Proceedings of the 16th Conference on Computer Science and Intelligence Systems pp. 521–531
DOI: 10.15439/2021F81
ISSN 2300-5963 ACSIS, Vol. 25

An Agent-based Cyber-Physical Production System using Lego Technology

Metehan Mustafa Yalcin*, Burak Karaduman†, Geylani Kardas‡ and Moharram Challenger§
*Department of Electric and Electronics Engineering, Ege University, Izmir, Turkey
metehanmustafayalcin@gmail.com
†Department of Computer Science, University of Antwerp and Flanders Make, Belgium
burak.karaduman@uantwerpen.be
‡International Computer Institute, Ege University, Izmir, Turkey
geylani.kardas@ege.edu.tr
§Department of Computer Science, University of Antwerp and Flanders Make, Belgium
moharram.challenger@uantwerpen.be

***Abstract*—To cope with the challenges of constructing Cyber-physical Production Systems (CPPS), many studies propose benefiting from agent systems. However, industrial processes should be mostly emulated while agent-based solutions are integrating with CPPS since it is not always possible to apply cyber-based solutions to these systems directly. The target system can be miniaturised while sustaining its functionality. Hence, in this paper, we introduce an agent-based industrial production line and discuss the system development using Lego technology while providing integration of software agents as well as focusing on low-level requirements. In this way, a CPPS is emulated while agents control the system.**

***Index Terms*—Software Agent, Multi-agent System, SPADE Agent Programming, Cyber-Physical Production System, SysML**

I. Introduction

ADVANCES in networked systems produce new paradigms and new design challenges in the embedded systems. The information processing and computation are merged with communication and control that creates Cyber-Physical Systems (CPS) [1]. This evolution expands the capabilities of embedded technology interacting with the physical world through computation, control and networked communication. In this way, medical devices, transportation vehicles, intelligent highways, robotic systems and factory automation can be instrumented and implemented considering new capabilities that are achieved by CPS. One of the specialized fields of CPS is the Cyber-physical Production Systems (CPPS) which is related to the autonomous and cooperative elements and subsystems that are connected based on the context within and across all levels of production, from processes through machines up to the production and logistics networks [2].

Smart manufacturing considers adapting the embedding software and hardware technologies to the CPS, including intelligent methodologies. It aims at increasing the efficiency in the production as well as improving the conditions in the delivery process. Moreover, it is one of the leading application domains since it can have large scale production in domestic and international marketing that can impact highly economic growth. Industry 4.0 takes a pioneering role to determine manufacturing standards of the future [3]. A highly challenge in manufacturing came forward is flexibility since there are high demands for products. It is very problematic to meet those demands because of safety and complexity that arise from frequent interactions and co-operative requirements between machines, lack of human experts, and absence of an intelligent mechanism that can reason unpredictable behaviours of the system [4]. However, the requirement for intelligence to achieve smart CPS has emerged due to the complexity of these systems and physical unpredictability.

To cope with the challenges of CPS, many studies propose benefiting from the features of multi-agent systems (MAS) (e.g. [5]–[7]). MAS are widely preferred for providing support for smartness, decentralization, autonomy, and socialization of CPS. They increase the effectiveness of CPS providing enhanced functionalities for production and automation. The software agents can decide reconfiguration of the control functions/parameters, monitor transition between processes, and observe the human errors while increasing the system/human safety. Moreover, they can detect module breakdowns, structural changes, and contradictory inputs and materials, then they plan and decide on a suitable solution. In this way, they can enhance product quality and prevent damages during critical processes.

An integration of MAS and CPS may facilitate the use of intelligent agents in various industrial applications [8]. Once agents can control the components of the CPS, the developer can focus on higher-level solutions such as implementing intelligence mechanisms [9], aggregating Big Data and creating Digital Twins [10]. However, industrial processes should be emulated while agent-based solutions are integrating with CPS to address its challenges. Because it is not always possible to apply cyber-based solutions to the operational systems and dangerous environment of the industry directly when requested. Moreover, it is a burden to prototype an actual industrial production system for development purposes. There-

fore, the target system can be miniaturised while sustaining its functionality, accuracy and goal-orientedness.

Firstly, a composable and concrete technology to mimic the industrial systems where CPSs are intensively operational is required. One of the technologies commonly used for imitating such systems is Lego (e.g. [11], [12]). Although Lego technology can be supported with tools or languages such as *Scratch* [13] for programming its hardware devices to control motors and collect data from sensors, it is not possible to integrate software agents and any intelligence mechanism easily. Secondly, a common development environment and language is required to merge Lego technology and agent software. Lastly, the integration should be seamless and built from scratch, and the system should behave as it is developed by the *Scratch* graphical programming language. Hence, in this study, we introduce an agent-based industrial production line and discuss the design and implementation of this system using Lego technology while providing the integration of the software agents both to address the abovementioned CPS problems and to focus on low-level requirements.

Since CPPS use different controller/computation parts, their relationship should be modelled to reduce their development and design complexity [14]. For this purpose, the analysis and design of both the software and system parts are realized using SysML [15] in our study. Physical implementation is done using Lego technology and Raspberry Pi (with PiStorms hat) while embedded and agent software is coded using Python and Smart Python Agent Development Environment (SPADE) [16], respectively, including RasberryPI-Lego library [17]. This paper discusses all these parts of the system development. In addition, the challenges during the integration of Lego technology and software agents to create a CPPS are discussed, and lessons learned are also given in the paper.

The rest of this paper is organized as follows: Section 2 briefly discusses the related work. Section 3 gives the analysis and design of the smart production line system. The implementation of the software components and the setup of the hardware are all discussed in Section 4. The challenges we faced and lessons learned are reported in Section 5. Finally, the paper is concluded and the future plan is described in Section 6.

II. Related Work

Multi-agent systems are broadly researched and developed for providing modularization of the dynamic systems, decentralization for distributed systems [18], autonomy for production, and re-usability for further development of physical systems [19]. However, before realising such complex operations, agent integration has to be provided [20]. Once agents are implemented into CPS, their control over embedded functions should also be ensured.

In [7], capabilities of agents and CPS challenges are matched while underlining the software agents are generally a good fit for the requirements of the next generation CPS. Therefore, agents can show paramount effects for creating collaboration and integrity when they are distributed, providing smart decisions when physical unpredictability exists during the operation of CPPS. Leitao et al. [7] also emphasize that agents are good at reasoning e.g. using machine learning techniques, providing sustainability and managing human interaction in CPS.

The study in [21] addresses joint characteristics of Industrial Internet-of-Things (IIoT) and CPS while it also provides methodologies about the applicability of IoT-enabled solutions to CPPS considering interoperability principles. Additionally, they also present modelling approaches for IIoT systems.

In [22], the association between CPS and Embedded systems is considered. It is suggested to use a micro-controller board with various communication interfaces such as CAN, UART, WLAN, Ethernet, and BLE. In this way, this micro-controller can provide system-level compatibility with various boards and technological diversity to extend the design space.

Lee [23] discusses the design challenges of CPS in general from various perspectives and proposes a model-based design as a complementary approach. Hence, the process of rewriting the CPS software every time for each system can be shortened or even eliminated.

Similarly, the application of zero defect manufacturing using software agents is studied in [24] to cope with the challenges of CPS in the smart manufacturing domain. The researchers create a four-layer architecture and benefit from IoT solutions to inter-operate it with CPS using an edge-fog-cloud methodology. They highly consider earlier detection of anomalies, product quality and data correlation to find the optimal solution without interfering with any control functions.

In [25], an agent-oriented system is proposed for an Automated Guided Vehicle (AGV) with the on-board camera. Xing et al. [25] benefit from the MAS paradigm to provide an effective organisation and communication between system components. They indicate that the MAS paradigm improves the intelligence of the systems by providing an onboard solution while achieving context-awareness for an autonomous AGV.

Queiroz et al. [26] discuss the cognitive requirement of CPS, exhibit the necessity of the distributed intelligence, and envision the usefulness of MAS as they fit the CPS. They indicate that autonomous decisions in a decentralised way can address some of the CPS challenges.

In [27], an ontological classification of CPS is made considering past, present, and future CPS technologies emphasising the requirement of intelligence. Moreover, intelligence level and self-* features of CPS are matched considering both the previous achievements and future projections. The study also focuses on the current research gaps in this domain.

In [6], the authors suggest using an agent development platform, called Tartarus, to implement both cyber-physical and IoT systems. They use a solution to run the software agent on the Intel Galileo and RaspberryPI boards using the Tartarus-Lego Mindstorms NXT robots programming interface. Although the current study also supports our vision to achieve agent-CPS integration using Lego development components, our solution differentiates in the sense that we

focus more on integrating agent behaviours with the low-level of embedded control of the system components. This refers to low-level problems of agent-CPS integration from the bare-metal embedded libraries to binding them with agent-based programming.

Petrovska et al. [28] propose a domain-independent approach for knowledge aggregation and reasoning of decentralized monitoring in multi-agent smart CPS. According to their logic algorithm, they tackle the uncertainty of partial, faulty and potentially conflicting context observations. Their approach allows capturing uncertainty at run-time on a local level while providing a global decision-making mechanism. They evaluate their approach using multiple rooms cleaning robots implementing MAPE-K feedback loop to their multi-robot system.

The study in [5] discusses how a domain-specific modeling language, called SEA_ML++ and its tool [29], [30] are used for the design and implementation of a cyber-physical garbage collection system. The system is first modelled according to SEA_ML++'s graphical concrete syntax. Then a significant portion of the agent-based implementation of the system is automatically generated from these models via a series of model-to-code transformations.

In [31], the use of agents on Raspberry Pi is introduced. The study mostly focuses on the networking of agents and the cyber part of their location-aware and tracking services to establish an indoor person tracking system.

The survey in [32] considers the state of the art of applying agent technologies into the industry. The authors indicate that the industrial systems should be coupled with software logic and software agents to design CPS. They also underline the integration of software agents with physical hardware is both a difficult and a long-term process, and hence the common software patterns and paradigms can be applied to construct industrial agents which control the industrial machines and devices. However, according to their results, there is no uniform way to integrate the software agents to the low-level automation functions to create the industrial agents. Our methodology, which will be discussed in the following sections of this paper, may provide a strong alternative on facilitating the related integration within this perspective, specifically by emulating the industrial system before the real implementation and benefiting from both the agents and the embedded software and hardware.

Karnouskos et al. [33] classify the industrial agents according to ISO/IEC SQuaRE standards [34] under 8 categories, namely *Usability*, *Compatibility*, *Performance Efficiency*, *Functional Suitability*, *Portability*, *Maintainability*, *Reliability* and *Security*. Considering these 8 categories, an industrial system can be mimicked, and these standards can be applied to test the validity of them before the developed methodologies are adapted to the actual system.

As can be seen, while most research in the literature focuses on providing intelligence, adaptiveness and awareness mechanisms for CPS using agent technologies from a higher level of view, our study contributes to these efforts by providing an underlying infrastructure to merge embedded software with agent programming as well as mimicking the system operations over Lego technology to achieve the physical emulation of the industrial-like systems before their construction. Thus, we believe that once such an infrastructure is provided, then applying high-level solutions via decision making, knowledge extraction or pattern matching as mainly considered in the current studies can become more feasible.

III. System Software Analysis & Design

In this section, we discuss the analysis and design of our smart manufacturing system using SysML. We provide a multi-agent, multi-layered, multi-process study for such manufacturing systems. At the cyber side, the scalability, reactivity, and communication are merged with the embedded software in order to control a composable, extensible and modular Lego-based physical system.

A. System Overview

During the analysis and design, an efficient, autonomous, and smart manufacturing system is aimed to emulate the industrial requirements and tasks. The different types of input products are sorted in this system and they are processed autonomously according to their features which are similar to the common functionalities in an industrial factory.

The operation of the production line starts from inputting Lego bricks into the system. Then, the system starts to deliver these bricks using conveyor belts and in the next phases, the system decides either to sort or to combine these Lego bricks according to their colours.

The system is represented by a block diagram, which is illustrated in Figure 1, to provide an overview of the design. Considering the achievement of an autonomous and a modular system, the system is designed to be working on two embedded devices which are represented as *layer 1* and *layer 2*. The essential requirement to run the whole system is the agent communication which is established between these two layers using XMPP protocol [35]. Two layers controlled with two *PiStorms* extension boards and two RaspberryPI3. The first layer controls 4 motors, 1 button, 1 ultrasonic sensor and 2 colour sensors while the second layer controls 3 motors and a limit switch.

Each software agent (shown in the photograph of the created system in Figure 2) has its own tasks and roles inside the subsystems of the production line. In the following subsections, they are discussed in detail. First of all, each agent has specific behaviours and actions to control hardware elements. These actions provide the sustainability to make the system complete its processes successfully. While four of seven agents work with cyclic behaviour, two agents have one-shot behaviour and an agent works based on a finite state machine (FSM) behaviour. To get the system and the agents ready, "Initialize" methods of all agents are triggered at first. Agents act based on their roles. The roles of the system agents are as follows: Drop agent is responsible for delivering products from system input to the Shredder agent. Shredder agent is responsible for

shredding products and delivering them to Sort agent. Sort agent should decide about the product and move it to a related process. Push agent removes the brick from the conveyor belt. Lastly, Build agent builds required products according to the current state of its FSM behaviour model. Collaboratively, all agents run and control the whole production process.

In this regard, agents execute their programmed behaviours to achieve their goals. Before they start executing their tasks, each agent awaits a message from the preceding agent. This communication system provides a proper sequence for agent executions in the system.

B. Architectural Design

We designed the system architecture using block definition diagrams. For instance, in Figure 1, hardware layers are represented with root classes, named *dev1* and *dev2*. These classes are specialized to assign specific functions for the goals of agents. These classes are created using *Singleton Pattern* to constraint the instance creation as only one instance per *PiStorms* device. We benefit from *PiStorms* library to program the device-specific features and the functions which are used by the *dev1* and *dev2* classes. Software agents control the hardware I/O ports via these singleton classes. These classes constraint the cardinality of object creation to one for each hardware element and these device objects are accessed by software agents to use device functions for I/O operations. In this way, agents control the device I/O to achieve their goals and sustain the operation of the production line.

C. Agent Communications

In a MAS, messaging is important for agents to complete tasks collaboratively. In SPADE, Agent Communication Language (ACL) messages have various parameters and commonly used ones are *type*, *receiver*, *sender*, and *content*. In our system, we use informative messages to establish organization between agents. When certain events occur in the system, agents send messages which include keywords (performatives) and lead triggering an action inside the agent receiving that message. SPADE uses the XMPP protocol to deliver messages and to ease connection creation. The sequence diagram given in Figure 3 represents the messaging between the system agents.

D. Behavioural Design

In this section, behavioural activities of the agents (emulating the product line robots) are discussed. As illustrated in Figure 4, each agent has specific behaviours and actions to control hardware elements.

Overall, while four of seven agents work with cyclic behaviour, two agents have one-shot behaviour and an agent work based on finite state machine (FSM) behaviour. These agents provide actions for the sustainability of the system. Moreover, the process transitions, controlled by the software agents of the system can be visualized as given in Figure 5.

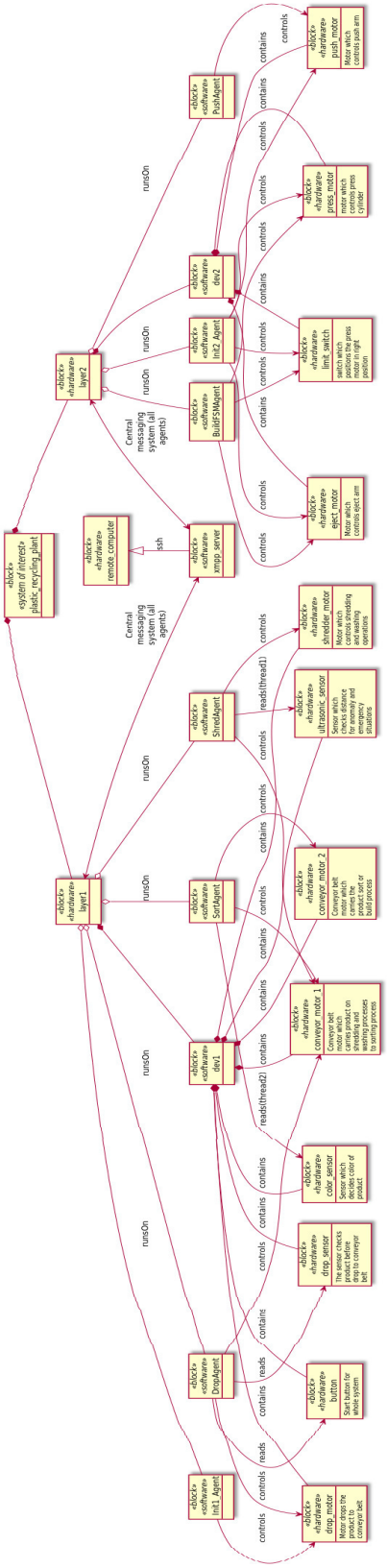

Fig. 1. Block definition diagram of the system.

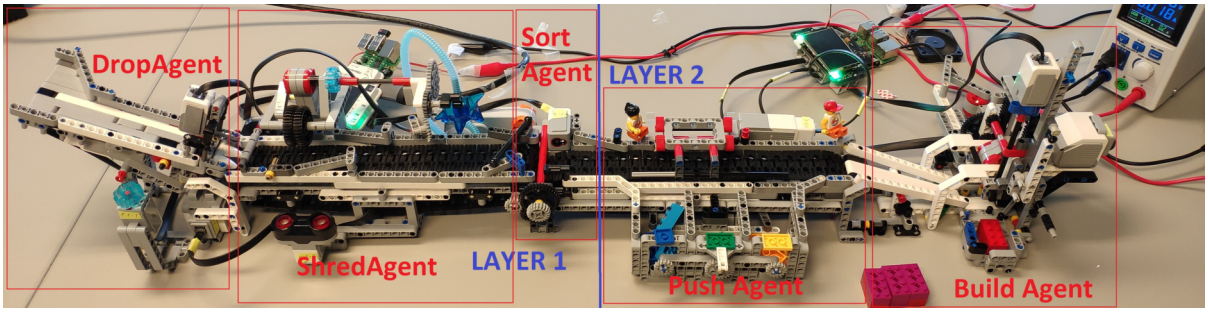

Fig. 2. Layers and agents of the Lego-based production System.

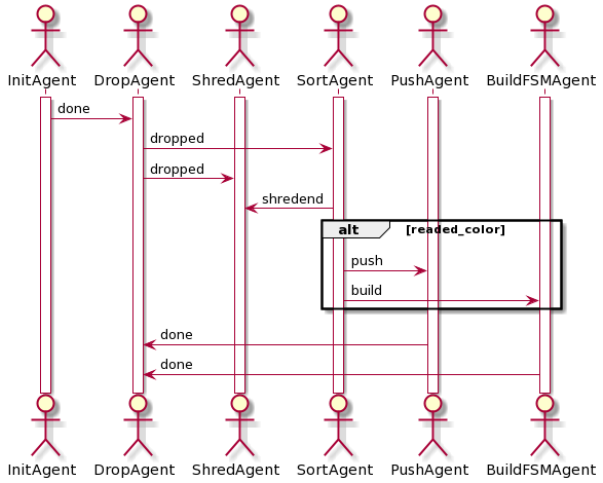

Fig. 3. Message sequence of the system agents.

1) Layer 1 Agents and their Behaviors:

Initialize Agent: Unpredictable power cuts and instant system shutdowns may cause positioning problems for the motors. When the power is cut, motors freeze at a position that is unknown by the system. Unknown motor positions cause failures on tasks. The main task of this agent is positioning the motors within mechanical limits. After motors are positioned, the agent sends a "done" message to the *Drop Agent*. This agent has a one-shot behaviour that works only once a time when the system starts up. There are 2 initializing agents for each layer. The initialize agent in *layer1* positions the drop motor with the mechanical limiter.

Drop Agent: Drop Agent is responsible for delivering the product (Lego brick) from system input to conveyor belt. It has a cyclic behaviour so it continuously samples data from 2 sensors while controls a motor. It waits for a "done" message from *Initialize Agent* or *Build Agent*, then the user presses the button to run the system continuously. The "done" message refers to the system is ready for the first run or the current process is done so that *Drop Agent* can deliver a new product to the conveyor belt.

Before *Drop Agent* runs the motor to drop a brick on the conveyor belt, it checks whether there is any brick in the input using the sensor at the input. If this condition is satisfied, then *Drop Agent* delivers the brick to the conveyor belt. It rotates the motor 90° clockwise to release the brick and then -90° anti-clockwise to return its initial position. Lastly, *Drop Agent* sends "dropped" message to *Shredder Agent* and *Sort Agent* to inform these agents about completion of its operation.

Shredder Agent: Shredder Agent is responsible for controlling shredding and washing processes. This agent has a continuous cyclic behaviour. The behaviour starts with receiving a "dropped" message from *Drop Agent* and stops when a "shredend" message is received from *Sort Agent*. While product shredding, washing and moving to the second conveyor belt, the agent concurrently checks an ultrasonic sensor with a thread. In case of any outside intervention, the system accepts this intervention as an emergency and stops the shredder motor, washing motor and conveyor belt.

Sort Agent: As represented in Figure 7, *Sort Agent* has major role for making decisions in the system. It executes a *Cyclic behavior*. After a product is dropped on the conveyor belt, *Sort Agent* starts waiting for a brick and activates the colour sensor. When the sensor realizes that the brick has arrived, it stops the conveyor belt. If the brick still does not arrive at the sensor after a certain time, the sort agent reverses the movement of the conveyor belt to set free the brick which is stuck. It reads colour sensor to recognize colour of the brick. Sensor sampling starts with receiving a "dropped" message and ends with product recognition. If the sensor recognizes product arrival to the sensor, then *Sort Agent* reads the colour of the brick and stops the conveyor belt. Then, it has 4 decision options to deliver brick and to inform related agents:

- Move brick to the bucket 1 and send "push" message to Push agent.
- Move brick to the bucket 2 and send "push" message to Push agent.
- Move brick to the bucket 3 and send "push" message to Push agent.
- Move brick to the press and send "build" message to Build agent.

2) Layer 2 Agents and their Behaviors:

Init2 Agent: Initializes the push motor, press motor and eject motor to their initial positions. This agent executes a one-shot

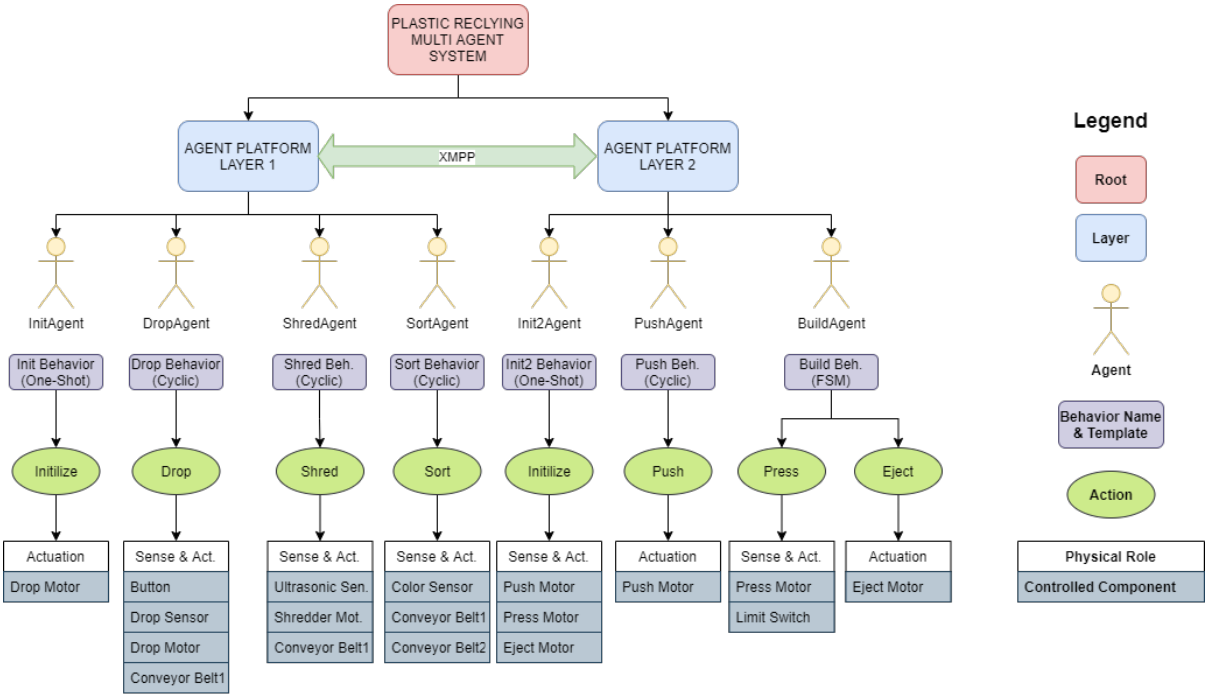

Fig. 4. Organisation diagram of the system.

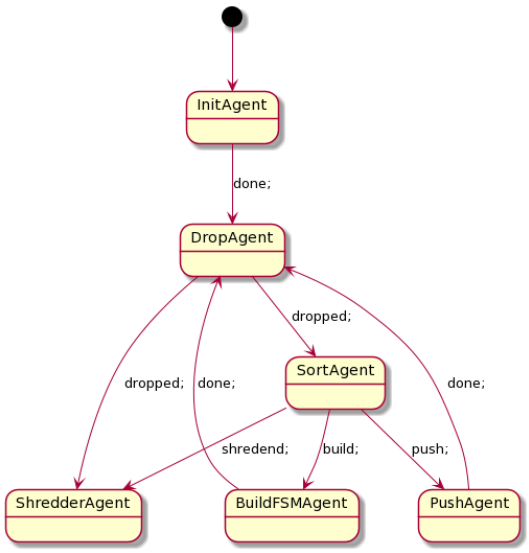

Fig. 5. Process transition between the agents.

behaviour. The agent is created with the system start up and dies after completing its behaviour and related task execution. In the Lego systems, the moving parts are usually limited with mechanical bounds. Therefore, we added an extra limit switch into this configuration for the press motor to obtain a much better initial position performance.

Push Agent: Push agent is an agent that has a cyclic behaviour. After it receives a "push" message, it pushes the mechanical line forward and then back. After, it sends the "done" message to *Drop agent* to inform the process is completed. When it receives a "push" message, *Push Agent* turns the motor clockwise with 120° and after a second, it turns counter-clockwise with 120°.

Build Agent: As Figure 6 illustrates, *Build Agent* controls the pressing process in an FSM manner. The agent starts pressing the first product after it receives the first "build" message. Then, it waits for the second "build" message which means the second product is about to arrive. When the agent receives the first "build" message, then it moves to *Press 1* state where it holds the first brick. Once it receives the second "build" message, then it switches to the *Press 2* state to combine these two bricks. After the completion of these two consecutive actions, *Build Agent* ejects the arm and pushes the products to the storage area. *Build Agent* executes its FSM behaviour continuously until the system shutdowns.

IV. Implementation of the Production Line

The software agents work in collaboration to control the heterogeneous parts of our production line which is, in fact, a complex CPS. These agents periodically sense their environment and operate to achieve their goals while keeping the system operational. Agents are self-containing entities that are able to achieve their tasks by providing local control for the different parts of the system. The role distribution to the agents are defined according to process phases to harvest the product and they are programmed to work in harmony with other agents. Their modularity and dynamic deployment also enhance the physical upgrades and changes, in other words,

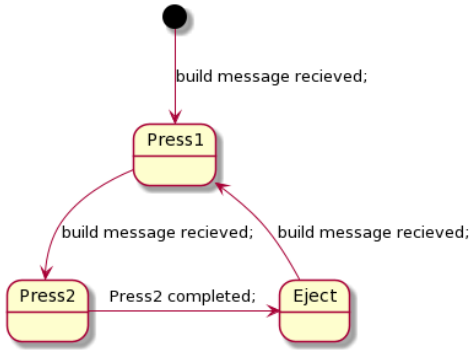

Fig. 6. State diagram of the Build agent.

new agents and new hardware can be added to the system easily.

In this section, the implementation of our smart manufacturing system is elaborated including the hardware setup and software agent implementation. The system configuration, the realization of the communication between agents, and the implementation of the corresponding behaviour classes are all discussed in the following subsections. The final structure of the implemented system has been previously shown in Figure 2.

A. System Configuration

The system is controlled by two *PiStorms* interface boards and two Raspberry Pi 3. Raspbian operating system runs Python 3.7 to interpret both embedded software and agent codes to control the system. SPADE is used for creating agents while PiStorms API is used to control Lego EV3 sensors/actuators.

In addition to the Lego production line pack, some modifications were made to resemble a more realistic industrial system. In the original system, the whole conveyor band had been controlled with only a single motor. Thanks to the modularity of Lego Technology, we separated conveyor bands to make each motor controls a separate conveyor belt so that two conveyor bands became controlled by the individual motors.

Generally, in most industrial production process implementations, limit switches are one of the most necessary hardware components to increase the reliability of the system. Hence, we added a limit switch for reducing the re-positioning error of the pressing process to zero shift. In case of any unexpected power cuts or environmental uncertainty, the system can obtain the initial position accurately using the limit switches.

Moreover, the initial version of the system had some issues about sampling colour value at the intersection point of *conveyor 1* and *conveyor2*. Sometimes there was some noise that effecting colour sampling data due to the moving parts. To fix this, we separated conveyors to find the optimal position for the colour sensor. Lastly, we added some brick parts as limiters to keep the moving bricks on the middle of the conveyors accurately.

B. Embedded Software

As discussed previously, we applied the singleton design pattern to restrict object creation from the class, including the hardware-specific I/O operations. Because the agents should access the same memory address and register so that an agent does not override other agent's access.

Inside the device-specific classes namely *dev1* and *dev2*, we also created inner classes for each hardware component. Inside these inner classes, there are functions specialized for each hardware element. For instance, an excerpt from the *ConveyorMotor* inner classes is given in Listing 1.

These inner classes can be accessed by an agent to control the hardware. Inside these inner classes, we developed a wrapper to raise the abstraction between the embedded Lego library and class implementation. In this way, wrapped code became more suitable for behavioural programming. In Listing 1, *start()*, *startSlow()*, *stop()*, *brickStucked()* and *runDegs()* functions are shown. These functions access the device-specific functions defined in the PiStorms library and wrap them to make them more usable for agent-based programming. Between lines 2 and 3, the conveyor motor is initialized and set to a certain speed. Lines 5 and 6 describe a lower speed setting for the conveyor motor while lines 8 and 9 instruct the stop function. When the system detects a stuck on the conveyor belt, it calls *brickStucked* function to reverse the conveyor belt. Lastly, lines between 14 and 16 define the *runDegs* method to rotate and run the motor according to the desired parameters.

Listing 1. ConveyorMotor inner class

```
1  class ConveyorMotor:
2      def start(self):
3          dev1.psm.BBM2.setSpeed(-100)
4          print(f'Conveyor Started')
5      def startSlow(self):
6          dev1.psm.BBM2.setSpeed(-20)
7          print(f'Conveyor Slow Started')
8      def stop(self):
9          dev1.psm.BBM2.setSpeed(0)
10         print(f'Conveyor Stopped')
11     def brickStucked(self):
12         dev1.psm.BBM2.runDegs(200, 100, True, False)
13         print(f'Brick stucked')
14     def runDegs(self,degree,speed):
15         dev1.psm.BBM2.runDegs(degree, speed, True, False)
16         motorState = dev1.psm.BBM2.isBusy()
17         print(f'Motor rotated {degree} degree on {speed}
               speed')
18         return motorState
```

It is the working principle of an agent to operate independently using behaviours and execute them in parallel with other agents. However, considering our I/O blocking situation, it is now possible to en-queue any sensor reading or motor actuating behaviours. Therefore, we need a concurrent system where it can run continuously without any interruption. Moreover, the system should sample data from the sensors while actuating a motor for 3 seconds in parallel with running another motor for 5 seconds.

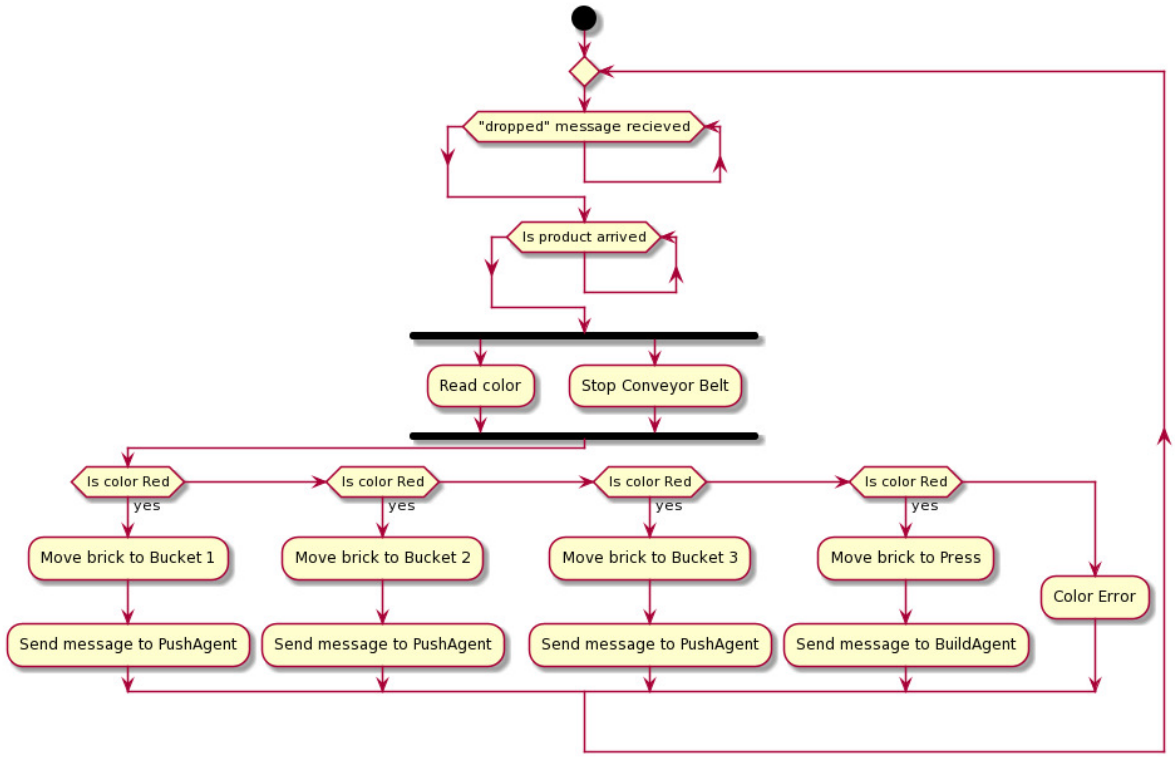

Fig. 7. Activity diagram of the Sort agent.

The obvious way to implement this concurrency is to assign a Python thread to each agent. However, there is a need for more parallelism within each agent, because an agent may also be involved in negotiations with other agents and each negotiation should proceed at its own pace while I/O blocking situations exist. In the implemented system, we used traditional threads for reading sensors and actuating motors instead of applying agent behaviours directly. Because sensor sampling is a crucial and continuous activity and agent behaviours can be blocked due to these I/O operations according to their processes considering the sampling rate. This considerably reduces the runtime-slices of each agent behaviour by blocking other operations when an agent reads the sensor inside these behaviours. As a solution, our implementation made each agent starts another thread within its setup and handles I/O operations.

Listing 2. Threading mechanism to avoid blocking I/O operations

```
1  threading.Thread(target=dev1.ColorSensor.waitBrick, args
       =(dev1.ColorSensor,))
2  async def setup(self):
3          print("SortAgent :: started")
4          b = self.SortBeh()
5          template = Template()
6          template.set_metadata("performative", "inform")
7          self.add_behaviour(b, template)
8          print("SortAgent :: running")
9          t.start()
```

In Listing 2, an excerpt from one of the created threads for the colour sensor is given. In line 1, the target method is defined. Lines between 2 and 9 describe the setup function of the agent which is also used for the initialization of the interrelated threads.

In Listing 3, a code excerpt from the LimitSwitch class which defines a limit switch is given. In the production line system, the limit switch is used to set the borders of motion of the components. For this purpose, *LimitSwitch* inner class is specialized for the limit switch hardware. In line 3, the state of the button is checked periodically, then "if/else" statement controls the state of the button. In this way, agents can detect the limit of motion when *isPressed()* function returns true and then they behave accordingly.

Listing 3. LimitSwitch innerclass

```
1  class LimitSwitch:
2      def isPressed(self):
3          touch = dev2.psm.BBS1.isTouchedEV3()
4          if touch != True:
5              return False
6          else:
7              print(f'LS Pressed')
8              return True
```

To minimize work accidents, many sensors are added to the manufacturing systems for occupational safety and health. These sensors must sense quickly as expected. For better sensor sampling rates and reactions, two threads of execution

were implemented in the responsible agents: Shredder Agent executes the thread to check emergency while Sort Agent executes the thread on checking the arrival of bricks to the colour sensor.

Colour sensors can be influenced negatively by the noises in the environment. To remove this effect, we implemented a sensor sampler inside the system. The system collects data from samples arriving from the sensor. If the last 15 samples are the same, the system accepts the colour. Otherwise, it continues to sample data (see. Listing 4).

Listing 4. Color Sensor Sampling

```
def waitBrick(self):
    readedcolorlist = [0]*15
    index = 0
    readSensor = True
    count = 0
    print("Starting to wait brick")
    dev1.retVal = 0.0
    while readSensor:
        color = dev1.psm.BBS2.colorSensorEV3()
        readedcolorlist[index] = color
        now = datetime.now()
        index = index + 1
        x=sum(readedcolorlist)/15
        if index ==15:
            index=0
            if x==2 or x==3 or x==4 or x==5 or x==6:
                print(str(index) + " -> " + "
                    ReadedColor:" + str(color))
                print("RETVAL:",str(x))
                dev1.retVal = x
            else:
                dev1.retVal = 0.0
        time.sleep(0)
```

Sample videos demonstrating how the implemented system executes the continuous production and manages a stuck event in the production line are available at https://youtu.be/dRUyXYuDPlY and https://youtu.be/_xgYyaBMv90.

V. Discussion

CPPS are expected to provide various features such as adaptiveness, awareness, intelligence, and abstraction to meet the requirements of the emerging industrial applications. Agent-based approaches can be a good alternative to support these features. However, an integration of the industrial systems with the agents is still a significant issue for the agentification of such systems as discussed in [33] and [36]. MAS is a paradigm derived from the distributed artificial intelligence field that covers distribution, decentralization, intelligence, autonomy and adaptation. Using these features, MAS provide flexibility, robustness, responsiveness and reconfigurability and create an ecosystem of intelligent, autonomous and cooperative computational entities. Despite the fact that MAS technology has already been integrated into several industrial applications such as smart production, smart power grids, smart logistics and smart healthcare, acceptance and standardisation of industrial agents is still under debate.

Seamless integration of MAS, embedded system and CPS may bring solutions to the abovementioned issues and lead to the realization of the expected features. Since CPS consists of both physical and cyber parts, top-level methodologies cannot be evaluated and shown without low-level architectures to emulate the industrial problems. There is no uniform way to integrate the software agents to the low-level automation functions to utilize them as the industrial agents [32]. Hence, the miniaturisation of the industrial systems, mimicking the process steps and reproducing the problems as described in our study can be a way to ease the burden of developing industrial agents within this context.

To achieve CPS and agent integration, device specific libraries are mostly required. Then, these libraries can be merged with agent development environments. The library can be wrapped to provide behavioral structures. Once the control of the physical components is achieved by the cyber side, the agentification process can be applied. Moreover, the integration process can be facilitated by using software engineering design principles e.g. benefiting from the design patterns. Moreover, the physical construction of the target system is still required because CPS is also a physical entity. To address this requirement, we suggested using the Lego technology, which allows the miniaturisation of the interaction between embedded systems and agents while providing extensibility for applying high-level solutions and mechanisms. When the MAS is integrated into any system, the agents inside can be distributed to the subsystems to achieve the control process distribution while establishing a network for negotiation and messaging. In this way, functionalities of the embedded devices can be encapsulated into the behaviours of the agents. Then, various behaviours can be defined for executing tasks, sending parameters, and controlling the process to achieve the system goals.

After the completion of the agentification process, the system can also be enhanced with the distributed wireless sensors for data acquisition [37], [38]. The edge, fog and cloud computing can be the key enabler technologies for CPPS considering IoT and CPS interoperability. Then, this data can be fed into the Machine Learning algorithms to achieve various computations such as pattern matching to detect system faults, prediction algorithms to avoid human errors and system-level reasoning to apply high-level plans.

During the implementation of the smart production line introduced in this paper, we followed some fundamental industrial application principles. Firstly, to keep the pressing operation calibrated, the limit switch was added to measure the elevation. While the press goes up and down, it touches the limit switch so that it operates between bounded limits. Secondly, we followed the separation of concerns principle and placed an agent for a section of the production line. In other words, only one agent is responsible for a process phase. Lastly, the same principle was also applied to conveyor belts to create *layer 1* and *layer 2*. When the task is finished in the *layer 1*, it delivers the product to the *layer 2* so that *layer 1* can receive a new task while *layer 2* processing the second step of the previous task operating as pipelined.

We believe that the constructed system based on the Lego

technology may be an appropriate tool for education considering the CPS and agent integration. Due to the fact that CPS is a multi-disciplinary field and owns multi-target domains, it is studied by a lot of researchers, engineers and practitioners. However, the recent advancements, open issues and challenges require multi-disciplinary knowledge as CPS has a wide umbrella that unites various engineering fields and disciplines. Most of the engineering and information technology courses now focus on CPS, agent-based programming and embedded technology and the requirement of autonomy and intelligence mostly becomes a must to achieve and sustain next-generation systems [39]. We need physically easy-to-construct and easy-to-modify technologies integrated with easy-to-deploy and easy-to-run programming paradigms. Lego technology provides modular and modifiable structures to meet these requirements while agent-oriented approaches present higher-level abstraction of programming. Moreover, the nature of the agents paves the way for integrating artificial intelligence, inter-operating IoT solutions, and high-level programming. As a result, multi-disciplinary studies can be taught to the future's talented engineers and students using our proposed approach.

As some technical notes, we would like to share that we faced with some challenges during the operation of the system. Due to the power requirement of RaspberryPI, Lego components and PI Storms, the system was fed with two power supplies and each power supply was feeding the system with 9.8 Volts and 3 amps. Alternatively, Li-Po batteries can also be used for short-term tests and mobility. Because the power requirements cannot be fed, then the motors fail, and the system shuts down. Moreover, if the motors get heated, then cold gels of the spray should be applied to cool down the components. To reduce the friction between Lego bricks and moving parts, we used machine oil.

Lastly, during the sensor sampling, we discovered that the colour sensor could not recognize the colour of the Lego bricks accurately due to the speed of the conveyor belt. Instead of reducing the velocity of the conveyor belt, we provided a buffered reading at the cyber part by wrapping the method into the sensor reading method and physical buffered transition by moving the colour sensor between two conveyor belts. Naturally, when Lego bricks are transferred from the first conveyor belt to the second one (*layer 1* to *layer 2*), we benefited from the natural delay caused by the friction between them. This delay and buffered reading raised accurate decisions on the colour of the Lego bricks. This decision can be supported by using pattern matching algorithms, machine learning, and/or dynamic buffer size. Because we are aware that selecting industrially standardised sensors does not guarantee ideal operation and reducing the sensor errors under harsh and corrosive conditions is another challenge [40].

VI. Conclusion

In this paper, a system to integrate software agents and CPS is proposed based on SPADE, RaspberryPI and Lego technologies. The design and implementation of this production line system are discussed. With employing agents and encapsulating embedded functions, an agent-based control on the CPPS is achieved. In this way, it is also avoided to deal with low-level details of embedded software for robot programming. Also, the distributed and mobility capabilities of software agents helped to develop heterogeneous components in the system. Our system based on the Lego technology may also assist the education activities especially considering how automation on CPPS can be supported via software agents.

As a future study, we aim to improve the current reasoning and planning capabilities of the agents in our system using belief-desire-intention (BDI) logic [41]. Additionally, we intend to provide a multi-paradigm approach, e.g. by benefiting from the IoT paradigm, so that our system both works with the same system instances (homogeneous infrastructures) and incorporate with different type systems (heterogeneous infrastructures) by establishing a network. For this purpose, both the state-of-the-art on agent-based IoT systems, as well as our past experiences, [42] will be considered. In addition, a model-based framework can also be developed to support the current development process by automatically synthesizing both agent code and embedded software [43]. To achieve this, model-driven engineering techniques similar to the ones we used in [44], [45] can be applied again to these systems to reduce the complexity.

References

[1] R. Baheti and H. Gill, "Cyber-physical systems," *The impact of control technology*, vol. 12, no. 1, pp. 161–166, 2011, doi: https://doi.org/10.1109/icmech.2019.8722929.

[2] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda, "Cyber-physical systems in manufacturing," *Cirp Annals*, vol. 65, no. 2, pp. 621–641, 2016, doi: https://doi.org/10.1016/j.cirp.2016.06.005.

[3] K.-D. Thoben, S. Wiesner, and T. Wuest, ""industrie 4.0" and smart manufacturing-a review of research issues and application examples," *International journal of automation technology*, vol. 11, no. 1, pp. 4–16, 2017, doi: https://doi.org/10.20965/ijat.2017.p0004.

[4] N.-H. Tran, H.-S. Park, Q.-V. Nguyen, and T.-D. Hoang, "Development of a smart cyber-physical manufacturing system in the industry 4.0 context," *Applied Sciences*, vol. 9, no. 16, p. 3325, 2019, doi: https://doi.org/10.3390/app9163325.

[5] M. Challenger, B. T. Tezel, V. Amaral, M. Goulao, and G. Kardas, "Agent-based cyber-physical system development with sea_ml++," in *Multi-Paradigm Modelling Approaches for Cyber-Physical Systems*, B. Tekinerdogan, V. Amaral, and H. Vangheluwe, Eds. Elsevier Pub., 2021, doi: https://doi.org/10.1016/B978-0-12-819105-7.00013-1.

[6] T. Semwal, M. Bode, V. Singh, S. S. Jha, and S. B. Nair, "Tartarus: a multi-agent platform for integrating cyber-physical systems and robots," in *Proceedings of the 2015 Conference on Advances in Robotics*, 2015, pp. 1–6.

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

[8] E. Schoofs, J. Kisaakye, B. Karaduman, and M. Challenger, "Software agent-based multi-robot development: A case study," in *2021 10th Mediterranean Conference on Embedded Computing (MECO)*. IEEE, 2021, pp. 1–8, doi: https://doi.org/10.1109/MECO52532.2021.9460210.

[9] B. Vogel-Heuser, J. Lee, and P. Leitão, "Agents enabling cyber-physical production systems," *at-Automatisierungstechnik*, vol. 63, no. 10, pp. 777–789, 2015, doi: https://doi.org/10.1515/auto-2014-1153.

[10] E. Negri, L. Fumagalli, and M. Macchi, "A review of the roles of digital twin in cps-based production systems," *Procedia Manufacturing*, vol. 11, pp. 939–948, 2017, doi: https://doi.org/10.1016/j.promfg.2017.07.198.

[11] J. Ding, Z. Li, and T. Pan, "Control system teaching and experiment using lego mindstorms nxt robot," *International Journal of Information and Education Technology*, vol. 7, no. 4, p. 309, 2017.

[12] D. Gauntlett, "The lego system as a tool for thinking, creativity, and changing the world," *Lego studies: Examining the building blocks of a transmedial phenomenon*, pp. 1–16, 2014, doi: https://doi.org/10.4324/9781315858012.

[13] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman *et al.*, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[14] F. Erata, M. Challenger, B. Tekinerdogan, A. Monceaux, E. Tüzün, and G. Kardas, "Tarski: A platform for automated analysis of dynamically configurable traceability semantics," in *Proceedings of the 32nd ACM SIGAPP Symposium on Applied Computing*, 2017, pp. 1607–1614, doi: https://doi.org/10.1145/3019612.3019747.

[15] J. Holt and S. Perry, *SysML for systems engineering*. IET, 2008, vol. 7.

[16] M. E. Gregori, J. P. Cámara, and G. A. Bada, "A jabber-based multi-agent system platform," in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, 2006, pp. 1282–1284.

[17] L. P. GitHub, "Lego PiStorms Lubrary," Available:{https://github.com/mindsensors/PiStorms}, [Online; accessed 9-May-2021].

[18] S. Demirkol, S. Getir, M. Challenger, and 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, doi: https://doi.org/10.1109/INISTA.2011.5946060.

[19] M. Merdan, M. Vallee, W. Lepuschitz, and A. Zoitl, "Monitoring and diagnostics of industrial systems using automation agents," *International journal of production research*, vol. 49, no. 5, pp. 1497–1509, 2011, doi: https://doi.org/10.1080/00207543.2010.526368.

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

[21] S. Jeschke, C. Brecher, T. Meisen, D. Özdemir, and T. Eschert, "Industrial internet of things and cyber manufacturing systems," in *Industrial internet of things*. Springer, 2017, pp. 3–19, doi: https://doi.org/10.1007/978-3-319-42559-7_1.

[22] N. Jazdi, "Cyber physical systems in the context of industry 4.0," in *2014 IEEE international conference on automation, quality and testing, robotics*. IEEE, 2014, pp. 1–4.

[23] E. A. Lee, "Cyber physical systems: Design challenges," in *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC)*. IEEE, 2008, pp. 363–369, doi: https://doi.org/10.1109/ISORC.2008.25.

[24] P. Leitão, J. Barbosa, C. A. Geraldes, and J. P. Coelho, "Multi-agent system architecture for zero defect multi-stage manufacturing," in *Service Orientation in Holonic and Multi-Agent Manufacturing*. Springer, 2018, pp. 13–26, doi: https://doi.org/10.1007/978-3-319-73751-5_2.

[25] W. Xing, Y. Jun, L. Peihuang, and T. Dunbing, "Agent-oriented embedded control system design and development of a vision-based automated guided vehicle," *International Journal of Advanced Robotic Systems*, vol. 9, no. 2, p. 37, 2012.

[26] J. Queiroz, P. Leitão, J. Barbosa, and E. Oliveira, "Distributing intelligence among cloud, fog and edge in industrial cyber-physical systems," in *16th International Conference on Informatics in Control, Automation and Robotics, ICINCO 2019*, 2019, pp. 447–454.

[27] I. Horváth, Z. Rusák, and Y. Li, "Order beyond chaos: Introducing the notion of generation to characterize the continuously evolving implementations of cyber-physical systems," in *ASME 2017 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers Digital Collection, 2017, doi: https://doi.org/10.1115/DETC2017-67082.

[28] A. Petrovska, M. Neuss, I. Gerostathopoulos, and A. Pretschner, "Run-time reasoning from uncertain observations with subjective logic in multi-agent self-adaptive cyber-physical systems," in *16th Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS*, 2021, doi: https://doi.org/10.1109/SEAMS51251.2021.00026.

[29] G. Kardas, Z. Demirezen, and M. Challenger, "Towards a dsml for semantic web enabled multi-agent systems," in *Proceedings of the International Workshop on Formalization of Modeling Languages*, ser. FML '10. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1943397.1943402

[30] M. Challenger, B. T. Tezel, O. F. Alaca, B. Tekinerdogan, and G. Kardas, "Development of semantic web-enabled bdi multi-agent systems using sea_ml: An electronic bartering case study," *Applied Sciences*, vol. 8, no. 5, 2018, doi: https://doi.org/10.3390/app8050688. [Online]. Available: https://www.mdpi.com/2076-3417/8/5/688

[31] T. Semwal and S. B. Nair, "Agpi: Agents on raspberry pi," *Electronics*, vol. 5, no. 4, p. 72, 2016.

[32] P. Leitão, S. Karnouskos, L. Ribeiro, P. Moutis, J. Barbosa, and T. I. Strasser, "Common practices for integrating industrial agents and low level automation functions," in *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2017, pp. 6665–6670, doi: https://doi.org/10.1109/IECON.2017.8217164.

[33] S. Karnouskos, P. Leitao, L. Ribeiro, and A. W. Colombo, "Industrial agents as a key enabler for realizing industrial cyber-physical systems: Multiagent systems entering industry 4.0," *IEEE Industrial Electronics Magazine*, vol. 14, no. 3, pp. 18–32, 2020, doi: https://doi.org/10.1109/MIE.2019.2962225.

[34] I. O. for Standardization, *Systems and Software Engineering: Systems and Software Quality Requirements and Evaluation (SQuaRE): Measurement of System and Software Product Quality*. ISO, 2016.

[35] A. Hornsby and R. Walsh, "From instant messaging to cloud computing, an xmpp review," in *IEEE International Symposium on Consumer Electronics (ISCE 2010)*. IEEE, 2010, pp. 1–6.

[36] L. Sakurada and P. Leitão, "Multi-agent systems to implement industry 4.0 components," in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, vol. 1. IEEE, 2020, pp. 21–26, doi: https://doi.org/10.1109/ICPS48405.2020.9274745.

[37] B. Karaduman, T. Aşıcı, M. Challenger, and R. Eslampanah, "A cloud and contiki based fire detection system using multi-hop wireless sensor networks," in *Proceedings of the Fourth International Conference on Engineering & MIS 2018*, 2018, pp. 1–5, doi: https://doi.org/10.1145/3234698.3234764.

[38] B. Karaduman, M. Challenger, and R. Eslampanah, "Contikios based library fire detection system," in *2018 5th International Conference on Electrical and Electronic Engineering (ICEEE)*, 2018, pp. 247–251, doi: https://doi.org/10.1109/ICEEE2.2018.8391340.

[39] J. Tavčar and I. Horváth, "A review of the principles of designing smart cyber-physical systems for run-time adaptation: Learned lessons and open issues," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 1, pp. 145–158, 2018, doi: https://doi.org/10.1109/TSMC.2018.2814539.

[40] K. Thiyagarajan, S. Kodagoda, L. Van Nguyen, and R. Ranasinghe, "Sensor failure detection and faulty data accommodation approach for instrumented wastewater infrastructures," *IEEE Access*, vol. 6, pp. 56 562–56 574, 2018, doi: https://doi.org/10.1109/ACCESS.2018.2872506.

[41] B. T. Tezel, M. Challenger, and 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, doi: https://doi.org/10.4230/OASIcs.SLATE.2016.8.

[42] N. Karimpour, B. Karaduman, A. Ural, M. Challenger, and O. Dagdeviren, "Iot based hand hygiene compliance monitoring," in *2019 International Symposium on Networks, Computers and Communications (ISNCC)*. IEEE, 2019, pp. 1–6, doi: https://doi.org/10.1109/ISNCC.2019.8909151.

[43] M. Challenger and H. Vangheluwe, "Towards employing abm and mas integrated with mbse for the lifecycle of scpsos," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2020, pp. 1–7, doi: https://doi.org/10.1145/3417990.3421439.

[44] B. Karaduman, M. Challenger, R. Eslampanah, J. Denil, and H. Vangheluwe, "Platform-specific modeling for riot based iot systems," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 639–646, doi: https://doi.org/10.1145/3387940.3392194.

[45] T. Z. Asici, B. Karaduman, R. Eslampanah, M. Challenger, J. Denil, and H. Vangheluwe, "Applying model driven engineering techniques to the development of contiki-based iot systems," in *2019 IEEE/ACM 1st International Workshop on Software Engineering Research & Practices for the Internet of Things (SERP4IoT)*. IEEE, 2019, pp. 25–32, doi: https://doi.org/10.1109/SERP4IoT.2019.00012.