced in Sect. 2.3). For this purpose, agent developers, who participated in the multi-case study, answered the online questionnaire which is composed of 24 quality measures listed in Table 2.1. The evaluators scored each quality measure between 1-5 points. All scores were processed inside the AgentDSM-Eval tool and average scores were calculated. In addition to PDT's general assessment score, the average scores achieved both for each quality characteristic and this characteristic's each specific quality measure were calculated.

In the first screen, AgentDSM-Eval displays average scores for each quality characteristic and total average via a radar graph. Figure 4.10 showed the results for PDT. When a user clicks one of the quality characteristics, the average score for this characteristic is displayed. Moreover, by using the dropdown menu at the top, scores for the quality measures of the quality characteristic selected on the radar graph are displayed in bar charts.

Based on the evaluators' answers to the questionnaire, PDT got 3.5 points for Functional Suitability, 3.13 points for Usability, 3.13 points for Reliability, 3.14 points for Expressiveness, 3.28 points for Compatibility and 3.06 points for MAS Development. The average of these quality characteristics is 3.21 points which is the overall result for the quality assessment.

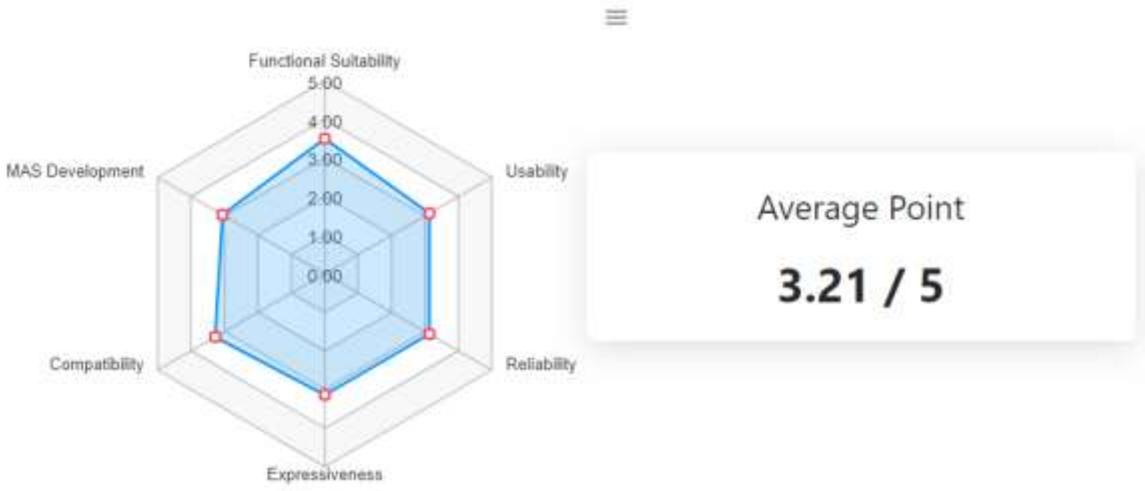


Figure 4.10: PDT's qualitative evaluation results

From these scores, we can see that the evaluators generally found the features of PDT useful in MAS development since all of the scores are above the average in the 1-5 points scale. Functionality of PDT was mostly confirmed which means the evaluators agreed on PDT's support on expressing the domain-specific concepts and its suitability for the specific agent applications. Taking into consideration the points given to the quality measures of Usability characteristics, PDT's graphical notations were mostly found user friendly. However, the evaluators also thought that design with PDT is quite difficult, i.e. it requires many design steps to complete the model.

For the reliability characteristic, the evaluators generally acknowledged PDT's static semantics controls, i.e. preventing the wrong relation constructions while model checking features need to be improved. On the other hand, Expressiveness of the language was mostly confirmed since the orthogonality of the MAS concepts defined in PDT was found sufficient by the evaluators as well as the level of abstraction from specific agent platforms was appreciated. However, the evaluators found that transition from design to agent programs was quite challenging and the notations in the language needed to be revised to represent each agent concept.

Compatibility with AOSE methodologies was generally confirmed with the evaluators' responses, which means PDT fits in the general development process of MAS. Although the evaluators mostly found PDT's MAS Development features above the average, the assessment scores given for this characteristic were lower in comparison with the previous characteristics.

AgentDSM-Eval tool presents additional graphs and charts for the detailed analysis of the scores when a user clicks a quality characteristic in Figure 4.10. Number of answers and their distribution can be seen from these graphs. Also, the average score received for each quality measure is also displayed in addition to the distribution of the answers for these specific quality measures. Furthermore, all of these distributions are displayed for each case study. Due to space limitations, here, we give examples of these detailed graphs only for the analysis of MAS Development characteristics. Graphs for the remaining quality characteristics and their quality measures, generated for PDT evaluation are available online in (AgentDSM-Eval, 2019).

The evaluators were requested to score PDT's MAS development features according to the measures numbered between 20 and 24 in Table 2.1. Figure 4.11 shows the general distribution of the scores given for all quality measures of MAS Development characteristic. Points given by the evaluators are mostly within the range between 2 and 4. 1 and 5 points were rarely given. In the same bar graph, AgentDSM-Eval also shows the distribution of the scores specific for each conducted case study with the different colors. Results for any case study can be added to or removed from the graph using the menu icon resided at the upper right corner.

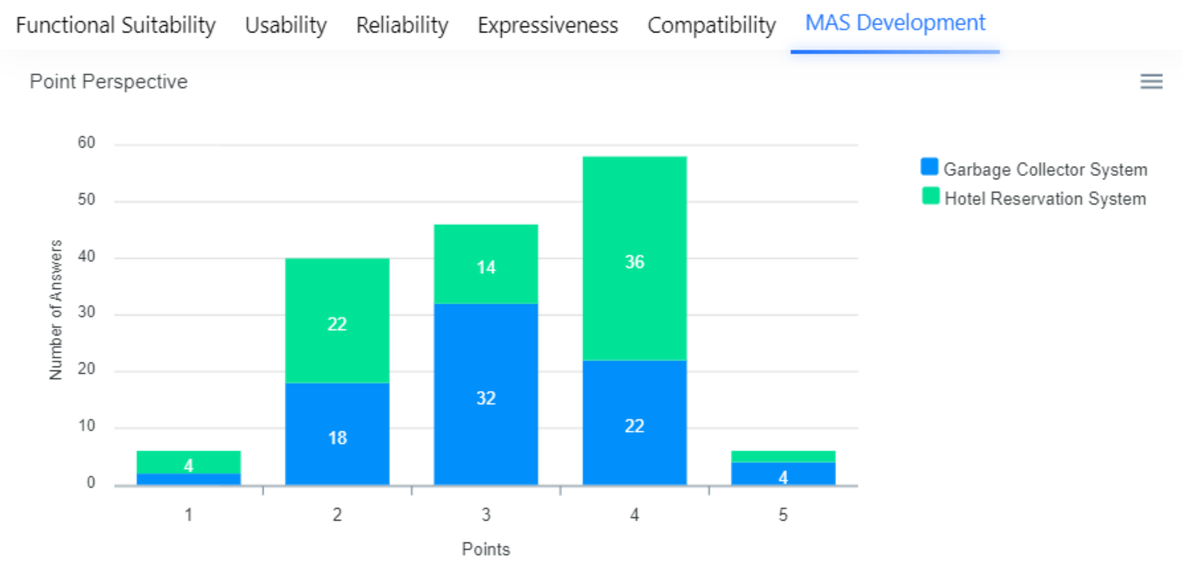
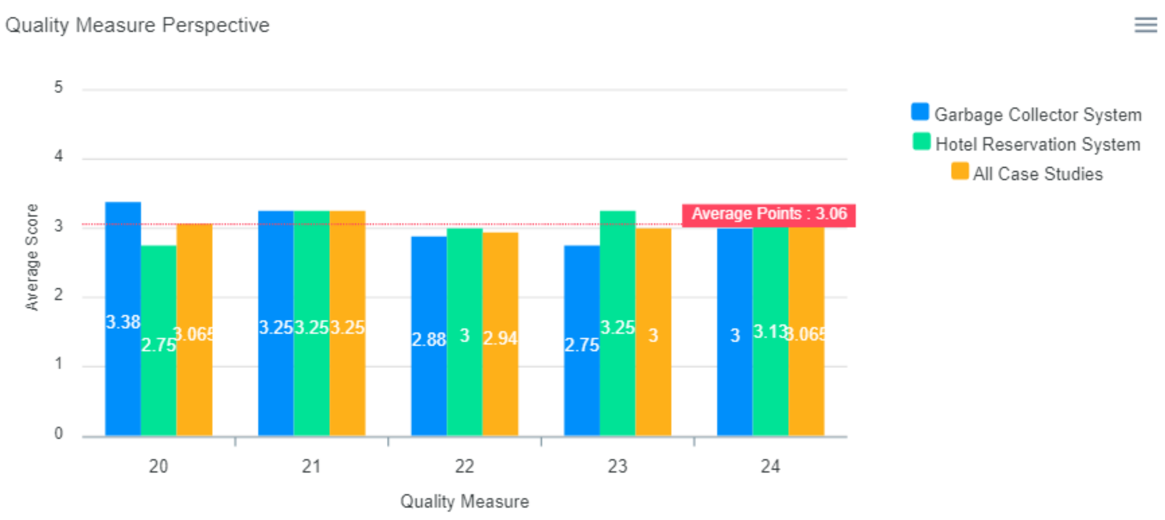


Figure 4.11: Distribution of the scores given for PDT's assessment according to MAS Development characteristic

The average of the scores given for each quality measure of MAS Development characteristic is also displayed inside the AgentDSM-Tool (see Figure 4.12). The x-axis of the bar chart is dynamically created with the numbers of the quality measures being inspected. Description of each measure is also listed at the bottom of the bar chart in dropdown menus to guide the users. Again, the average scores are given both for total and each specific case study. Red line in the graph shows the grand average for the quality characteristic, here MAS Development. When each quality measure description is clicked in the dropdown menu, the score distribution for this quality measure is also displayed as similar to the graph in Figure 4.11.

Based on the current distribution, we can say that the evaluators found PDT is capable of modeling agents according to the well-known agent architectures. For PDT, that support mainly originated from its pre-built components for BDI modeling. It seems that the graphical modeling environment of PDT was generally accredited by the evaluators based on the average score for the quality measure no. 24. However, in comparison with the other quality measures, PDT's IDE was evaluated as quite complicated for MAS development from different modeling viewpoints. In fact, this score is consistent with PDT's assessment within the remaining characteristics as we previously discussed at the beginning of this subsection. For instance, the evaluators also found the use of PDT a little challenging in the sense of complex design steps. Moreover, the transition from design to implementation was scored relatively lower when compared to the other quality measures of the Expressiveness characteristic. Nevertheless, PDT was evaluated as a convenient language for MAS development since it was found above the average for the corresponding quality characteristic.



20 . The MAS DSML makes MAS development easier.

21 . The MAS DSML is appropriate for the construction of specific agent architectures and/or autonomous agent planning models such as reactive agents or Belief-Desire-Intention (BDI) models.

22 . IDE of the MAS DSML is easy to use and provides a handy interface for software development from different MAS perspectives.

23 . The MAS DSML is powerful enough to implement the general MAS structure with including the construction of agent plans, agent internals, communication between agents and agent interactions with the resources residing in the MAS environment.

24 . The MAS DSML enables graphical modeling of both static and runtime aspects of agents and assists the implementation of agent components with sufficient code generation.

Figure 4.12: Average scores given to PDT for each quality measure of MAS Development characteristic

As previously discussed, AgentDSM-Eval's quality characteristics and quality measures can also be used to compare various MAS DSMLs, guiding developers when choosing the most appropriate language for the MDE of a MAS. To exemplify how these characteristics and measures can be used for this purpose, the evaluators are asked to answer the same questionnaire for three additional MAS DSMLs on which they have prior experience. These DSMLs are SEA_ML, Sam, and DSML4BDI.

SEA_ML (Challenger et al., 2014; Challenger et al., 2018) provides modeling MAS with various viewpoints, each representing a different aspect for developing agent systems, especially considering their interactions with the web services defined by service ontologies. The built-in SEA_ML code generator leads to achieving MAS source code for MAS implementations and execution platforms, including JADE (JADE, 2000) and JACK (JACK, 2001). Sam (Faccin and Nunes, 2017) supports the design and implementation of BDI agents. It enables both the graphical design of BDI plans and implementing these plans for BDI4JADE agent execution platform (BDI4JADE, 2011). Finally, DSML4BDI (Kardas et al., 2018) also provides the MDE of agents according to BDI principles. The agent structure, composed of plans, beliefs, rules, and goals, can be modeled with DSML4BDI as well as visually designing the logical expressions required for creating agent plans. Agent descriptions derived from DSML4BDI models can be executed on the Jason platform (Jason, 2007). In addition to the evaluators' prior knowledge and experience, the above languages are selected for this evaluation since they are fully-functional with their modeling and implementation tools that are accessible online and running at the time of conducting this evaluation.

Table 4.1 lists the evaluators' average points for all these MAS DSMLs for each specific AgentDSM-Eval quality characteristic. As can be seen, PDT's scores are also added to this table to compare these DSMLs. According to this evaluation, three languages, DSML4BDI, PDT, and SEA_ML, scored above the average when we consider all MAS DSML quality characteristics, i.e., the evaluators confirmed that using these languages can be beneficial in the development of MAS.

SEA_ML got the highest point for the functional suitability, which means almost all evaluators agreed on SEA_ML's support on modeling specific MAS applications and its expressive power on the wide range coverage of agent domain concepts required to construct various MAS. SEA_ML's features on both enabling the MAS's reliability being designed and facilitating the MAS development with a convenient IDE are well acknowledged.

Sam got the lowest points, and hence, in the overall assessment, it felt behind the average. The developers considered the relatively weak compatibility of the language with the MAS domain. The same also happened with respect to comprehensibility, learnability, operability, and the general usability of the language's graphical notations.

PDT seems to represent a fair MAS DSML with moderate features within this comparison group. Although it was scored above the average for all AgentDSM-Eval quality characteristics, the results showed that the evaluators prefer DSML4BDI or SEA_ML in pairwise comparisons with PDT taking into account all quality measures.

As it is clearly shown in the results, DSML4BDI language was favored by the evaluators within this group. Usability of DSML4BDI's modeling notations and IDE and its level of concept coverage and suitability to the applications domain were all confirmed with the higher points almost closer to 5. Although many features of DSML4BDI helped to receive this score, probably it came to the fore with its built-in support of detailed logical expression modeling required for constructing elaborate agent plans and having visual notations with more customizable and dynamic representations based on the newest Sirius environment (Sirius, 2015) instead of solely Graphical Editing Framework (GEF) (GEF, 2004), as is the case in the remaining MAS DSMLs, which is quite old and difficult to use.

Table 4.1: Average scores of each MAS DSML for each AgentDSM-Eval quality characteristic

MAS DSML / **AgentDSM-Eval Quality Characteristic**	DSML4BDI	PDT	Sam	SEA_ML
Functional Suitability	4.71	3.5	3.03	4.98
Usability	4.83	3.13	2.93	3.78
Reliability	4.51	3.13	3.22	4.13
Expresiveness	4.59	3.14	2.97	3.70
Compatibility	3.73	3.28	2.29	3.56
MAS Development	4.26	3.06	3.01	4.05
Overall Score	**4.44**	**3.21**	**2.91**	**4.03**

While these four languages were compared in the above according to the quality characteristics defined in this study, the comparison results herein need to be extended with the quantitative results of applying AgentDSM-Eval's multi-case study and measuring the development time and throughput for all of these different MAS DSMLs similar to the complete evaluation performed for PDT. Hence, with the combination of these quantitative analyses and evaluation, the order and the user preference of these languages may change. However, a complete discussion on conducting and analyzing such a very large-scale comparative evaluation would need further investigation and individually compose another full-length research paper by nature.

4.5 Threats to the validity

As it is the case in any evaluation study, there are also some threats to the performed evaluation's validity. First, a relatively limited number of evaluators could participate in the assessment. Compared to the many other computer science and software engineering disciplines, AOSE is a young research field. Hence, the number of developers having an interest in MAS implementation is relatively low. Also, we had to consider only graduate students with knowledge and experience on programming agents since courses covering AOSE topics are mostly given at the graduate level worldwide. The length and comprehensiveness of the conducted multi-case studies also affected the number of volunteers since they were requested to implement two different MAS using the DSML completely. They had to repeat the development of each of these MAS without using this DSML.

Second, a single evaluator group was used instead of two different groups, which could pose a threat to the execution phase. We experienced using both single and double evaluator groups in our previous empirical studies on evaluating both different MAS DSMLs (e.g., Kardas et al., 2017; Kardas et al., 2018; Miranda et al., 2019) and DSMLs in other industrial domains (e.g., Saritas and Kardas, 2014; Arslan and Kardas, 2020). Using a single group may raise the risk that the evaluators take advantage of their prior development experience using the MAS DSML while developing the same MAS without using this MAS DSML (or vice-versa). Using two groups may minimize this risk. However, in the case of two groups, the qualitative evaluation based on the user feedback will not be completed fruitfully since the groups with or without using the MAS DSML will be different. It is crucial for the questionnaire-based comparison that a single group implements the same MAS with or without using this MAS DSML. There is also the difficulty of creating two homogeneous groups with almost the same level of domain knowledge, experience, and skills. Randomizing the order of the evaluator groups and/or the applied case studies can be an option. For instance, we followed such a randomization technique on evaluating the usability of the syntax of a MAS DSML in (Miranda et al., 2019) where evaluator groups and MAS development case studies are randomized. However, we could not follow the same approach here due to: 1) the enormous size of the case studies, complete implementation of two complex MAS from scratch instead of just modeling as is the case in (Miranda et al., 2019); 2) time limitations for the experiments; and 3) the unavailability of all evaluators for such an extensive repeating model of evaluation. Nevertheless, the current assessment showed that using the MAS DSML succeeded in shortening the development time and generating the executable artifacts.

Third, the structure and the coverage of the case studies may influence the results. Variations on the case studies would affect the evaluation of a MAS DSML's comprehension and support on various aspects of agent modeling. During the conducted evaluation, one case study is selected to examine the language's capability to model agent internals, e.g., goals and plans. In contrast, while the other case study is formalized, especially to investigate whether modeling with this DSML considers the complicated agent interactions inside a MAS.

It is worth indicating that all the above validity threats are only specific to the MAS DSML evaluation discussed in this paper and do not originate from the AgentDSM-Eval framework's features. It is possible to conduct a multi-case study for evaluating a MAS DSML with many more participants and case studies using our tool. The AgentDSM-Eval tool's capacity within this context is only limited to the storage capacity of its underlying technologies Cloud Firestore and Google Cloud Storage, to store all evaluation data and materials. Moreover, using two different evaluator groups and randomizing both these groups and the case studies' order are possible while using the AgentDSM-Eval framework if the risks mentioned above of following such an approach can be minimized. Multiple groups can use both the framework itself and the tool for evaluating the same MAS DSML.

5. Related Work

Originating mainly from various agent metamodel definitions (e.g. Bernon et al., 2005; Omicini et al., 2008; Beydoun et al., 2009; Hahn et al., 2009; Challenger et al., 2011; Garcia-Magarino, 2014; Tezel et al., 2016), AOSE researchers have made significant efforts on the derivation of DSLs / DSMLs for facilitating the MDE of MAS (Kardas and Gomez-Sanz, 2017). Among these studies, Agent-DSL (Kulezsa et al., 2005) is used for modeling agent features, like knowledge, interaction and autonomy. Rougemaille et al. (Rougemaille et al., 2007) define abstract syntax of two agent modeling languages specialized for modeling the adaptivity of agents.

Applying some of the AOSE methodologies are supported with DSML tools for visual MAS modeling and code generation. For instance, PDT (Thangarajah et al., 2005) and Prometheus Graphical Editor (PGE) (Gascuena et al., 2012) enable modeling of agents and their interactions according to Prometheus methodology and lead the implementation of the modeled agents in JACK platform. Similarly, Pavon et al. (Pavon et al., 2006) suggest using a tool called

IDK for agent software development by following the principles of INGENIAS MAS methodology (Pavon et al., 2005). Fuentes-Fernandez et al. (Fuentes-Fernandez et al., 2010) discuss how an agent modeling environment can be generated to support both the lifecycle and the tasks of INGENIAS methodology again.

Hahn (Hahn, 2008) introduces a DSML4MAS language, whose abstract syntax is structured into several aspects, each focusing on a specific viewpoint of a MAS. Graphical notations for the concepts and relations are defined to provide a visual concrete syntax. Furthermore, DSML4MAS supports the deployment of modeled MASs both in JACK (JACK, 2001) and JADE (JADE, 2000) agent platforms by providing an operational semantics over model transformations. The metamodel of DSML4MAS is employed in (Ayala et al., 2014) as a source metamodel to support the modeling of context aware systems. Agents created from these models are run in the ambient intelligence devices.

Ciobanu and Juravle (Ciobanu and Juravle, 2012) define and implement a language for mobile agents. They generate a text editor with auto-completion and error signaling features and present a way of code generation for agent systems starting from their textual description. Likewise, SEA_L (Demirkol et al., 2013; Challenger et al., 2016a) and JADEL (Bergenti et al., 2017) are two agent DSLs both providing textual syntaxes based on Xtext specifications (Eysholdt and Behrens, 2010). Agents and services used by the agents can be modeled with SEA_L and these models can be used to implement agents on JADEX agent platform (JADEX, 2007) which is a reasoning engine for executing BDI agents. JADEL is designed to support the effective implementation of JADE agents by natively supporting agent-oriented abstractions. Finally, Sredejovic et al. (Sredejovic et al., 2018) introduce another agent DSL, called ALAS, to allow software developers to create intelligent agents having reasoning systems based on non-axiomatic logic. It is possible to convert ALAS code to Java code and hence execute agents.

The work conducted in (Goncalves et al., 2015) aims at creating a UML-based agent modeling language, called MAS-ML, which is able to graphically model various types of agent internal architectures. ERE-ML (HoseinDoost et al., 2019) extends MAS-ML according to the concepts of emergency response environments and presents a modeling environment to MDD of agents for disaster management. However, the language only provides static environment modeling and runtime dependent agent interactions and the coordination and collaboration strategies can not be modeled with ERE-ML.

SEA_ML language, introduced in (Challenger et al., 2014), supports graphical modeling of MAS and enables the construction of modeled agents over a series of model-to-code transformations. Specifically, it supports the detailed modeling of the interactions between agents and semantic web services to realize service discovery, agreement and execution dynamics.

Wautelet and Kolp (Wautelet and Kolp, 2016) investigate how a model-driven framework can be constructed to develop BDI agents by proposing strategic, tactical and operational views. 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. Based on this framework, MAS implementations can be built from high level analysis models, called the Rationale Trees (Wautelet et al., 2017).

A development method to design and implement agents via a transformation between agent models to platform-specific code is discussed in (Faccin and Nunes, 2017). This method can be applied by using a modeling tool, called Sam, in which graphical modeling of agents and generating source code for BDI4JADE agent framework (BDI4JADE, 2011) are possible. Sam delegates to the developers the task of implementing domain-specific code that cannot be represented using this tool. DSML4BDI (Kardas et al., 2018) is another DSML proposed for creating agents conforming to BDI architecture. In addition to modeling internal structure of agents, their beliefs, goals, events and knowledgebase, DSML4BDI specifically allows modeling the difficult logical expressions, which might be used in any agent plan or rule. It is possible to generate agent descriptions from these models for implementing them on the open source Jason platform (Jason, 2007), which is an interpreter for an extended version of a Prolog-like logic programming language for BDI agents.

Although all abovementioned DSML studies contribute MDD of MAS by introducing dedicated metamodels and model transformations for agent implementations, the vast majority of these studies do not include an evaluation of both the usability of the proposed languages and generated artefacts. Instead, they just exemplify how the new DSML and its supporting tools can be used to develop agent systems. Only a few of them (Challenger et al., 2016b; Faccin and Nunes, 2017; Kardas et al., 2017; Kardas et al., 2018) can be said considering the evaluation of their proposed methods and/or agent DSMLs.

As we previously discussed in Section 2, the evaluation framework in (Challenger et al., 2016b) brings a way of evaluating MAS DSMLs systematically within the perspective of use cases and specifically shows how this approach can be used to evaluate SEA_ML (Challenger et al., 2014). In fact, AgentDSM-Eval refines the methodology and the metrics of this framework, introduces new quantitative and qualitative metrics to generalize the approach and it specifically enriches the usability assessment of the MAS DSMLs based on the user feedback. DSML4BDI's (Kardas et al., 2018) code generation performance and time savings are measured using Challenger et al.'s framework. Cost of both building model transformations between MAS DSMLs and applying these transformations to extend the execution platform support of these DSMLs over language interoperability (Kardas et al., 2017) is also analyzed using the same evaluation framework. The evaluation of Sam tool in (Faccin and Nunes, 2017) consists of investigating whether the tool both facilitates the agent developers' understanding of an existing BDI agent project and/or improves the evolution of an existing BDI agent project. Moreover, Sam tool's language elements and code generation performance are evaluated in the same study according to Challenger et al.'s framework (Challenger et al., 2016b) again. However, all of these studies take into account evaluating the features of only one specific MAS DSML, which is also created by the same researchers and hence it is difficult to generalize these evaluations to apply for other MAS DSMLs.

Finally, in our recent work (Silva et al., 2018; Miranda et al., 2019), we investigate how the graphical syntax of MAS DSMLs can be improved by applying the principles of the "Physics" of Notations (Moody, 2009). The hypothesis is examined under 4 research goals covering comprehensiveness, usability, effectiveness, and efficiency. The experiments conducted by the participants show that the participants are more likely to select the refined graphical syntax for the agent modeling languages and the new symbols are easy to understand. Use of the approach is demonstrated over the graphical syntax of SEA_ML (Challenger et al., 2014) and DSML4MAS (Hahn, 2008) languages. The structure of these evaluations is naturally not fully-fledged as AgentDSM-Eval since they just concentrate on improving the modeling notations of MAS DSMLs.

6. Conclusion

A general framework, called AgentDSM-Eval, which enables the systematic evaluation of MAS DSMLs according to various quantitative and qualitative metrics, has been introduced in this paper. The framework is supported with an online tool which enables both conducting multi-case empirical evaluation of MAS DSMLs and automatic generation and analysis of the evaluation results. The related tool is publicly available on the AgentDSM-Eval website (AgentDSM-Eval, 2019).

In this paper, we have also discussed how AgentDSM-Eval can be used by giving an example of a comprehensive evaluation performed for the MAS DSML, named PDT. A group of agent developers, playing the evaluator role, experienced using this DSML in a series of MAS development case studies. Data achieved from these evaluators' MAS development sessions were automatically processed inside the AgentDSM-Eval tool. We have ensured that the tool successfully processed these data and, after that, we could obtain qualitative evaluation results of PDT's performance on development time and development throughput. Moreover, AgentDSM-Eval tool realized the automatic analysis on both this language's metamodel and MAS models, created by each individual evaluator, and hence it generated evaluation results regarding MAS domain coverage and agent developers' adoption for language constructs respectively. Finally, evaluators' feedback on using this DSML were obtained and processed inside AgentDSM-Eval tool again and the tool produced qualitative assessment results based on a set of DSML quality characters such as functional suitability, usability, expressiveness and MAS development. All results pertaining to the evaluation of PDT are also available in AgentDSM-Eval tools website.

Both MAS DSML users and MAS DSML developers can benefit from the evaluations conducted with AgentDSM-Eval. On one hand, an agent developer, who intends to use a DSML for a MAS development, can first examine the evaluation results for this DSML and then decide whether this language fits the requirements of his/her own system development. Furthermore, agent developers can also compare the evaluation results of different MAS DSMLs inside the AgentDSM-Eval tool which may assist them in choosing the most appropriate one for the system-to-be-developed. For this purpose, we have exemplified how the qualitative evaluation process brought by the AgentDSM-Eval framework can be used in comparing four different MAS DSMLs including PDT. On the other hand, metamodel comparison degrees and model element usage frequencies calculated by the tool, may guide language developers to improve their MAS DSMLs, e.g. by updating the language syntax and other definitions towards strengthening the MAS domain coverage. Also, feedback gained from the users on the usability and the expressiveness of the MAS DSML may also help implementing the new versions of this language.

The reference metamodel currently used inside AgentDSM-Eval can be enriched e.g. by including the new entities and relations for specific agent modeling viewpoints on e.g. agent mobility, adaptivity or service interaction. For this purpose, we plan to work on extending FAML in our future work. Another option can be integrating any other MAS metamodel into AgentDSM-Eval in addition to the existing reference metamodel. Although FAML sufficiently covers the general model of agent components and their relations, additional metamodels can also be provided to the AgentDSM-Eval users. They could choose this way for comparing the features of a MAS DSML according to specific agent modeling viewpoints indicated above. Finally, the current semi-automatic process of MAS DSML metamodel comparison inside the AgentDSM-Eval tool can be improved. For example, applying metamodel clone detection (Babur et al., 2019) covers feature extraction and statistical model analysis. Hence, it can be possible to automate the matching between the agent concepts in the MAS metamodels being compared. The investigation of enhancing the AgentDSM-Eval tool's capabilities within this context will be our other future work.

Acknowledgement

This study was funded as a bilateral project by the Scientific and Technological Research Council of Turkey (TUBITAK) under grant 115E591 and the Portuguese Foundation for Science and Technology (FCT) under grants FCT/MCTES TUBITAK/0008/2014 and FCT/MCTES PEst UID/ CEC/04516/2013 and NOVA LINCS (UIDB/04516/2020).

References

(AgentDSM-Eval, 2019) AgentDSM-Eval Tool, https://agent-dsml-evaluation-tool.firebaseapp.com/#/ (last access: October, 2020)

(Arslan and Kardas, 2020) Arslan, S., Kardas, G. 2020. “DSML4DT: A domain-specific modeling language for device tree software”. Computers in Industry, 115, 103179: 1-13.

(Ayala et al., 2014) Ayala, I., Amor, M., Fuentes, L. 2014. “A model driven engineering process of platform neutral agents for ambient intelligence devices”. Autonomous Agents and Multi-agent Systems, 28: 214-255.

(Babur et al., 2019) Babur, O, Cleophas, L., van den Brand, M. 2019. “Metamodel clone detection with SAMOS”. Journal of Computer Languages, 51: 57-74.

(Barišić et al., 2018) Barišić, A., Amaral, V., Goulao, M. 2018. "Usability driven DSL development with USE-ME". Computer Languages, Systems & Structures, 51: 118-157.

(BDI4JADE, 2011) BDI4JADE: A BDI Layer on Top of JADE, http://www.inf.ufrgs.br/prosoft/bdi4jade/ (last access: October, 2020)

(Bergenti et al., 2017) Bergenti, F., Iotti, E., Monica, S., Poggi, A. 2017. "Agent-oriented model-driven development for JADE with the JADEL programming language". Computer Languages, Systems & Structures, 50: 142-158.

(Bernon et al., 2005) Bernon, C., Cossentino, M., Gleizes, M. P., Turci, P., Zambonelli, F. 2005. "A Study of some Multi-Agent Meta-Models". Lecture Notes in Computer Science, 3382: 62-77.

(Beydoun et al., 2009) Beydoun, G., Low, G. C., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J. J., Pavon, J., Gonzalez-Perez, C. 2009. "FAML: A Generic Metamodel for MAS Development". IEEE Transactions on Software Engineering, 35(6): 841-863.

(Bresciani et al., 2004) Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J. 2004. "Tropos: An agent-oriented software development methodology". Autonomous Agents and Multi-Agent Systems 8(3): 203-236.

(Challenger et al., 2011) Challenger, M., Getir, S., Demirkol, S., Kardas, G. 2011. "A Domain Specific Metamodel for Semantic Web Enabled Multi-Agent Systems". Lecture Notes in Business Information Processing, 83: 177-186.

(Challenger et al., 2014) 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 et al., 2016a) Challenger, M., Mernik, M., Kardas, G., Kosar, T. 2016. "Declarative specifications for the development of multi-agent systems". Computer Standards & Interfaces, 43: 91-115.

(Challenger et al., 2016b) Challenger, M., Kardas, G., Tekinerdogan, B. 2016. "A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems". Software Quality Journal, 24(3): 755-795.

(Challenger et al., 2018) Challenger, M., Tezel, B.T., Alaca, O., Tekinerdogan, B., Kardas, G. 2018. "Development of semantic web-enabled BDI multi-agent systems using SEA_ML: An electronic bartering case study". Applied Sciences, 8(5):1-32.

(Ciobanu and Juravle, 2012) Ciobanu, G., Juravle, C. 2012. “Flexible Software Architecture and Language for Mobile Agents”. Concurrency and Computation-Practice & Experience, 24(6): 559-571.

(Demirkol et al., 2013) Demirkol, S., Challenger, M., Getir, S., Kosar, T., Kardas, G., Mernik, M. 2013. “A DSL for the development of software agents working within a semantic web environment”. Computer Science and Information Systems, 10(4: 1525-1556.

(Eysholdt and Behrens, 2010) Eysholdt, M., Behrens, H. 2010. “Xtext: implement your language faster than the quick and dirty way”. In proc. Companion to the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (SPLASH/OOPSLA 2010), Reno/Tahoe, Nevada, USA, pp. 307-309.

(Faccin and Nunes, 2017) Faccin, J. Nunes, I. 2017. “A Tool-Supported Development Method for Improved BDI Plan Selection”. Engineering Applications of Artificial Intelligence, 62: 195-213.

(Fuentes-Fernandez et al., 2010) Fuentes-Fernandez, R., Garcia-Magarino, L., Maria Gomez-Rodriguez, A., Carlos Gonzalez-Moreno, J. 2010. “A technique for defining agent-oriented engineering processes with tool support”. Engineering Applications of Artificial Intelligence, 23(3): 432-444.

(Garcia-Magarino, 2014) Garcia-Magarino, I. 2014. “Towards the integration of the agent-oriented modeling diversity with a powertype-based language”. Computer Standards & Interfaces, 36: 941-952.

(Gascuena et al., 2012) Gascuena, J. M., Navarro, E., Fernandez-Caballero, A. 2012. “Model-Driven Engineering Techniques for the Development of Multi-agent Systems”. Engineering Applications of Artificial Intelligence, 25(1): 159-173.

(GEF, 2004) Graphical Editing Framework, https://www.eclipse.org/gef/ (last access: October, 2020).

(Goncalves et al., 2015) 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.

(Goulao et al., 2016) Goulao, M., Amaral, V., Mernik, M. 2016. “Quality in model-driven engineering: a tertiary study”. Software Quality Journal, 24(3): 601-633.

(Hahn, 2008) 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.

(Hahn et al., 2009) Hahn, C., Madrigal-Mora, C., Fischer, K. 2009. “A Platform-Independent Metamodel for Multiagent Systems”. Autonomous Agents and Multi-Agent Systems, 18(2): 239-266.

(HoseinDoost et al., 2019) HoseinDoost, S., Adamzadeh, T., Zamani, B., Fatemi, A. 2017. “A model-driven framework for developing multi agent systems in emergency response environments”. Software and Systems Modeling, 18(3): 1985-2012.

(JACK, 2001) JACK Autonomous Software, http://aosgrp.com/products/jack/ (last access: October, 2020)

(JADE, 2000) JADE: JAVA Agent DEvelopment Framework, https://jade.tilab.com/ (last access: October, 2020)

(JADEX, 2007) Jadex BDI Agent System, https://sourceforge.net/projects/jadex/ (last access: October, 2020)

(Jason, 2007) Jason platform, http://jason.sourceforge.net/wp/ (last access: October, 2020)

(Kahraman and Bilgen, 2015) Kahraman, G., Bilgen, S. 2015. “A framework for qualitative assessment of domain-specific languages”. Software & Systems Modeling, 14(4): 1505-1526.

(Kardas, 2013) Kardas, G. 2013. “Model-driven development of multiagent systems: a survey and evaluation”. The Knowledge Engineering Review, 28(4): 479-503.

(Kardas and Gomez-Sanz, 2017) Kardas, G., Gomez-Sanz, J.J. 2017. “Special issue on model-driven engineering of multi-agent systems in theory and practice”. Computer Languages, Systems & Structures, 50: 140-141.

(Kardas et al., 2017) Kardas, G., Bircan, E., Challenger, M. 2017. “Supporting the platform extensibility for the model-driven development of agent systems by the interoperability between domain-specific modeling languages of multi-agent systems”. Computer Science and Information Systems, 14(3): 875-912.

(Kardas et al., 2018) Kardas, G., Tezel, B. T., Challenger, M. 2018. “Domain-specific modelling language for belief-desire-intention software agents”. IET Software, 12(4): 356-364.

(Kelly and Tolvanen, 2008) Kelly, S., Tolvanen, J.-P. 2010. “Domain-Specific Modeling: Enabling Full Code Generation”. Wiley-IEEE Computer Society, Hoboken, New Jersey, USA, 444 pages.

(Kosar et al., 2016) Kosar, T., Bohra, S., Mernik, M. 2016. “Domain-Specific Languages: A Systematic Mapping Study”. Information and Software Technology, 71: 77-91.

(Kulesza et al., 2005) Kulesza, U., Garcia, A., Lucena, C., Alencar, P. 2005. “A generative approach for multi-agent system development”. Lecture Notes in Computer Science, 3390: 52-69.

(Leitao and Karnouskos) Leitao, P., Karnouskos, S. 2015. “Industrial Agents: Emerging Applications of Software Agents in Industry”. Elsevier Science Publishers, Amsterdam, Netherlands, 476 pages.

(Liang et al., 2019) Liang, C.-C., Liang, W.-Y., Tseng, T.-Z. 2019. “Evaluation of intelligent agents in consumer-to-business e-Commerce”, Computer Standards & Interfaces, 65: 122-131.

(Mascardi et al., 2019) Mascardi, V., Weyns, D., Ricci, A., Earle, C. B., Casals, A., Challenger, M., Chopra, A., Ciortea, A., Dennis, L. A., Díaz, A. F., El Fallah-Seghrouchni, A., Ferrando, A., Fredlund, L.-A., Giunchiglia, E., Guessoum, Z., Gunay, A., Hindriks, K., Iglesias, C. A., Logan, B., Kampik, T., Kardas, G., Koeman, V. J., Larsen, J. B., Mayer, S., Mendez, T., Nieves, J. C., Seidita, V., Tezel, B. T., Varga, L. Z. and Winikoff, M. 2019. “Engineering multi-agent systems: state of affairs and the road ahead”, ACM SIGSOFT Software Engineering Notes, 44(1): 18-28.

(Mernik et al., 2005) Mernik, M., Heering, J., Sloane, A. 2005. “When and how to develop domain-specific languages”. ACM Computing Surveys, 37(4): 316-344.

(Miranda et al., 2019) Miranda, T., Challenger, M., Tezel, B. T., Alaca, O. F., Barišić, A., Amaral, V., Goulao, M., Kardas, G. 2019. “Improving the Usability of a MAS DSML”. In proc. 6th International Workshop on Engineering Multi-Agent Systems (EMAS 2018), held in 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, Lecture Notes in Artificial Intelligence, 11375: 55-75.

(Moody, 2009) Moody, D. 2009. “The “physics” of notations: toward a scientific basis for constructing visual notations in software engineering”. IEEE Transactions on Software Engineering, 35(6): 756-779.

(Omicini et al., 2008) Omicini, A., Ricci, A., Viroli, M. 2008. “Artifacts in the A&A meta-model for multi-agent systems”. Autonomous Agents and Multi-Agent Systems 17(3): 432-456.

(Padgham and Winikoff, 2005) Padgham, L., Winikoff, M. 2005. “Prometheus: A practical agent-oriented methodology”. Agent-oriented Methodologies. Henderson-Sellers, B. Giorgini, P. (Eds.). Idea Group Publishing, pp. 107-135.

(Pavon et al., 2005) Pavon, J., Gomez-Sanz, J.J., Fuentes, R. 2005. “The INGENIAS Methodology and Tools”. Agent-oriented Methodologies. Henderson-Sellers, B. Giorgini, P. (Eds.). Idea Group Publishing, pp. 236-276.

(Pavon et al., 2006) Pavon, J., Gomez-Sanz, J., Fuentes, R. 2006. “Model driven development of multi-agent systems”. Lecture Notes in Computer. Science, 4066: 284-98.

(PDT, 2011) Prometheus Design Tool, https://sites.google.com/site/rmitagents/software/ (last access: October, 2020).

(Rao and Georgeff, 1998) Rao, A.S., Georgeff, M.P. 1998. "Decision procedures for BDI logics". Journal of Logic and Computation, 8(3): 293-343.

(Rougemaille et al., 2007) Rougemaille, S., Migeon, F., Maurel, C., Gleizes, M-P. 2007. "Model Driven Engineering for Designing Adaptive Multi-agent Systems". Lecture Notes in Artificial Intelligence, 4995: 318-333.

(Saritas and Kardas, 2014) Saritas, H. B., Kardas, G. 2014. "A model driven architecture for the development of smart card software". Computer Languages, Systems & Structures, 40(2): 53-72.

(Sebastián et al., 2020) Sebastián, G., Gallud, J. A., Tesoriero, R. 2020. "Code generation using model driven architecture: A systematic mapping study". Journal of Computer Languages, 56, 100935.

(Shehory and Sturm, 2014) Shehory, O., Sturm, A. 2014. "Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks". Springer, New York, USA, 331 pages.

(Silva et al., 2018) Silva, J., Barišić, A., Amaral, V., Goulao, M., Tezel, B. T., Alaca, O. F., Challenger, M., Kardas, G. 2018. "Comparing the Usability of two Multi-Agents Systems DSLs: SEA_ML++ and DSML4MAS - Study Design". In proc. 3rd International Workshop on Human Factors in Modeling (HuFaMo 2018), held in ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018), Copenhagen, Denmark, pp. 770-777.

(Sirius, 2015) Sirius Modeling Tool, https://www.eclipse.org/sirius/ (last access: October, 2020).

(Sredejovic et al., 2018) 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.

(Syriani et al., 2018) Syriani, E., Luhunu, L., Sahraoui, H. 2018. "Systematic mapping study of template-based code generation". Computer Languages, Systems & Structures, 52: 43-62.

(Tezel et al., 2016) Tezel, B. T., Challenger, M., Kardas, G. 2016. "A Metamodel for Jason BDI Agents". In proc. 5th Symposium on Languages, Applications and Technologies (SLATE 2016), Maribor, Slovenia, OpenAccess Series in Informatics, vol. 51, pp. 8:1-8:9.

(Thangarajah et al., 2005) Thangarajah, J., Padgham, L., Winikoff, M. 2005. "Prometheus Design Tool (system demonstration)". In proc. 4th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2005), Utrecht, Netherlands, pp. 127-128.

(Wautelet and Kolp, 2016) Wautelet, Y., Kolp, M. (2016) "Business and model-driven development of BDI multi-agent systems". Neurocomputing, 182: 304-321.

(Wautelet et al., 2017) Wautelet, Y., Heng, S., Kiv, S., Kolp, M. 2017. “User-story driven development of multi-agent systems: A process fragment for agile methods”. Computer Languages, Systems & Structures, 50: 159-176.

(Weiss, 2016) Weiss, G. 2016. “Multiagent Systems (Intelligent Robotics and Autonomous Agents series) 2nd edition”. The MIT Press, Cambridge, Massachusetts, USA, 920 pages.

(Zambonelli et al., 2003) Zambonelli, F., Jennings, N. R., Wooldridge, M. 2003. “Developing Multiagent Systems: The Gaia Methodology”. ACM Transactions on Software Engineering and Methodology, 12(3): 317-370.

Omer Faruk Alaca received the B.Sc. degree in computer engineering and the M.Sc. degree in information technologies from Ege University, Turkey, in 2015 and 2019, respectively. Between 2017 and 2018, he worked as a researcher (with full scholarship) in the international R&D project funded by TUBITAK, Turkey and FCT, Portugal. He is currently working as a software engineer at Kron Information Technology and Services Corporation. His research interests include software quality, model-driven engineering, multi-agent systems and domain-specific languages.

Baris Tekin Tezel received the B.Sc. and the M.Sc. degrees in statistics from Ege University, Turkey in 2013 and 2015, respectively. He also received the B.Sc. degree in mathematics from Ege University, Turkey in 2017. He is currently working toward the Ph.D. degree in information technologies with the International Computer Institute, Ege University. Since 2014, he has been working as a Research Assistant with the Department of Computer Science, Dokuz Eylul University, Turkey. His research interests include multi-agent systems, domain-specific modelling languages, fuzzy logic and soft computing.

Moharram Challenger received his Ph.D. in IT from Ege University in 2016. He also received his B.Sc. and M.Sc. degrees in software engineering from IAU-Shabestar and IAU-Arak Universities in 2001 and 2005 respectively. From 2005 to 2009, he has been a tenure-track faculty member, as a lecturer, at the computer engineering department, IAU-Shabestar University. He was also an external post-doc researcher at the IT group of the Wageningen University of Research from 2016 to 2017. In 2017 and 2018, he has been a member of the faculty as an assistant professor at Ege University. In 2019 and 2020, he worked as a post-doc researcher in the Electronics and ICT department, University of Antwerp. He is currently a research professor at the Department of Computer Science, University of Antwerp. His research interests include MDE/MBSE, MAS, and CPS and IoT Modelling.

Miguel Goulão, Ph.D., is an Assistant Professor of the Informatics Department of FCT/UNL and an associate member of the NOVA LINCS. With 25 years of experience with experimental software engineering, Miguel is currently focused on how complexity affects the usability of software development languages, namely requirements engineering and domain-specific languages. Miguel has published over 70 peer-reviewed papers and served as PC member and journal reviewer in top-ranked conferences and journals. He was a co-author of the paper receiving the best paper award in CAiSE 2014.

Vasco Amaral, IEEE senior member, received his Phd in Computer Science from the University of Mannheim, Germany. He is currently an assistant professor at the Department of Informatics at Faculdade de Ciências e Tecnologia of Universidade Nova de Lisboa and full researcher of NOVA LINCS. He has collaborated in the past with Particle Physics's experiments at DESY (project Hera-B) and CERN (project Atlas). His current research interests are on Software Engineering, Model Driven Engineering, Domain-Specific Languages and Cyber-physical Systems Modelling.

Geylani Kardas received the B.Sc. degree in computer engineering and the M.Sc. and Ph.D. degrees in information technologies from Ege University, Turkey, in 2001, 2003 and 2008, respectively. He is currently an Associate Professor with the International Computer Institute (ICI), Ege University, and the Head of Software Engineering Research Laboratory (Ege-SERLab), ICI. His research interests mainly include agent-oriented software engineering, model-driven software development, and domain-specific (modeling) languages. He has authored or co-authored over 90 peer-reviewed papers in these research areas. Dr. Kardas is an Editorial Board Member of the Journal of Computer Languages (Elsevier). He has been working as the Principle Investigator, a Researcher or a Consultant in various R&D projects funded by governments, agencies and private corporations.