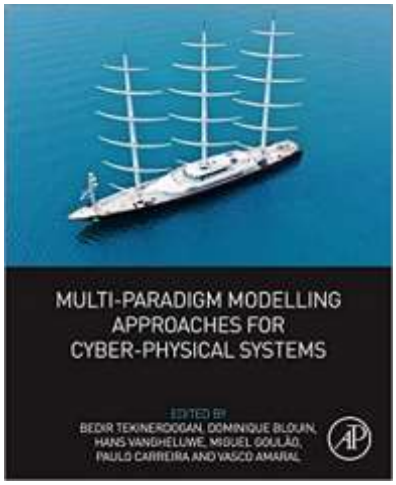
Accepted Manuscript



Agent-based cyber-physical system development with SEA_ML++

Moharram Challenger, Baris Tekin Tezel, Vasco Amaral, Miguel Goulao, Geylani Kardas

DOI: 10.1016/B978-0-12-819105-7.00013-1

To appear in: *Multi-Paradigm Modelling Approaches for Cyber-Physical Systems (1st Edition)*

Published online: 15 December 2020

Please cite this article as: Moharram Challenger, Baris Tekin Tezel, Vasco Amaral, Miguel Goulao, Geylani Kardas, Agent-based cyber-physical system development with SEA_ML++, Multi-Paradigm Modelling Approaches for Cyber-Physical Systems (1st Edition), Tekinerdogan et al. (Eds.), doi: 10.1016/B978-0-12-819105-7.00013-1

Agent-based Cyber-physical System Development with SEA_ML++

Moharram Challenger[a,b,*], Baris Tekin Tezel[b,c], Vasco Amaral[d], Miguel Goulao[d], Geylani Kardas[b]

[a]*University of Antwerp and Flanders Make, Belgium*
[b]*International Computer Institute, Ege University, Izmir, Turkey*
[c]*Department of Computer Science, Dokuz Eylul University, Izmir, Turkey*
[d]*Software Systems Group, NOVA LINCS, Universidade NOVA de Lisboa, Portugal*

Abstract

Intelligent agents are software components that can work autonomously and proactively to solve the problems collaboratively. To this end, they can behave in a cooperative manner and collaborate with other agents constituting systems called Multi-agent Systems (MAS). These systems have different perspectives such as the internal structure, plan, interaction, organization, role, environment and so on. By having these views, MASs can consider the structure, behaviour, interaction, and environment of the complex systems such as Cyber-physical Systems (CPS). Therefore, intelligent software agents and MASs can be used in the modeling and development of CPSs.

There are different Domain-specific Modeling Languages (DSMLs) to build MASs with a focus on various MAS aspects. One of the generative MAS DSMLs is SEA_ML++ which presents a thorough Model-driven Engineering practice with including the abstract syntax, graphical concrete syntax, model to model transformations and model to code transformations with the support of Platform Independent and Platform Specific levels of MAS modeling. In this chapter, we discuss how SEA_ML++ is used for the design and implementation of agent-based CPSs. An MDE methodology is introduced in which SEA_ML++ can be used to design agent-based CPS and implement these systems on various agent execution platforms. As the evaluating case study, the development of a multi-agent garbage collection CPS is taken into consideration. The conducted study demonstrates how this CPS can be designed according to the various viewpoints of SEA_ML++ and then implemented on JASON agent execution platform.

Keywords:
Multi-agent System, Intelligent Agent, Cyber-physical Systems, SEA_ML++

* Corresponding author: Moharram Challenger (m.challenger@gmail.com,+32 3 265 1731)
Email addresses: moharram.challenger@uantwerpen.be (Moharram Challenger), baris.tezel@deu.edu.tr (Baris Tekin Tezel), vasco.amaral@fct.unl.pt (Vasco Amaral), mgoul@fct.unl.pt (Miguel Goulao), geylani.kardas@ege.edu.tr (Geylani Kardas)

1. Introduction

According to Russell and Norvig [1], an agent is anything that can be considered to be able to perceive its environment through sensors and act on this environment through actuators. Moreover, the agents are located in a certain environment and are capable of flexible autonomous actions within this environment in order to meet its design objectives [2]. These autonomous, reactive, and proactive agents also have social ability and can interact with other agents and humans in order to solve their own problems. They may also behave in a cooperative manner and collaborate with other agents for solving common problems.

In order to perform their tasks and interact with each other, intelligent agents constitute systems called Multi-agent Systems (MASs). A MAS is a loosely coupled network of problem-solving entities (agents) that work together to find answers to problems that are beyond the individual capabilities or knowledge of each entity (agent).

Agents and MASs with their capabilities such as mobility, intelligence, distributedness, autonomy and dynamicity, can be utilized in very different applications, ranging from software intensive applications such as e-bartering [3, 4] and the stock exchange system [5], to the system level applications such as smart waste collection [6]. MASs can also be used along with other paradigms such as Model-based System Engineering (MBSE) to target the challenges of Cyber-physical Systems (CPS) including resource limitation, uncertainty, and distributedness. Hence, the fundamental components of CPS for various business domains can be designed and built as autonomous agents interacting with each other. MBSE is here applied to leverage the abstraction level to minimize the system complexity and facilitate the agent design. Finally, it provides a convenient way to implement and execute these systems for various MAS execution platforms [7, 8, 9].

This chapter discusses how agents and MAS can be used in both the modelling and the development of CPS. To this end, we demonstrate how using a Domain-specific Modeling Language (DSML), called SEA_ML++ enables the model-driven engineering of agents, their planning mechanisms and agent collaborations which leads to the construction of the desired CPS. SEA_ML++ can represent different aspects of MASs such as environment, interaction, agent internal, organization, plan, and role [10]. In this way, it can specify various aspects of a complex and dynamic system such as CPS. To demonstrate the proposed development methodology, a case study called multi-agent garbage collection is designed and implemented based on SEA_ML++.

This chapter is organized as follows: Section 2 and 3 discuss the background and related work respectively. In Section 4, SEA_ML++ is introduced. The methodology for modeling and development of agent-based CPS is discussed in Section 5. A multi-agent garbage collection system is designed and developed using SEA_ML++ in Section 6 to demonstrate the proposed methodology. Finally, the chapter is concluded in Section 7.

2. Background

Cyber-Physical Systems (CPS) consist of tightly integrated and coordinated computational and physical elements [11]. They represent an evolution of embedded systems to a higher level of complexity by focusing on the interaction with highly uncertain environments (such as human interaction or wear & tear of devices). In these systems, embedded computers and networks monitor (through sensors) and control (through actuators) the physical processes, usually with feedback loops where physical processes and computations affect each other. The computational part of these systems plays a key role and needs to be developed in a way that can handle (mostly in real-time) the uncertain situations with the limited resources (including computational resource, memory resource, communication resource, and so on) [12]. However, considering both the heterogeneity of the components and the variety of system behaviour interacting with physical environment, the design and the development of these systems are complex, time-consuming and costly tasks.

Generally, one of the approaches to address the complexity of engineering systems is to exclude the extra details and have an abstract model/representation of the system where we can do some tasks (e.g. analyze, comprehend, develop, and so on) which are difficult or sometimes impossible to do on the original system [13]. Modelling a system represents the properties of interest in that system which can be used for various purposes. There can be different models (with specific paradigms or formalisms) for a complex system such as a CPS in which each of the models represents one aspect of the system. This approach is called multi-paradigm modeling [14]. The modelling approach can be used for different purposes, such as model-driven engineering which is a software and systems development paradigm that emphasizes the application of modeling principles and best practices throughout the System Development Life Cycle (SDLC).

Within an MDE approach, a Domain-specific Modelling Language (DSML) uses the notations and constructs tailored towards a particular application domain (e.g. Multi-agent Systems or Concurrent Programs[15]). The end-users of DSMLs have knowledge from the observed problem domain, but they usually have little programming experience. DSMLs raise the abstraction level, expressiveness, and ease of use. The main artifacts of DSML are models instead of software codes and they are usually specified in a visual manner. A DSML's graphical syntax offers benefits, like easier design, when modeling within certain domains (e.g. IoT domain [16]).

The development of DSML is usually driven by language model definition. That is, concepts and abstractions from the domain which need to be defined in order to reflect the target domain (language model). Then, relations between the language concepts need to be defined. Both of them constitute an abstract syntax of a modeling language. Usually, a language model is defined with a metamodel. The additional parts of a language model are constraints that define those semantics which cannot be defined using only the metamodel. Domain abstractions and relations need to be presented within a concrete syntax and serve as a modeling block within the end-users modeling environment.

This modeling environment can be generated automatically if dedicated software is used, otherwise, modeling editor must be provided manually. Then, the model transformations need to be defined in order to call the domain framework, which is a platform that provides the functions for implementing the semantics of DSMLs within a specific environment. Usually, the semantics is given by translational semantics.

3. Related Work

This section discusses the related work considering both the MAS DSMLs and Agent-based CPS development studies.

Agent-DSL [17] is used for modelling agent features, like knowledge, interaction, and autonomy by presenting a metamodel. The agent modelling languages introduced e.g. in [18, 19] consider the syntax definitions rather than operational language semantics. Studies like [20, 21] also discuss MDE of agent systems by introducing a series of transformations on MAS metamodels in different abstractions. Although those transformations may guide to construct some sort of semantics, related studies describe MAS development methodologies instead of specifying a complete DSML. In addition, there exist MAS metamodel proposals (e.g. [22, 23, 24]) from which abstract syntaxes of MAS DSMLs can originate. In [25], a DSML is provided for MASs based on EMF[1]. The language supports modelling of agents according to one of the specific MAS methodologies called Prometheus. Likewise, SEA_L [26] and JADEL [27] are two agent DSLs both providing textual syntaxes based on Xtext specifications. Sredejovic et al. [28] introduce another agent DSL, called ALAS, to allow software developers to create intelligent agents having reasoning systems based on non-axiomatic logic. The work conducted in [18] aims at creating a UML-based agent modeling language, called MAS-ML, which is able to graphically model various types of agent internal architectures. However, the current version of the language does not support any code generation, which prevents the execution of modeled agent systems. DSML4BDI [9] is another DSML proposed for creating agents conforming to Belief-Desire-Intention (BDI) architecture. In addition to modeling the 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. In addition to providing an abstract syntax based on a metamodel, SEA_ML++ [10, 29] offers a full-fledged modelling language including all syntax and semantics constructs required for the MDE of agents according to well-known BDI and re-active agents principles. SEA_ML++ supports the execution of modelled agents over a series of model-to-model and model-to-code transformations enabling the construction of interactions between agents.

In Road2CPS EU support Action[2], a roadmap and recommendations for

[1] Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/
[2] http://www.road2cps.eu/

future deployment of CPS are proposed [30]. Similarly, in CPSoS EU Support Action[3], the challenges posed by engineering and operation of CPSoS are defined [31, 32] and a research and innovation agenda on CPSoS are presented. In [33], the authors report recent software engineering studies for CPSs. Also, in [34] and [35], the authors address multi-paradigm modelling aspect of cyber-physical systems. Finally, in [36], the authors present a DSL for designing CPSs. However, non of these studies address the construction of these systems using agents and multi-agent systems, e.g. to provide autonomy, reactivity and/or proactivity.

The study in [37] addresses the modeling methodology and tool for autonomous objects in CPS. The authors propose a framework called CPS-Agent to model objects with consideration of temporal-spatial traits and interaction with the physical environment. They present a role-based strategy formulation to make work patterns of CPS-Agents more clear. In terms of network communication among CPS-Agents, a set of communicative primitives is tailored based on the FIPA-ACL specification.

In [38], the authors discuss the development of an agent-based CPS for Smart Parking Systems. They believe that the inclusion of multi-agent systems, combined under the scope of CPS, ensure flexibility, modularity, adaptability and the decentralization of intelligence through autonomous, cooperative and proactive entities. They also mention that the smart parking systems can be adapted to other types of vehicles to be parked and scalable in terms of the number of parking spots and drivers/vehicles. The authors focus on how the software agents are interconnected with the physical asset controllers using proper technologies.

The authors of [39] address the challenges of MAS for CPS. They state that CPS application domains include three major characteristics (intelligence, autonomy and real-time behavior) and MAS can be used to implement such systems. They believe that MAS address the first two characteristics, but miss to comply with strict timing constraints. The main reasons for this lack of real-time satisfiability in MAS originate from current theories, standards, and technological implementations. In particular, internal agent schedulers, communication middlewares, and negotiation protocols have been identified as co-factors inhibiting the real-time compliance.

Recently in [40, 41], a group of researchers (in the scope of a European research project[4]) have studied the application of agents to deal with the problems in Cyber-physical Production Systems. They focused on quality control in manufacturing by proposing an agent architecture (which presents a distributed intelligence) for distributed analysis of the CPS and tackling the defects in multi-stage manufacturing.

Although all of the abovementioned studies present noteworthy applications of agent paradigm in CPS, they do not address the MDE of these systems in

[3] https://www.cpsos.eu/

[4] http://go0dman-project.eu/

the way of both increasing the abstraction level needed during CPS design and facilitating CPS construction with agent features. The enrichment of CPS implementations with the agent capabilities improves the execution of such systems which may also lead to the widespread use of CPS in various application domains e.g. ranging from smart manufacturing to self-adaptive systems. However, design and implementation of CPS with agent capabilities naturally become more difficult in comparison with the conventional way of CPS development since the developers may face new challenges originating especially from the autonomous and the proactive behavior of the agents built in the new CPS in addition to the already existing CPS development challenges such as the complexity and the interoperability of CPS components. MDE of agent-based CPS may contribute to minimize these challenges and hence provide a more convenient way of development. Within this context, this chapter investigates the use of a DSML enabling the MDE of CPS which is missing in the existing agent-based CPS development approaches.

4. SEA_ML++

This section elaborates the SEA_ML++ language and its components, including its abstract syntax, graphical concrete syntax, and transformations.

SEA_ML++ is an extended version of SEA_ML [42] and SEA_L [43, 26] languages by the systematic evaluation [7, 29] of agent modeling components and applying the physics of notation principles [44] to improve the graphical syntax [10] used during MAS modeling. The initial idea for creating such a DSML is first introduced in [45] and its metamodel and concrete syntax in [22].

4.1. Abstract Syntax

The abstract syntax of the SEA_ML++ is constituted by the metamodel which is divided into different viewpoints each describing a different aspect of MAS. The important viewpoints are MAS/Organization, Agent Internal, Plan, Role, Interaction, Environment, and Ontology. They were previously defined by the partial metamodels in the abstract syntax of the SEA_ML [42]. However, all these partial metamodels were improved and combined into the metamodel of the SEA_ML++.

The SEA_ML++ covers the main agent entities and their relationship which are mostly agreed by the agent research community. In addition, more specific aspects of the domain, such as Plan, Role, and Environment, are also supported in the syntax of the SEA_ML++ in detail. The general overview and relations between the viewpoints of the SEA_ML++ is given in the Figure 1.

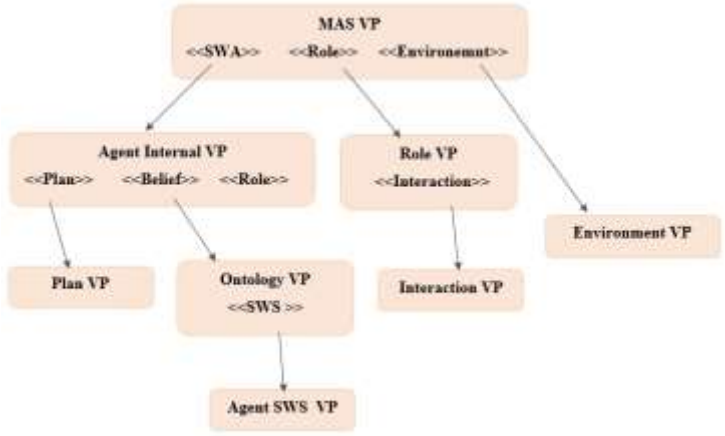


Figure 1: The general overview of the viewpoints of the SEA_ML++

All viewpoints of the SEA_ML++ are briefly discussed in the following:

MAS Viewpoint

The MAS viewpoint of the SEA_ML++ is related to the creation of a MAS as an overall aspect of the metamodel. It contains the main blocks that form the complex system as an organization.

Agent Internal Viewpoint

This viewpoint focuses on the internal structure of each agent in a MAS organization. The abstract syntax of the SEA_ML++ supports both reactive and BDI agents via this viewpoint. While meta-entities such as *Belief*, *Plan* and *Goal* support the BDI agents, *Behavior* meta-entity and its nested structure support the reactive agents.

Plan Viewpoint

The Plan viewpoint defines the internal structure of an agent's plans. When an agent implements a *Plan*, it executes its *Tasks* that consist of *Actions* which are atomic elements. *Send* and *Receive* entities are extended from Action. These types of actions are linked to a Message entity.

Role Viewpoint

Agents can play some roles, and use ontologies, and infer about the environment based on the known facts within the system. The content of roles is defined in the Role viewpoint.

Interaction Viewpoint

This viewpoint focuses on communications and interactions between agents in a MAS, and identifies entities and relationships, such as *Interaction*, *Message*, *MessageSequence*.

Environment Viewpoint

The environmental viewpoint focuses on the relationships between agents and to what they access. The Environment in which the agents are located includes all non-agent entities too such as Resources (for example, a database, a network device), Facts and Services.

Ontology Viewpoint

A MAS Organization can use ontologies for their reasoning. An ontology represents a source of information gathering and reasoning for all MAS members. All ontology sets and ontological concepts are brought together via the ontology viewpoint.

4.2. Graphical Concrete Syntax

A screenshot from the Sirius-based IDE of SEA_ML++ is given in Figure 2. Developers can visually create models of agent systems conforming to SEA_ML++ specifications by simply drag-and-dropping the required items from the palette residing at the right side of the modelling environment.

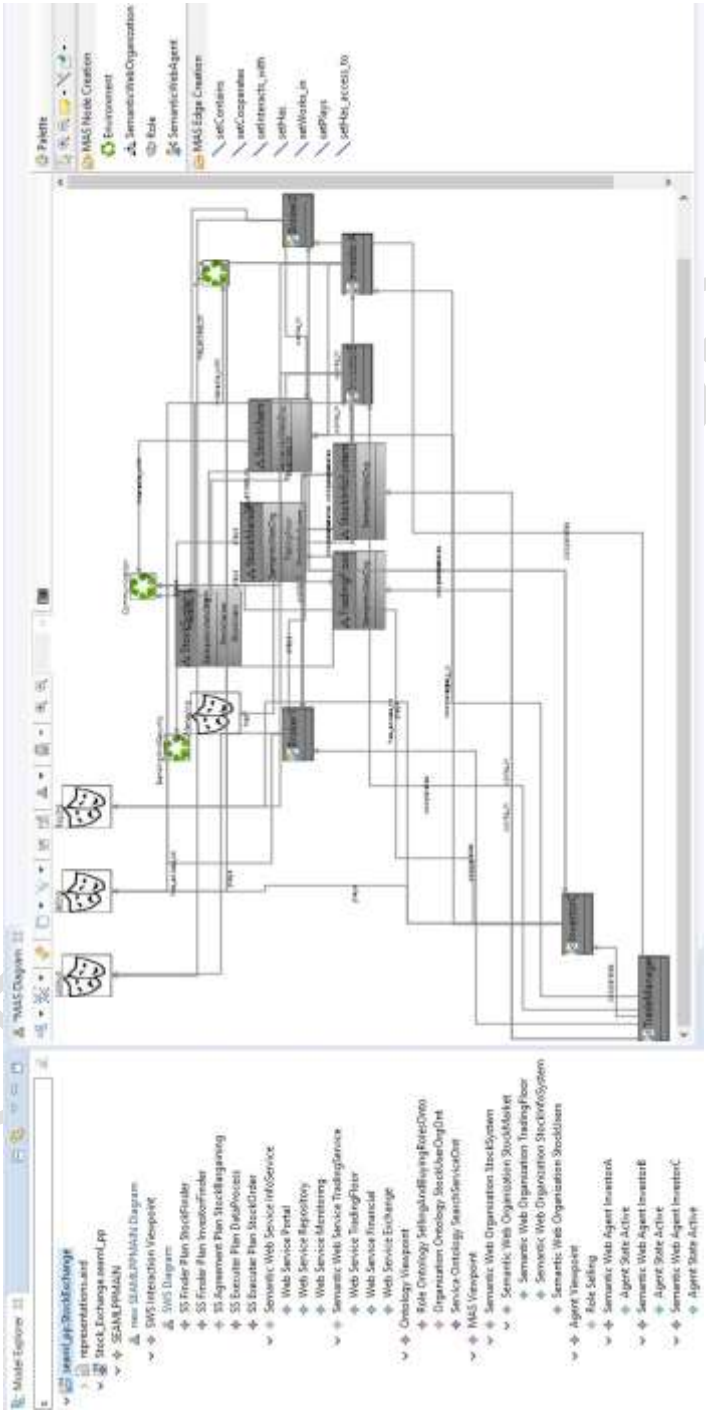


Figure 2: SEA ML++'s IDE

4.3. Transformations

It is not enough to present a DSML simply by defining concepts and representations [15, 9]. The exact definition entails the semantic of the language. The semantic of the SEA_ML++ is given as a transitional semantic that is matching the concepts of the language in terms of the other concepts already established to realize MAS. Model-to-model transformations in SEA_ML++ are provided for MAS, Agent and Interaction models, to various MAS programming languages and their implementation platforms such as JADEX[5], JACK[6], JADE[7], and Jason[8]. These agent programming languages were chosen as the target agent platform since they are among the well-known and frequently used agent platforms in MAS research and development [46]. Also, JADEX, JACK and Jason support to develop BDI agents. In this study, ATL translation language was used to provide model-to-model transformations between SEA_ML++ and above target platforms.

After the generation of platform-specific models through model transformations, a series of model-to-text transformations should be applied to generate executable software code for the modelled MAS. For this purpose, SEA_ML++ includes model-to-text transformation rules written in Acceleo to auto-generate MAS code from SEA_ML++ models.

5. Agent-based CPS Modelling and Development using SEA_ML++

In this section, a model-based methodology is proposed for the design and implementation of agent-based CPSs. To this end, the proposed methodology covers the use of the MAS DSML, SEA_ML++. In this way, a complex system such as CPS is modelled using MAS components at a higher level of abstraction. As a result, the system can be analyzed, and the required elements can be designed using the terms and notations of agent and MASs. These domain-specific elements and their relations to each other create the domain-specific instance models which pave the way to implement the system. As these models are persisted in a structural and formal way, they can be transformed to other proper paradigms, such as mathematical logics. In this way, they can be analyzed and validated based on formal methods, e.g. Satisfiability (SAT) solvers can be used to find counter-examples violating the agent model constraints (see [47] for a more extensive discussion). Furthermore, these models can be used to automatically generate the architectural code for agents and artifacts of the CPS which can end up with less syntactical errors and speed up the development procedure. Faster development also brings cost reduction in the projects. Moreover, less syntactic and semantic errors mean less iterations in the MAS development phase and short testing phase which also reduce the development cost and the

[5] https://sourceforge.net/projects/jadex/
[6] https://aosgrp.com/products/jack/
[7] https://jade.tilab.com/
[8] http://jason.sourceforge.net/wp/

effort. So, the MAS-to-be-implemented can be checked, and the errors can be partially found in the early phases of development, namely analysis and design phases, instead of finding them in the implementation and testing phases.

In this chapter, SEA_ML++ is used as a DSML for the modeling and development of agent-based CPSs. SEA_ML++ enables the developers to model the agent systems in a platform independent level and then automatically achieve code and related documents required for the execution of the modeled MAS on target MAS implementation platforms. In order to support CPS experts during MAS programming, SEA_ML++ covers all aspects of an agent system from the internal view of a single agent to the complex MAS organization. In addition to these capabilities, SEA_ML++ also supports the model-driven design and implementation of autonomous agents who can work on CPS elements.

Based on SEA_ML++, the analysis and the modelling/design of CPS can be realized using the application domains terms and notations. This helps the end users to work at a higher level of abstraction (independent of target platform) and close to the expert domain. Also, generative features of SEA_ML++ pave the way to produce the configured templates from the designed models for the software system in the underlying languages and technologies. Currently, SEA_ML++ can generate architectural code for several agent programming languages using model to model transformations of the designed platform independent instance models to the instance models of the target MAS languages. Then, these platform specific models are transformed to the platform specific codes by model to code transformations. This generation capability of SEA_ML++ can increase the development performance of the software system considerably. Finally, by constraints checking provided in SEA_ML++, the instance models are controlled considering domain-specific syntactic and semantic rules. These rules are applied in the abstract and the concrete syntaxes of the language. This feature helps to reduce the number of errors during the analysis and design of the software system and avoid postponing them to the development and the testing phases.

Although the new development methodology, introduced here, considers the adoption of SEA_ML++, it differentiates from the previous development approaches [42, 4] brought for SEA_ML as being a complete development methodology for agent-based CPSs covering the analysis, design and implementation. Analysis and design phases include two types of iterations. Also another iteration loop is considered for the implementation and maintenance of the CPS. In addition to the modification of models, the methodology also supports the changes in auto-generated codes if required. The proposed SEA_ML++ based CPS development methodology includes several steps following each other (see Figure 3): MAS based Analysis, MAS based Modeling, Model-to-Model (M2M) and Model-to-Code (M2C) (or Model-to-Text (M2T)) Transformations, and finally code completion for exact agent-based CPS implementation.

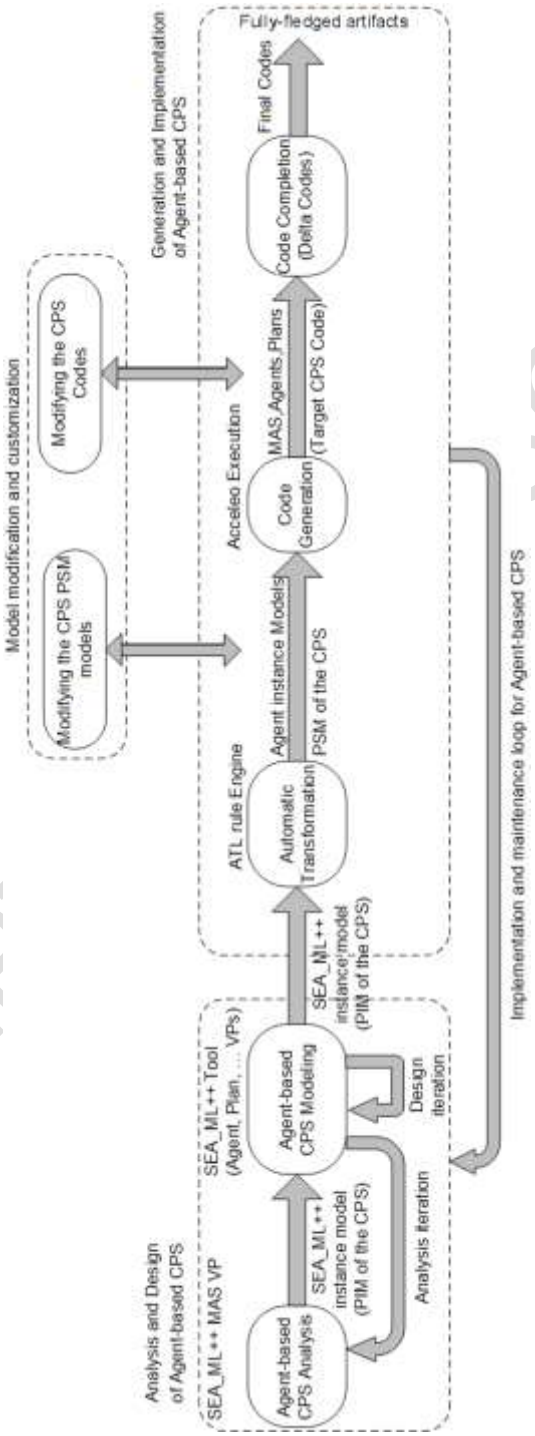


Figure 3: Agent-based CPS modeling and development using SEA ML++

Based on the proposed methodology, the development of an agent-based CPS starts with the analysis of the system by considering the MAS viewpoint of SEA_ML++ (see Figure 3). This viewpoint includes MAS elements such as organizations, environments, agents and their roles. This viewpoint provides the eagle-view of the system and shapes the high-level structure of the system. The result is a partial platform independent instance model of the system covering the analysis phase of the development and providing a preliminary sketch of the system.

In the system modeling step the CPS developer can use the fully functional graphical editors of SEA_ML++ to elaborate the design of the agents and MAS for the CPS under development, which can include 7 viewpoints of the SEA_ML++s syntax, in addition to the MAS viewpoint used in the analysis phase. These viewpoints cover all aspects of a MAS. Each viewpoint has its own palette which provides various controls leading the designers to provide more accurate models. By designing each of these models for viewpoints, additional details are added to the initial system model provided in the analysis phase. These modifications immediately are updated in the diagrams of all other viewpoints. As the other viewpoints may have some constraint checks to control some properties related to the newly added element, the developer will be directed to complete those other viewpoints to cover the errors and warnings (coming from the constraint checks). This can lead to several iterations in the design phase. The result of this phase is the development of a complete and accurate platform-independent model for the designed MAS.

The next step in the agent-based CPS development methodology using SEA_ML++ is the automatic model transformations. The models created in the previous step need to be transformed from platform-independent level into the platform-specific level, e.g. to the JASON models as in the case study of this chapter. These transformations are called M2M transformations.

According to OMG's well-known Model-driven Architecture (MDA)[9], SEA_ML++ metamodel can be considered as a Platform Independent Metamodel (PIMM) and JASON metamodel can be considered as a Platform-specific Metamodel (PSMM). The model transformations between this PIMM and PSMM lead to the implementation of of the agent-based CPSs on JASON agent execution platform. These transformations are implemented using ATL Language to produce the intermediate models which enable the generation of the architectural code for the agents and their artifacts. A CPS developer does not need to know both the details of these transformations written in ATL and the underlying model transformation mechanism. Following the creation of models in the previous modeling steps, the only action requested from a developer is to initiate the execution of these transformations via the interface provided by SEA_ML++'s Graphical User Interface (GUI).

Upon completion of model transformations, the developers have two options at this stage: 1) They may directly continue the development process with

[9] https://www.omg.org/mda/

code generation for the achieved platform-independent MAS models or 2) if they need, they can visually modify the achieved target models (e.g. JASON models) to elaborate or customize them, which can lead to gain more complete software codes in the next step, code generation. In either case, the outputs of this step are several system models each specific to a MAS execution platform (e.g. JASON platform).

The next step in the proposed methodology is the code generation for the MAS implementing the CPS. To this end, the developers' platform-specific models (conforming to PSMMs) are transformed into the code in the target languages. The M2T transformation rules are automatically executed on the target models and the codes are obtained for the implementation of the MAS. In SEA_ML++, it is possible to generate code for BDI agent languages such as JASON from SEA_ML++ models. Based on the initial models of the developer, the generated files and codes are also interlinked during the transformations where they are required. To support the interpretation of SEA_ML++ models, the M2T transformation rules are written in Acceleo. Acceleo is a language to convert models into text files and uses metamodel definitions (Ecore files) and instance files (in XMI format) as its inputs. More details on how mappings and model transformation rules between SEA_ML++ and the target PSMMs are realized as well as how codes are generated from PSMs can be found in [4].

As the last step, the developer needs to add his/her complementary codes, aka delta codes, to the generated architectural code to have fully functional system. However, some agent development languages, such as JACK, have their graphical editor in which the developer can edit the structure of MAS code. The generated codes achieved from the previous step can be edited and customized to add more platform specific details which helps to reach more detailed agents and artifacts. Then the delta code can be added to gain the final code.

It is important to note that, although all above mentioned steps are supported by SEA_ML++ to be done automatically, at any stage the developer may intervene in this development process if he/she wishes to elaborate or customize the generated agents and artifacts.

6. Development of a multi-agent garbage collection CPS

In this section, the design and implementation of a multi-agent garbage collection system, as an agent-based CPS is taken into consideration. The CPS will be executed on the JASON agent platform. JASON provides a platform for the development of BDI agents and MASs. It is a Java-based interpreter for an extended version of a Prolog-like logic programming language for BDI agents, called AgentSpeak.

The following subsections elaborate how the required CPS is designed and implemented according to the MDE process introduced in Sect. 5 and then demonstrate the execution of the modeled multi-agent garbage collection CPS.

6.1. System Design

In this subsection, we discuss the design of the multi-agent garbage collection CPS. System design is carried out by providing MAS models from different viewpoints of the system using SEA_ML++ language.

The agent-based garbage collection CPS consists of different types and numbers of agents, which cooperate with each other to collect garbage from the environment. The system design phase is performed by considering different viewpoints of SEA_ML++ language. MAS and Organization viewpoint provides an overview of the system which is shown in Figure 4. Considering the general structure of the system, the garbage collection CPS constitutes of two organizations, one covering the other, called *MAS_Organization* which include *GarbageFinder* and *GarbageBurner* agents, and *CollectorOrganization* where *GabageCollector* agents reside. It should be noted that the entities given in this overview can be regarded as stereotypes and there may be many instances of these entities in the real system implementation. For example, there may be many agents of the type *GarbageCollector* running on this system, and in fact they are expected to be more than one in a real implementation.

In this multi-agent CPS, a *GarbageFinder* agent handles the interaction between *GarbageCollectors*. This agent is responsible for finding available garbages in the environment and assigning proper *GarbageCollector* agents to collect them. *GarbageFinder* agent interacts with all candidate *GarbageCollector* agents to assign collecting garbage mission, and inform *GarbageBurner* agent about this assignment.

Garbage Collector agents represent the main entities of the designed CPS. This agents receive garbage locations and carry these pieces of garbage to the *GarbageBurner* agent to get rid of them.

The *GarbageBurner* agent is responsible for removing the garbage, which is brought to him.

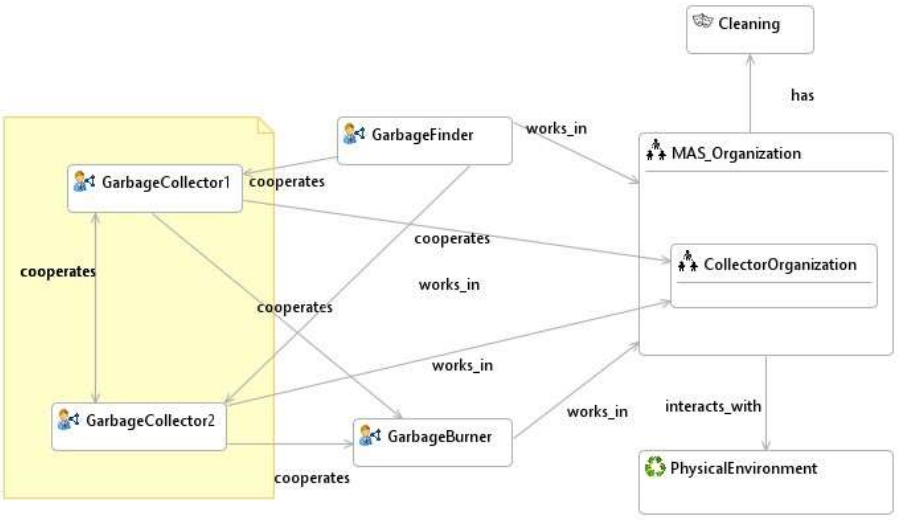


Figure 4: Overview of Multi-agent Garbage Collection CPS designed in SEA ML++ Editor

Figure 5 shows an instance model of SEA_ML++'s Agent Internal Viewpoint, which demonstrates the general internal process of the *GarbaceCollector* agent for collecting garbage.

A *GarbageCollector* agent has only one *Goal* called ("CollectTheGarbarges"). To accomplish this goal, the agent has two different *Beliefs* and four different *Plans*. Considering the *Beliefs*, the agent uses them to know the position of the garbage that it has to collect and the position of *GarbageBurner* agent, which is responsible for destroying the garbage. Thanks to the four different plans that agent uses to achieve the *Goal*, it can go to the garbage that it knows, and pick and bring it to the GarbageBurner agent to eliminate it. While carrying out these plans, it plays 3 different roles.

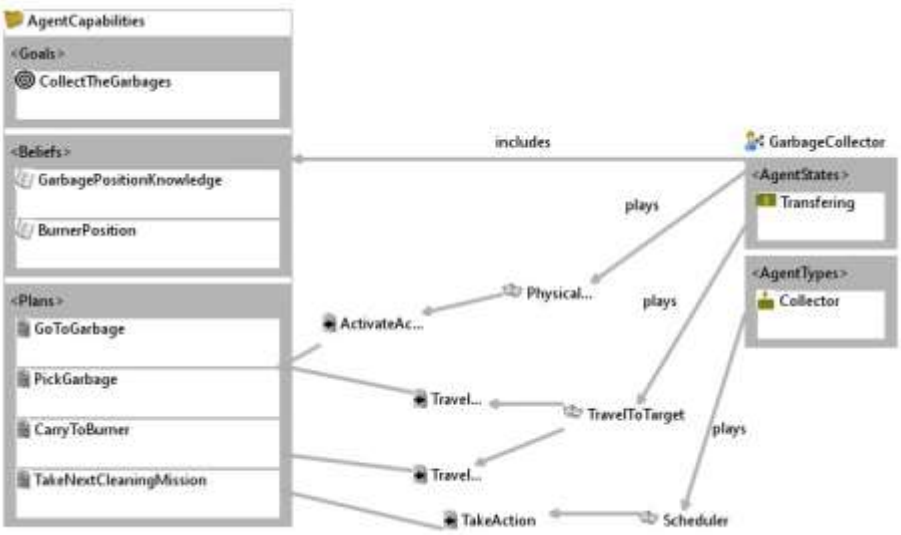


Figure 5: Agent Internal Diagram of the Garbage Collector Agent

As discussed in Sect. 5, the models provided for the design of the CPS are used in the implementation phase. To this end, the conceptual mappings between SEA_ML++ and JASON platform elements, shown in Table 1, are used. Hence, it will be possible to transform SEA_ML++ model instances to JASON platform models and then execute them inside JASON agent execution platform via a series of M2M and M2T transformations discussed in Sect. 5. It is worth indicating that these mappings and transformations are all built-in SEA_ML++ and they are used and executed behind the scene without any human intervention, e.g. CPS developers do not need to deal with them. The implementation is discussed in detail in the next subsection.

Table 1: The conceptual mappings between some of the SEA_ML++ and JASON elements

SEA_ML++ concepts	JASON Concepts
MAS/Organisation	MAS
Agent	Agent
Plan	Plan
Behaviour	Actions
agent state	Belief Base
agent type	Agent Class
Goal	Goal
Role	Goal
Belief	Belief
Fact	Literal
Environment	Environment

6.2. System Development

In this study, the proposed multi-agent garbage collection CPS is implemented using the JASON platform . JASON is selected as it is one of the widely accepted Java-based BDI MAS development platform with ongoing support.

Based on the proposed methodology, M2M transformations are applied for transforming the models designed in SEA_ML++ as platform independent models and JACK BDI agent as platform-specific models. Then, the M2T rules are applied on platform-specific models (JASON models) for the generation of ASL files, which include JASON BDI agent codes. As an example of the generated code, ("garbageCollector.asl") file is shown in Figure 6. This file consists of Prolog-like agent codes. Although the codes generated for MAS can be executed directly in the Java-based interpreter of JASON environment, some of the business logics which are case specific are missing. Therefore, additional codes are added to these generated codes, called delta codes, to have a fully functional system.

The source codes generated using SEA_ML++ models include architectural codes. The generated mechanism uses the syntactically correct templates. Also, the models are controlled by the language during the semantic control stage. This prevents many semantic errors in the code when compared with a manual development. So, it is assumed that the codes do not have any syntactic errors at this level. However, the delta codes must be added manually to create behavioral logic and they may include some syntactic or semantic errors. Manual completion of this code is required for both the MAS and CPS (Environment) parts of the system. Interested readers may find an extensive discussion on the evaluation of SEA_ML++'s code generation performance and the degree of manual code completion in [7] and [4]. As can be seen in these empirical studies, it is possible to generate more than 80% of a MAS software just by modeling in SEA_ML++.

In Figure 6, the codes given in lines 24-35 were generated and delta codes were added. They create the behavioral logic of the *takeNextCleaningMission* agent plan. When we examine the relevant code snippet, we see that the garbage collector agent searches the location of the garbage on its own belief base and then runs the plan to go to the location of the garbage. Then, garbage collector agent will execute the relevant plans respectively to pick the garbage and carry it to the garbage burner agent. Similarly, the other agent plans are generated and some delta codes are added to give the fully functional behavioral logic.

As another example, the codes in lines 12-19 of Figure 7 were generated and delta codes added, too. These codes form the behavioral logic of the nextGarbage plan. Here, the position of a randomly generated garbage in the environment is sent both to the GarbageBurner agent and to all GarbageCollector agents in the environment.

6.3. Demonstration

To demonstrate the implemented system, this section shows the system execution consisting *GarbageColector*, *GarbageBurner* and *GarbageFinder* agents.

```
// Agent garbageCollector in project garbageWorld

/* Initial beliefs and rules */

/* Initial goals */

/* Plans */

+garbage(X,Y)[burnerLocation(X1,Y1),source(Ag)] : not .desire(takeNextCleaningMission) <-
    +burnerLoc(X1,Y1);
    .print("Garbage has been detected in ",X," and ",Y," coordinates.");
    .send(Ag, askOne, garbageAsigned(X,Y),Reply,3000);
    !makeDecision(Reply,garbage(X,Y)).


+!makeDecision(Reply,garbage(X,Y)): Reply == false <-.print("I will start my next cleaning mission.");
!takeNextCleaningMission.


-!makeDecision(Reply,garbage(X,Y)): true <- .print("Abort the cleaning mission"); .abolish(garbage(X,Y)).

+!takeNextCleaningMission
   <-
      !garbage(Xg,Yg);
      ?pos(X,Y);
      -+pos(last,X,Y);
      !goToGarbage(Xg,Yg);
      !pickGarbage(Xg,Yg);
      ?burnerLoc(Xb,Yb);
      !carryToBurner(Xb,Yb);
      .print("The cleaning task has been completed.");
      .print("Move to previous position.")
       ?pos(last,Xo,Yo);
       goTo(Xo,Yo);
       -+pos(Xo,Yo).

 +pickedGarbage(X,Y):true <- delGarbage(X,Y); pickGarbage.
 -pickedGarbage(X,Y):true <- dropGarbage.

+!goToGarbage(X,Y):garbage(X,Y)<-.print("going to the garabage in ",X," and ",Y," coordinates."); goTo(X,Y); -+pos(X,Y).
+!goToGarbage(X,Y).

+!pickGarbage(X,Y):garbage(X,Y)<-.print("I pick the garbage");
                  ?burnerLoc(X1,Y1);.abolish(garbage(X,Y));+pickedGarbage(X,Y).
+!carryToBurner(X,Y):pickedGarbage(X1,Y1)<-?burnerLoc(Xb,Yb);.print("I carry the garbage");goTo(Xb,Yb); -+pos(X,Y);
.send(garbageBurner,achieve,burnGarbage(pickedGarbage(X1,Y1))); -pickedGarbage(X1,Y1).

+?add_proposal(X): X==false.
+?add_proposal(X): not X==false <- .fail.
```

Figure 6: The Garbage Collector ASL File

In Figure 8, the user interface of the system when the system is started to run, is shown. In this interface, the user just can see the graphical representation of the agents and the garbage on a grid model.

As soon as the system executes inside the JASON platform, the *GarbageFinder* agent finds the garbage and sends its position to all other agents. In this way, *GarbageCollector* agents contact with the *GarbageFinder* agent to be assigned to the relevant garbage collection task. Then one of them, which is not busy by any other task, is assigned by *GarbageFinder* to collect the relevant garbage. The *GarbageCollector* agent, who takes the task, picks the garbage and carries it to the *GarbageBurner* agent. Thus, *GarbageBurner* agent destroys the garbage. After completing the task, the *GarbageCollector* agent returns to its pre-task position to wait for a new task. The process described above is shown in the screenshots taken from the GUI of the execution environment given in Figure 9, Figure 10 and Figure 11.

An excerpt from the console output of the whole execution of the system is given in Figure 12. As can be seen, all agents of the garbage collection CPS, which are modeled with SEA_ML++, were initialized inside JASON platform

```
// Agent garbageFinder in project garbageWord

/* Initial beliefs and rules */

/* Initial goals */

!nextGarbage.

/* Plans */

+!nextGarbage : true <-
    .random(R1); X = math.round(20*R1);
    .random(R2);  Y = math.round((20*R2));
    .print("I find a garbage located in ",X," and ",Y);
    +garbage(X,Y);
    .send(garbageBurner,askOne,pos(Xb,Yb),pos(Xb,Yb));
    .send([garbageCollector1,garbageCollector2],tell,garbage(X,Y)[burnerLocation(Xb,Yb)]);
     .wait(1000);
    !!nextGarbage.

+garbage(X,Y) : true <- addGarbage(X,Y).
-garbage(X,Y) : true <- delGarbage(X,Y).

+burnedGarbage(X,Y)[assigned(Ag),source(garbageBurner)]<--garbage(X,Y);-garbageAsigned(X,Y,_);
.print("Garbage located in ",X," and ",Y," has been cleaned by ",Ag).

+?garbageAsigned(X,Y)[source(Ag)]: not garbageAsigned(X,Y,_) <- +garbageAsigned(X,Y,Ag).
```

Figure 7: The Garbage Finder ASL File

and their plans were successfully executed to provide the required agent interactions.

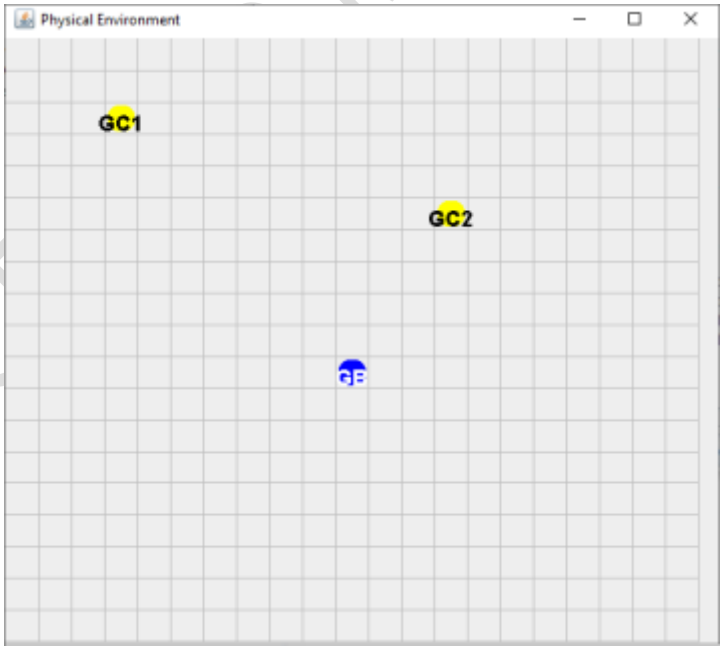


Figure 8: A screenshot of the user interface of the system

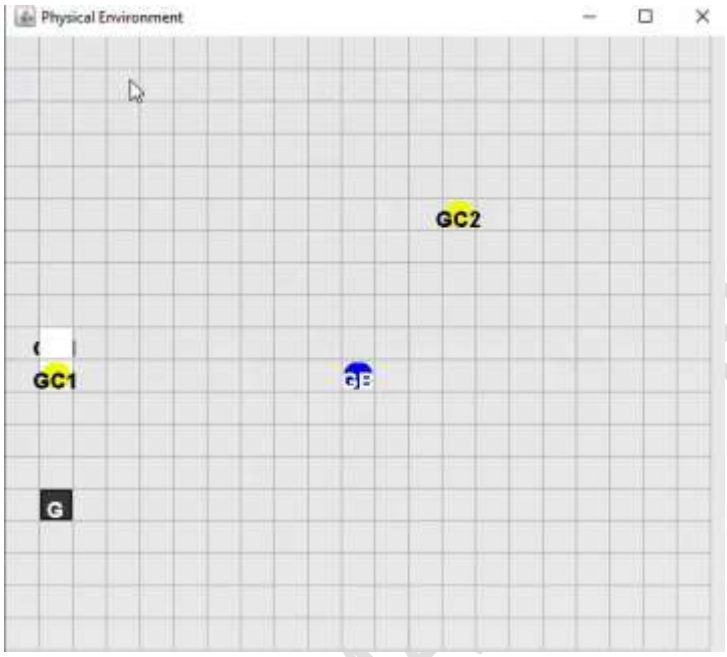


Figure 9: A garbage is found in the environment by the Garbage Finder Agent and one of the garbage collector agents goes to pick it up

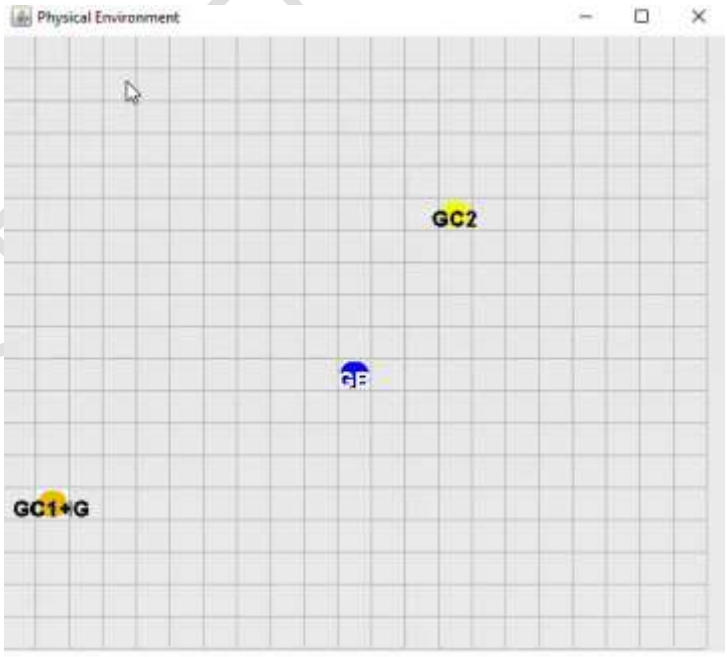


Figure 10: The Garbage Collector Agent 1 picks the garbage up

Figure 11: Garbage Collector Agent 1 brings picked garbage to the Garbage Burner Agent

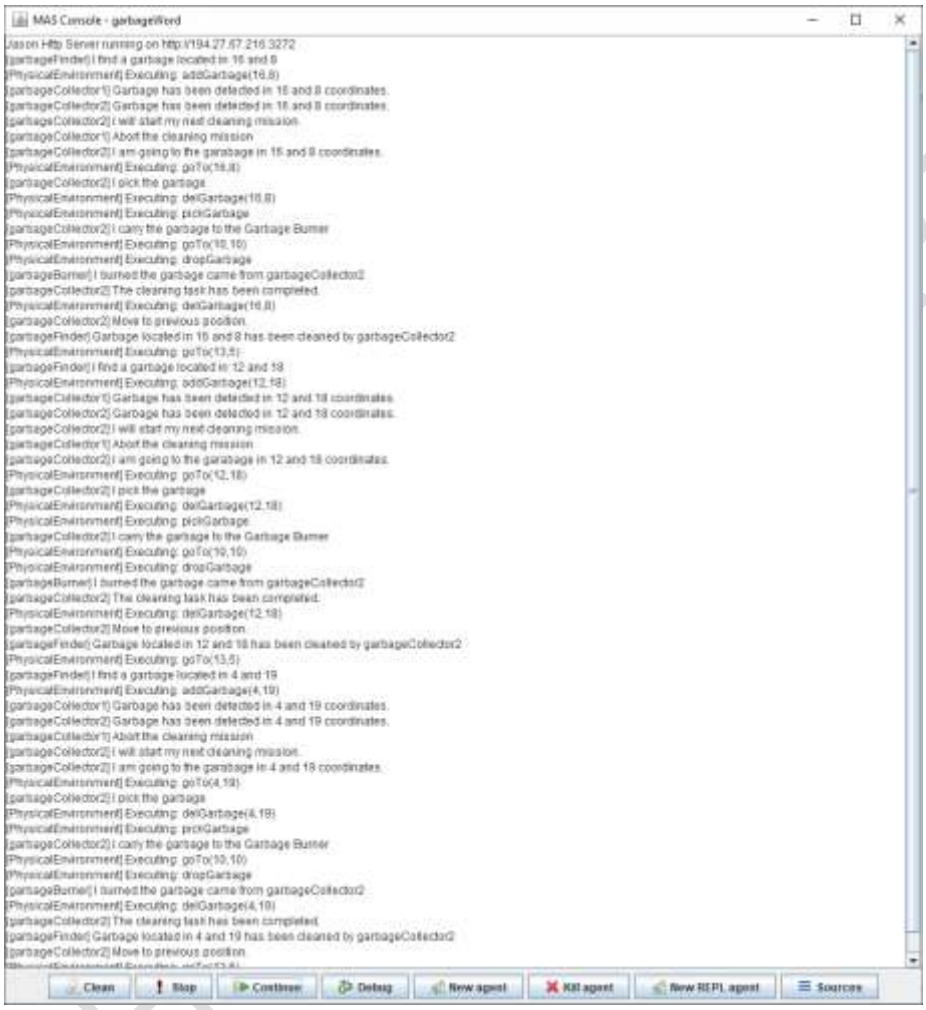


Figure 12: Screenshot of the console output showing the initialization & interaction of agents

7. Conclusion

In this chapter, we discussed the modeling and development of CPS using agents and MAS. To this end, a generative agent modeling language, called SEA_ML++ was employed. This DSML can support the CPS designer to model different aspects using various viewpoints such as MAS/Organization, Agent Internal, Plan, Interaction, Environment and so on.

In addition, we introduced an MDE methodology in which SEA ML++ can be used to design agent-based CPS and implement these systems on various agent execution platforms. To give some flavor of using this methodology, the development of a multi-agent garbage collection CPS was taken into consideration. The conducted study demonstrated how this CPS can be designed

according to the various viewpoints of SEA_ML++ and then implemented on JASON BDI agent platform.

Use of SEA_ML++ improved the way of both design and implementation of the agent-based CPS. Based on the conducted case study, one can deduce that a graphical syntax of a MAS DSML facilitates the design of a CPS which is composed of various interacting agents. It is possible to model the structure of an agent-based CPS according to different viewpoints before implementing and deploying the system. A developer can also benefit from the visual modeling environment to check the modeled system at the early phases of the design which may pave to minimize the errors during the implementation. We also showed that the translational semantics of SEA_ML++ language lead the auto-generation of the source codes required to execute the agent-based CPS. Hence, the system development process can be shortened with the application of the MDE methodology introduced in this chapter.

In the future, the platform support of SEA_ML++ can be extended with new agent execution environments in addition to JASON. Hence, SEA_ML++ models can be used to implement agent-based CPS on various platforms. However, just creating new transformations to these platforms will not be enough for this purpose. Probably both the syntax and semantics definitions of the language will also need to be updated and extended when the diversity of main CPS components including sensors, actuators, networks and other physical components are taken into consideration.

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. The authors would also like thank the European Cooperation in Science & Technology (COST) Action networking mechanisms and support of COST Action IC1404: Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS). COST is supported by the EU Framework Programme Horizon 2020.

References

[1] R. Stuart, N. Peter, Artificial intelligence-a modern approach 3rd ed (2016).

[2] M. Wooldridge, N. R. Jennings, Intelligent agents: Theory and practice, The knowledge engineering review 10 (2) (1995) 115–152.

[3] S. Demirkol, S. Getir, M. Challenger, G. Kardas, Development of an agent based e-barter system, in: 2011 International Symposium on Innovations in Intelligent Systems and Applications, IEEE, 2011, pp. 193–198.

[4] M. Challenger, B. T. Tezel, O. F. Alaca, B. Tekinerdogan, G. Kardas, Development of semantic web-enabled bdi multi-agent systems using sea_ml: An electronic bartering case study, Applied Sciences 8 (5) (2018) 688.

[5] G. Kardas, M. Challenger, S. Yildirim, A. Yamuc, Design and implementation of a multiagent stock trading system, Software: Practice and Experience 42 (10) (2012) 1247–1273.

[6] E. D. Likotiko, D. Nyambo, J. Mwangoka, Multi-agent based iot smart waste monitoring and collection architecture, arXiv preprint arXiv:1711.03966.

[7] M. Challenger, G. Kardas, B. Tekinerdogan, A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems, Software Quality Journal 24 (3) (2016) 755–795.

[8] V. Mascardi, D. Weyns, A. Ricci, C. B. Earle, A. Casals, M. Challenger, A. Chopra, A. Ciortea, L. A. Dennis, Á. F. Díaz, et al., Engineering multi-agent systems: State of affairs and the road ahead, ACM SIGSOFT Software Engineering Notes 44 (1) (2019) 18–28.

[9] G. Kardas, B. T. Tezel, M. Challenger, Domain-specific modelling language for belief–desire–intention software agents, IET Software 12 (4) (2018) 356–364.

[10] T. Miranda, M. Challenger, B. T. Tezel, O. F. Alaca, A. Barišić, V. Amaral, M. Goulao, G. Kardas, Improving the usability of a mas dsml, in: International Workshop on Engineering Multi-Agent Systems, Lecture Notes in Artificial Intelligence, Vol. 11375, Springer, Cham, 2019, pp. 55–75.

[11] J. M. Bradley, E. M. Atkins, Optimization and control of cyber-physical vehicle systems, Sensors 15 (9) (2015) 23020–23049.

[12] R. Fujimoto, C. Bock, W. Chen, E. Page, J. H. Panchal, Research challenges in modeling and simulation for engineering complex systems, Springer, 2017.

[13] M. Brambilla, J. Cabot, M. Wimmer, Model-driven software engineering in practice, © Morgan & Claypool Publishers, 2017.

[14] M. Amrani, D. Blouin, R. Heinrich, A. Rensink, H. Vangheluwe, A. Wortmann, Towards a formal specification of multi-paradigm modelling, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, 2019, pp. 419–424.

[15] E. Azadi Marand, E. Azadi Marand, M. Challenger, Dsml4cp: a domain-specific modeling language for concurrent programming, Computer Languages, Systems & Structures 44 (2015) 319–341.

[16] C. Durmaz, M. Challenger, O. Dagdeviren, G. Kardas, Modelling contiki-based iot systems, in: OASIcs-OpenAccess Series in Informatics, Vol. 56, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017, pp. 5:1–5:13.

[17] U. Kulesza, A. Garcia, C. Lucena, P. Alencar, A generative approach for multi-agent system development, in: International Workshop on Software Engineering for Large-Scale Multi-agent Systems, 2004, pp. 52–69.

[18] E. J. T. Gonçalves, M. I. Cortês, G. A. L. Campos, Y. S. Lopes, E. S. Freire, V. T. da Silva, K. S. F. de Oliveira, M. A. de Oliveira, Mas-ml 2.0: Supporting the modelling of multi-agent systems with different agent architectures, Journal of Systems and Software 108 (2015) 77–109.

[19] B. T. Tezel, M. Challenger, G. Kardas, A metamodel for jason bdi agents, in: 5th Symposium on Languages, Applications and Technologies (SLATE'16), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016, pp. 1–9.

[20] C. Hahn, C. Madrigal-Mora, K. Fischer, A platform-independent metamodel for multiagent systems, Autonomous Agents and Multi-Agent Systems 18 (2) (2009) 239–266.

[21] G. Kardas, A. Goknil, O. Dikenelli, N. Y. Topaloglu, Model driven development of semantic web enabled multi-agent systems, International Journal of Cooperative Information Systems 18 (02) (2009) 261–308.

[22] M. Challenger, S. Getir, S. Demirkol, G. Kardas, A domain specific metamodel for semantic web enabled multi-agent systems, in: International Conference on Advanced Information Systems Engineering, Springer, Berlin, Heidelberg, 2011, pp. 177–186.

[23] I. Garcia-Magarino, Towards the integration of the agent-oriented modeling diversity with a powertype-based language, Computer Standards & Interfaces 36 (6) (2014) 941–952.

[24] G. Beydoun, G. Low, B. Henderson-Sellers, H. Mouratidis, J. J. Gomez-Sanz, J. Pavon, C. Gonzalez-Perez, Faml: a generic metamodel for mas development, IEEE Transactions on Software Engineering 35 (6) (2009) 841–863.

[25] J. M. Gascueña, E. Navarro, A. Fernández-Caballero, Model-driven engineering techniques for the development of multi-agent systems, Engineering Applications of Artificial Intelligence 25 (1) (2012) 159–173.

[26] S. Demirkol, M. Challenger, S. Getir, T. Kosar, G. Kardas, M. Mernik, A dsl for the development of software agents working within a semantic web environment, Computer Science and Information Systems 10 (4) (2013) 1525–1556.

[27] F. Bergenti, E. Iotti, S. Monica, A. Poggi, Agent-oriented model-driven development for jade with the jadel programming language, Computer Languages, Systems & Structures 50 (2017) 142–158.

[28] D. Sredojević, M. Vidaković, M. Ivanović, Alas: agent-oriented domain-specific language for the development of intelligent distributed non-axiomatic reasoning agents, Enterprise Information Systems 12 (8-9) (2018) 1058–1082.

[29] J. Silva, A. Barišic, V. Amaral, M. Goulão, B. T. Tezel, O. F. Alaca, M. Challenger, G. Kardas, Comparing the usability of two multi-agents systems dsls: Sea ml++ and dsml4mas study design, in: 3rd International Workshop on Human Factors in Modeling (HuFaMo 18) hold under ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS), 2018, pp. 1–8.

[30] M. Reimann, C. Ruckriegel, S. Mortimer, S. Bageritz, M. Henshaw, C. E. Siemieniuch, M. A. Sinclair, P. J. Palmer, J. Fitzgerald, C. Ingram, et al., Road2CPS priorities and recommendations for research and innovation in cyber-physical systems, © Steinbeis-edition, 2017.

[31] S. Engell, R. Paulen, M. A. Reniers, C. Sonntag, H. Thompson, Core research and innovation areas in cyber-physical systems of systems, in: International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems, Springer, 2015, pp. 40–55.

[32] H. Thompson, R. Paulen, M. Reniers, C. Sonntag, S. Engell, Analysis of the state-of-the-art and future challenges in cyber-physical systems of systems, EC FP7 project 611115.

[33] T. Bures, D. Weyns, B. Schmerl, J. Fitzgerald, A. Aniculaesei, C. Berger, J. Cambeiro, J. Carlson, S. A. Chowdhury, M. Daun, et al., Software engineering for smart cyber-physical systems (sescps 2018)-workshop report, ACM SIGSOFT Software Engineering Notes 44 (4) (2019) 11–13.

[34] H. Vangheluwe, Multi-paradigm modelling of cyber-physical systems, in: SEsCPS@ ICSE, 2018, p. 1.

[35] D. Morozov, M. Lezoche, H. Panetto, Multi-paradigm modelling of cyber-physical systems, IFAC-PapersOnLine 51 (11) (2018) 1385–1390.

[36] F. van den Berg, V. Garousi, B. Tekinerdogan, B. R. Haverkort, Designing cyber-physical systems with adsl: A domain-specific language and tool support, in: 2018 13th Annual Conference on System of Systems Engineering (SoSE), IEEE, 2018, pp. 225–232.

[37] Y. Hu, X. Zhou, Cps-agent oriented construction and implementation for cyber physical systems, IEEE Access 6 (2018) 57631–57642.

[38] L. Sakurada, J. Barbosa, P. Leitão, G. Alves, A. P. Borges, P. Botelho, Development of agent-based cps for smart parking systems, in: IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society, Vol. 1, IEEE, 2019, pp. 2964–2969.

[39] D. Calvaresi, M. Marinoni, A. Sturm, M. Schumacher, G. Buttazzo, The challenge of real-time multi-agent systems for enabling IoT and CPS, in: International conference on web intelligence, 2017, pp. 356–364.

[40] Jonas Queiroz. and Paulo Leito. and Jos Barbosa. and Eugnio Oliveira., Distributing Intelligence among Cloud, Fog and Edge in Industrial Cyber-physical Systems, in: Proceedings of the 16th International Conference on Informatics in Control, Automation and Robotics - Volume 1: ICINCO,, INSTICC, SciTePress, 2019, pp. 447–454.

[41] J. Queiroz, P. Leitão, J. Barbosa, E. Oliveira, Agent-based approach for decentralized data analysis in industrial cyber-physical systems, in: International Conference on Industrial Applications of Holonic and Multi-Agent Systems, Springer, 2019, pp. 130–144.

[42] M. Challenger, S. Demirkol, S. Getir, M. Mernik, G. Kardas, T. Kosar, On the use of a domain-specific modeling language in the development of multiagent systems, Engineering Applications of Artificial Intelligence 28 (2014) 111–141.

[43] S. Demirkol, M. Challenger, S. Getir, T. Kosar, G. Kardas, M. Mernik, Sea_l: a domain-specific language for semantic web enabled multi-agent systems, in: 2012 Federated Conference on Computer Science and Information Systems (FedCSIS), IEEE, 2012, pp. 1373–1380.

[44] D. Moody, The physics of notations: toward a scientific basis for constructing visual notations in software engineering, IEEE Transactions on software engineering 35 (6) (2009) 756–779.

[45] G. Kardas, Z. Demirezen, M. Challenger, Towards a dsml for semantic web enabled multi-agent systems, in: Proceedings of the International Workshop on Formalization of Modeling Languages, 2010, pp. 1–5.

[46] K. Kravari, N. Bassiliades, A survey of agent platforms, Journal of Artificial Societies and Social Simulation 18 (1) (2015) 1–18.

[47] S. Getir, M. Challenger, G. Kardas, The formal semantics of a domain-specific modeling language for semantic web enabled multi-agent systems, International Journal of Cooperative Information Systems 23 (3) (2014) 1–53.