Agents for Industry 4.0: the Case Study of a Production Cell

Matteo Baldoni[1], Cristina Baroglio[1], Valeriano Ditano[1], Roberto Micalizio[1] and Stefano Tedeschi[2,*]

[1]*Università degli Studi di Torino, Dipartimento di Informatica, Torino, Italy*
[2]*Università della Valle d'Aosta – Université de la Vallée d'Aoste, Aosta, Italy*

Abstract
In the era of Industry 4.0, where advanced technologies and interconnected systems are reshaping the landscape of manufacturing and production, the integration of intelligent automation and decision-making processes has become pivotal. One of the most promising paradigms to achieve this integration is through the use of multiagent Systems (MAS). In this paper we present a simulation environment for MAS that is based on a real-world production cell. Its aim is to provide a realistic testbed for MAS applications, demonstrating the suitability of an agent-oriented approach for the design and implementation of modern industrial systems.

Keywords
Intelligent Agents, Multiagent Systems, MAS Simulation, Industry 4.0

1. Introduction

The strong digitization process that is characterizing many aspects of today's society, from the growth of social networks to the emergence of connected wearable devices, from self-driving cars to smart cities, is also transforming the corporate world in all its facets and processes, especially in the industrial sector. This enormous transformation offers great potential for development, but at the same time it brings a whole series of new challenges that must be faced (e.g., interoperability of data and processes, complexity, autonomous decisions, etc.). Big data, smart services, Internet of Things (IoT), robotics and Artificial Intelligence (AI) are radically changing the way humans, machines and IT systems work together, leading to a strong reshaping of industrial environments. In this sense, the term *Industry 4.0* [1, 2, 3] indicates the progressive trend of industrial automation to integrate new production technologies, such as "intelligent" machines, interconnected and connected to the Internet, to improve working conditions, create new business models and increase the productivity and quality of production plants.

WOA 2023: 24th Workshop From Objects to Agents, November 6–8, Rome, Italy
*Corresponding author.
matteo.baldoni@unito.it (M. Baldoni); cristina.baroglio@unito.it (C. Baroglio); valeriano.ditano@edu.unito.it (V. Ditano); roberto.micalizio@unito.it (R. Micalizio); s.tedeschi@univda.it (S. Tedeschi)
0000-0002-9294-0408 (M. Baldoni); 0000-0002-2070-0616 (C. Baroglio); 0000-0001-9336-0651 (R. Micalizio); 0000-0002-9861-390X (S. Tedeschi)

The theme of Industry 4.0 has also recently attracted the attention of the Italian Ministry of Economic Development, leading to the definition of a National Industry 4.0 Plan consisting of a series of measures and investments aimed at fully seizing the opportunities offered by what is defined as a "fourth industrial revolution" [4]. In a highly dynamic context such as that of Industry 4.0, the enormous increase in the amount of data available to companies and the growing automation of cognitive tasks, until recently entrusted exclusively to human actors, are raising important problems related to:

- Ensuring interoperability among data, tools, and services within industrial ecosystems that are progressively more open and geographically distributed;
- Effectively managing of the vast volume of data generated by distributed production and by business processes enhanced by the IoT (i.e., "intelligent" objects interconnected in order to exchange the information they possess, collect and/or process);
- Establishing coherent and coordinated autonomous decisions and activities undertaken by heterogeneous entities (machines, robots, softwares, and humans) within decentralized, open and dynamic environments;
- Guaranteeing robustness against perturbations and abnormal events in decentralized, distributed contexts with a high degree of autonomy.

From this point of view, the technologies developed in the field of AI related to knowledge representation, automatic reasoning, autonomous agents and, in general, distributed artificial intelligence can provide promising tools to address these challenges [5, 6]. At the same time, it is of primary importance to encourage the sharing of experiences and skills between the academic world and the corporate and industrial realities present in the area.

Under this perspective, studies in the field of multiagent systems (MAS) [7, 8] have shown the effectiveness of an agent-based approach in the modeling and implementation of distributed, heterogeneous and autonomous systems. At its core, a multiagent system is a software system composed of multiple computing units, called agents, which operate and interact within a shared and potentially distributed (physical or virtual) environment. In MAS, the agents represent mutually autonomous components that use common resources and interact in order to achieve certain either common or individual objectives. Agents are capable of perceiving their environment, making decisions based on their internal state and external inputs, and executing actions to influence the environment. This context has given rise to the development of many programming paradigms and frameworks, including, just to name a few of them, JADE [9], Jason [10], ASTRA [11] and SARL [12] for agent programming and CArtAgO [13] for modeling and programming (software) environments in which agents are located and operate.

The decentralized approach to problem-solving that characterizes multiagent systems aligns well with the distributed nature of Industry 4.0 environments, where traditional centralized control methods may fall short. Indeed, the application of MAS in Industry 4.0 may bring forth several advantages. First and foremost, their decentralized nature allows for greater adaptability and robustness in the face of system disruptions. Each agent can respond independently to local changes, leading to a more efficient overall system response. Moreover, MAS excel in handling heterogeneous systems where different components might have varying capabilities and requirements. This is particularly relevant in smart factories where a diverse range of

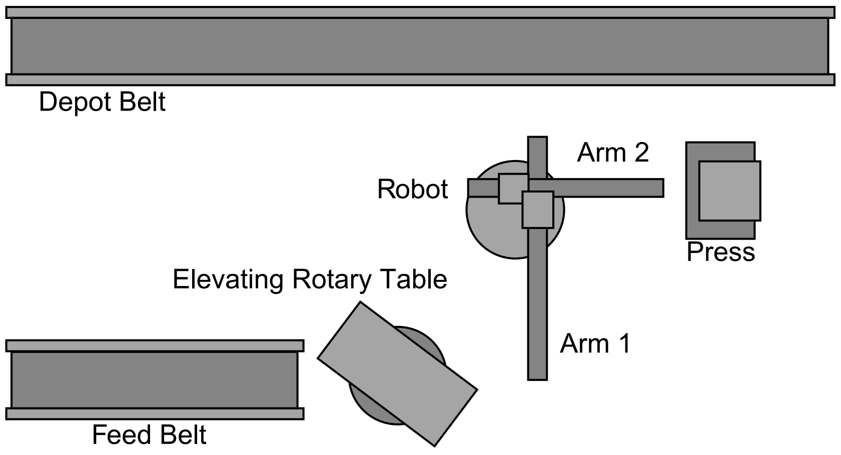

Figure 1: An industrial production cell.

machines, sensors, and devices collaborate. Additionally, MAS foster collaboration and coordination among different entities, including humans and machines. This is crucial in achieving the seamless integration of human expertise and automation, a hallmark of Industry 4.0. Agents can communicate, negotiate, and share information, enabling a holistic decision-making process that optimizes both production efficiency and resource utilization.

Surprisingly enough, however, currently available agent technologies are not extensively utilized in industrial applications at present. As a result, most proposals which are made in the field are often validated by relying on simple, narrowly scoped, toy problems. This makes it difficult to assess their applicability in case of complex, real-world use cases. In this paper we present the prototypal implementation of a simulation environment based on an existing industrial production cell for metal plates. A multiagent system designed to autonomously govern the aforementioned cell is presented as an illustration, as well. The well-known JaCaMo framework [14] has been leveraged for its development. The aim of the paper is twofold. The long term objective of this work, whose the presented implementation is a starting point, is to provide a realistic testbed for MAS applications. Secondly, we aim at promoting and demonstrating the suitability of an agent-oriented approach for the design and implementation of modern, modular, and intelligent industrial systems.

The remainder of this paper is organized as follows. Section 2 introduces the main features of the production cell adopted for the simulation. Section 3 describes the implemented simulator. Section 4 presents the MAS that has been plugged onto the simulated production cell. Conclusions end the paper.

2. Overview of the Production Cell

For the simulation purpose, we consider an industrial scenario, inspired by the well-known and widely used production cell of the KorSo project, University of Karlsruhe, 1989 [15], for manufacturing metal plates. The KorSo production cell, reported in Figure 1, is composed of five machines: two conveyor belts (a feed belt and a depot belt), an elevating rotary table, a press, and a rotary robot equipped with two extensible arms. Each device has a set of sensors that provide information about the environment and a set of actuators. The task of the cell is to get a metal plate from a storage rack via the feed belt, transform it into a forged plate by using the press, and return it to the deposit via the depot belt. The production sequence should be able to run without an operator. The cell should perform the following steps for each metal plate that is provided:

1. The feed belt conveys the plate from the storage rack to the elevating rotary table;
2. The table rotates and lifts so that the robot can grab the plate;
3. Arm 1 of the robot extends and picks the plate up;
4. The robot turns and Arm 1 places the plate onto the press;
5. The press forges the plate while the robot turns again;
6. Arm 2 picks up the forged plate and places it on the depot belt;
7. The depot belt carries the plate forward to the depot.

It is worth noting that the original system presented in [15] encompasses a traveling crane, too. The task of the traveling crane consists in picking up metal plates from the depot belt, moving them to the feed belt and unloading them there, in order to realize a closed cycle. It acts as a link between the two belts that makes it possible to let the model function continuously, without the need for an external operator. In a more realistic setting, the traveling crane could unload the metal plates into a container, or link the production cell to a further manufacturing unit. For the purpose of this simulation, the traveling crane has been omitted.

2.1. Machines

More in detail, the production cell is composed of the following machines.

Feed Belt The purpose of the feed belt is to move metal plates to the elevating rotary table. The conveyor is driven by an electric motor, which can be activated or deactivated through the control program. A photoelectric cell is situated at the conveyor's end; it detects whether a blank has entered or exited the last section of the conveyor.

Elevating Rotary Table The role of the elevating rotary table is to turn the raw materials by approximately 45 degrees and raise them to a height that enables the first robotic arm to grasp them. This vertical motion is essential due to the distinct positioning of the robot arm compared to the conveyor belt. Furthermore, the table's rotation is indispensable because the gripper on the arm lacks rotation capability, thus being unable to independently position the metal plates accurately into the press.

Robot The robot is constructed with two orthogonal arms, set at two different levels. Each arm can retract or extend horizontally. Both arms rotate jointly. Mobility on the horizontal plane is necessary, since elevating rotary table, press, and depot belt are all positioned at varying distances from the robot's central pivot point. The end of each robot arm is fitted with an electromagnet that allows the arm to pick up metal plates. Arm 1 of the robot picks up the plates from the elevating rotary table and puts them on the press, while Arm 2 picks up the forged piece and places them on the depot belt.

Press The task of the press is to forge metal blanks. The press consists of two horizontal plates, with the lower plate being movable along a vertical axis. The press operates by pressing the lower plate against the upper plate. The plate must coordinate with the robot in order to avoid collisions.

Depot Belt Finally, the task of the depot belt is to transport the work pieces unloaded by the second robot arm to the deposit. A photoelectric cell is installed at the end of the belt; it reports when a work piece reaches the end section. The belt can either run continuously or move only when necessary (i.e., when at least one plate is on it).

2.2. Actuators and Sensors

The system can be controlled using the following actions, which will be simulated in our software simulator.

1. Activate and deactivate feed belt (electric motor);
2. Activate and deactivate depot belt (electric motor);
3. Rotate the elevating rotary table (electric motor);
4. Move the elevating rotary table vertically (electric motor);
5. Move the lower part of the press (electric motor);
6. Rotate the robot (electric motor);
7. Pick up and drop a metal plate with robot Arm 1 (electromagnet);
8. Pick up and drop a metal plate with robot Arm 2 (electromagnet);
9. Extend and retract robot Arm 1 (electric motor, not implemented);
10. Extend and retract robot Arm 2 (electric motor, not implemented);

The control program, in turn, receives the following information from the sensors.

1. Metal plate at the extreme end of the feed belt (photoelectric cell);
2. Metal plate at the extreme end of the depot belt (photoelectric cell);
3. Elevating rotary table position (switch);
4. Elevating rotary table rotation (potentiometer);
5. Press position (switch);
6. Robot rotation (potentiometer);
7. Robot Arm 1 extension (potentiometer, not implemented);
8. Robot Arm 2 extension (potentiometer, not implemented).

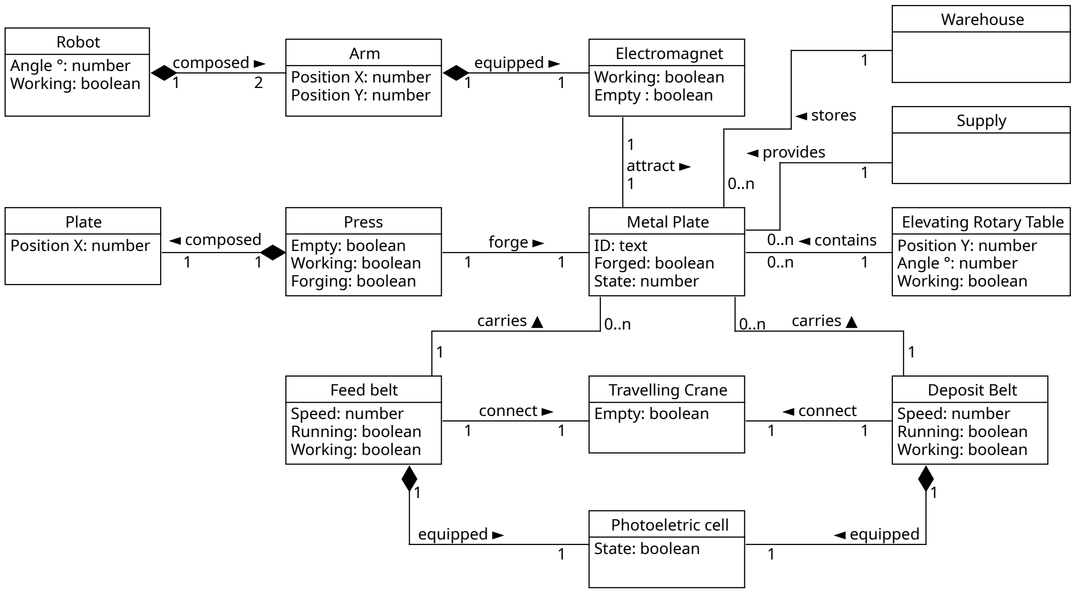

Figure 2: Domain model of the production cell.

3. Simulator Implementation

Starting from the production cell specification described above, we developed a Java-based simulator[1]. To this end, we took inspiration from the work in [16] where the cell is visually simulated by means of Tcl/Tk [17]. The simulator imitates the main features of the real production cell. Moreover, the simulation runs a graphical user interface which displays the elements of the production cell during its functioning and through which the user can interact with the system (e.g., governing the machines).

Figure 2 shows the domain model obtained from the cell specification, which constitutes the core of the simulator. Each machine in the cell has been implemented as a separate class which keeps track of the state of the machine while the simulation is running. The status of the corresponding sensors is maintained, as well. Moreover, each machine offers a usage interface that allows the control program to perform actions over the cell, modifying its state. Such usage interfaces follow the specification of the actuators described in Section 2.2. For example, the following actions can be executed over the elevating rotary table:

rotateTable() makes the table rotate of 20° counterclockwise;

transferTo() moves a metal plate on the table from the terminating point of the feed belt (if any);

[1]The prototype is freely available at https://di.unito.it/productioncell.

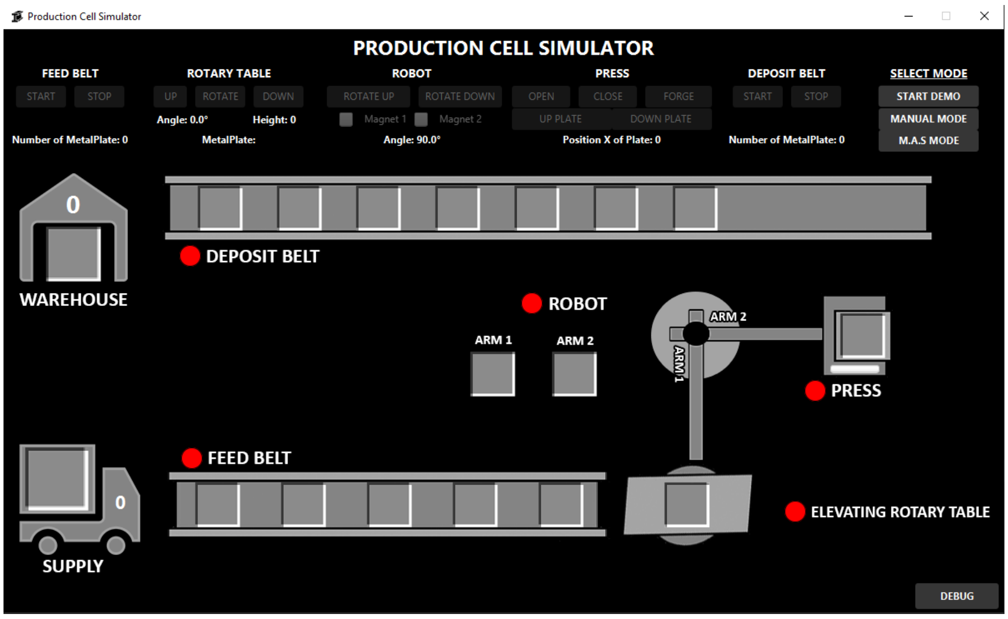

Figure 3: Graphical user interface of the simulator.

upTable()/downTable() modify the height of the table by one unit, either up or down. The minimum reachable height is represented by 0, while the maximum height is represented by 10;

resetTable() resets the table to its initial position (both for what concerns rotation and height).

For brevity we do not report the operations provided by the other machines, which have been implemented in a similar way. It's worth noting that any control program can interact with the machines via dedicated sockets which are maintained and handled by the simulator. As we will explain in the following section, the MAS implemented in JaCaMo has been plugged to the simulator by means of these sockets; through them agents have the possibility to send commands to the machines and receive feedback concerning their status from the simulated sensors.

As said, the simulator also includes a graphical user interface implemented in JavaFX, which is reported in Figure 3. The top part of the interface provides a set of commands through which the user can interact with the cell, sending commands to the various machines. The bottom part of the GUI, instead, shows the machine during its functioning. It is possible to monitor the movements of the machines as well as the positions of the manipulated metal plates. At the top right corner it is possible to select the functioning mode of the simulator. We have three modes:

Demo In this mode, a production cell demonstration is displayed. All machinery are activated, indicated by green lights, and the management of their actions is entirely automatic, just like in a traditional industrial line. The only manual action possible is to add raw

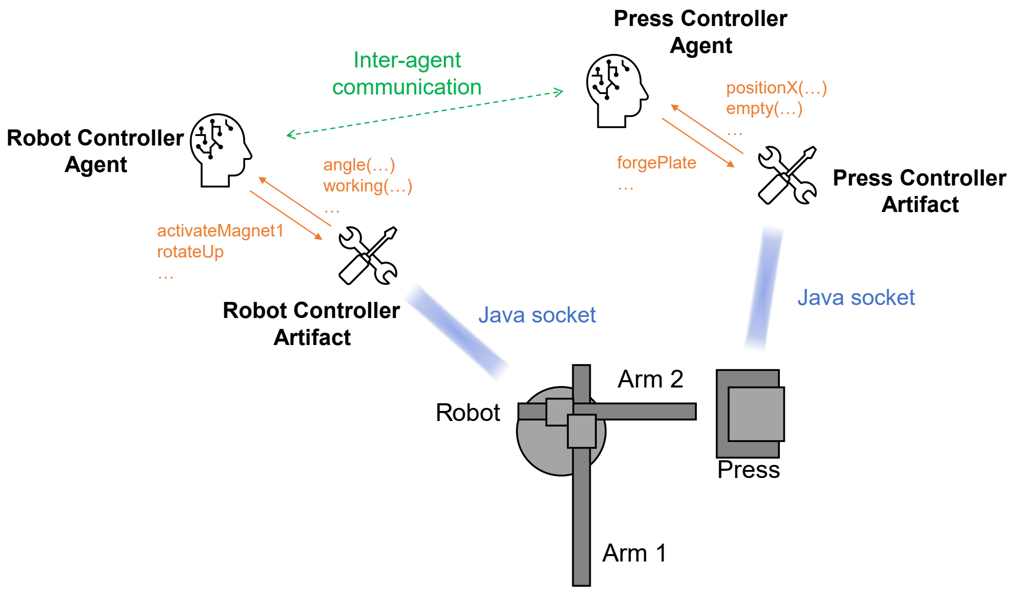

Figure 4: Excerpt of the general architecture of the MAS that governs the production cell.

plates to the cell using the button at the bottom left of the GUI. Of course, here the whole management of the cell is centralized, with serious limitations concerning flexibility, adaptability, and robustness.

Manual mode In this mode, the production cell is entirely governed by the human user through the GUI. Each machinery can be activated by clicking on the corresponding LED indicator, which unlocks all the buttons associated with it. In this mode, the management of machinery actions is entirely manual. The user must use the associated buttons to execute each individual action over the machines.

MAS mode In this mode, the cell is governed by the JaCaMo MAS described in the upcoming section. Once started, the simulator shows the functioning of the cell as soon as the machines receive the corresponding commands from the agents. While using this mode, as well, the only action available to the human user is to add raw plates to the cell.

The simulator is also equipped with a logging service that keeps track of all the operations performed over the cell during the simulations (independently from what mode is selected).

4. MAS Development

For implementing the MAS that constitutes the control program of our simulated production cell, we relied on the well-known JaCaMo framework. JaCaMo [14] is one of the best-known conceptual models and platforms for programming MAS that integrates three different dimensions: agents, environments and organizations. It is built on top of Jason [10], for developing agents, CArtAgO [13], for programming environments, and Moise, [18] for programming organizations. An agent is an entity composed of a set of beliefs, representing the agent's current state and knowledge about the environment, a set of goals, which correspond to tasks the agent has to perform, and a set of plans which are courses of actions, either internal or external, triggered by events, that can be taken by the agent in given circumstances. Software environments are dynamic and distributed sets of *artifacts*. An agent can perceive the observable state of an artifact, reacting to events, and can act upon it by performing actions that correspond to operations provided by an artifact usage interface.

The multiagent system encompasses one controller agent for each machine. At the same time, agents are able to interact with machines by means of a set of dedicated artifacts. An excerpt of the system architecture is depicted in Figure 4. Each artifact maps operations executed by agents to actual commands sent to the production machines. We developed one artifact for each machine, which constitute the access route to the production cell for the agents. Besides operations, artifacts provide agents with a set of observable properties. Observable properties are first order ground facts that are automatically mapped to agents' beliefs and, thus, can be used in their decision making. As said, each time a command is sent to a machine through an artifact (or periodically), the machine sends back information about its updated status (e.g., position of the plates, rotation, height, etc.). This sensory information is exploited within the artifact to define corresponding observable properties, which are then made available to agents. It's worth noting that this allows agents to act upon the cell in real-time, along with the functioning of the machines. Listing 1, for example, show an excerpt of the artifact controlling the press.

```java
public class PressArtifact extends Artifact {
    String serverName = ...;
    int port = ...;

    void init(){
        defineObsProperty("positionX", 0);
        defineObsProperty("empty", true);
        defineObsProperty("forging", false);
        defineObsProperty("isForged", false);

    }


    @OPERATION
    void forgePlate() {
        try {
            Socket clientSocket = new Socket(serverName, port);
            OutputStream outToServer = clientSocket.getOutputStream();
            DataOutputStream out = new DataOutputStream(outToServer);
            out.writeUTF("FORGE");
        } catch (IOException e) { ... }
    }
    @OPERATION void openPress() { ... }
    @OPERATION void closePress() { ... }


    @OPERATION
    void checkEmpty() {
        try {
            Socket clientSocket = new Socket(serverName, port);
            OutputStream outToServer = clientSocket.getOutputStream();
            DataOutputStream out = new DataOutputStream(outToServer);
            out.writeUTF("CHECK PRESS IS EMPTY");
            InputStream inFromServer = clientSocket.getInputStream();
            DataInputStream in = new DataInputStream(inFromServer);
            Boolean empty = Boolean.parseBoolean(in.readUTF());
            clientSocket.close();
            updateObsProperty("empty", empty);
        } catch (IOException e) { ... }
    }
    @OPERATION void checkPositionX() { ... }
    @OPERATION void checkForging() { ... }
    @OPERATION void checkIsForged() { ... }
}
```

Listing 1: Excerpt of the press artifact.

As soon as the artifact is initialized, four observable properties are defined. They encode the position of the press as well as if the press is empty, in operation, or it contains a forged plate to be collected. Annotation @OPERATION denote all the artifact operations that are made available to the agents. Operation forgePlate() for instance, is to be executed in order to to start the forging of a plate positioned under the press. As anticipated, communication between the artifact and the simulator is achieved by means of dedicated sockets. The artifact sends a command to the simulator through the socket, which then executes it over the cell and eventually returns a feedback. Operation checkEmpty() for example, can be used by agents to check whether the press is currently empty or if it already contains a metal plate. In this case, depending on the feedback returned by the simulator, the corresponding observable property's value is updated. The other artifacts (one for each machine) have been implemented in a uniform way.

Listing 2, in turn, shows an excerpt of the code of the agent controlling the press.

```
!start.

+!start
    <- checkEmpty;
       checkPositionX;
       checkForging;
       checkIsForged;
       !controlPress.


+!controlPress
     : empty(false) & forging(false) & positionX(0) & isForged(false)
    <- forgePlate;
       setCondition;
       .wait(...);
       .send(robot,achieve,pickFromPress);
       .send(robot,achieve,activateRobot);
```

```
        !controlPress.

+!controlPress
    <- .wait({+!activatePress});
        checkEmpty;
        checkPositionX;
        checkForging;
        checkIsForged;
        !controlPress.

+!open <- openPress.
+!close <- closePress.
+!activatePress <- ...
```

Listing 2: Excerpt of the press controller agent.

Agents, together, constitute the control program of the production cell. In the example, as soon as the the press controller agent starts, it checks the status of the press. If the press is not empty and it is not yet forging (i.e., a plate is ready to be forged), the second plan will be enabled, making the agent activate the press. After waiting for some time, it notifies the robot that the forged plate can be taken. Otherwise, the third plan will be enabled and the agent will simply wait until a request to move the press is received (e.g., from the robot) before activating the press again. The implementation of the other agents follows the same approach.

It is worth noting that the agents receive sensory information from the environment, but they can also interact and coordinate with each other. This is the case, e.g., for the press and the robot agents. The press controller notifies the robot as soon as the press has finished operating. This activates some internal goals of the latter one allowing the two to act in cooperation.

The adoption of an agent-oriented approach brings along some positive consequences. First of all, it promotes separation of concerns. The the control logic of the system remains separate from the implementation details of the cell, which remain encapsulated inside the artifacts. For this reason, agent programs remain quite simple and easily maintainable. At the same time, since agents can be realized independently from each other, heterogeneity is promoted. In principle, agents may be even programmed by using different programming platforms, and collaborate altogether to the governance of the production cell. By distributing tasks among agents, efficiency and speed are enhanced, enabling parallel processing of tasks. Collaboration between agents, in turn, allows for expertise-sharing, problem-solving, and knowledge aggregation, leading to more flexible and informed outcomes. Finally, the decentralized nature of agents allows addressing robustness and fault tolerance easily, as the failure of one agent does not necessarily disrupt the entire system.

5. Conclusion and Future Work

While there has been a wide research interest in the area of multiagent systems, a few challenges have limited their widespread adoption in real-world industrial scenarios. Some of them may be related, e.g., to scalability, coordination, interoperability, standardization, resource requirements, risk aversion, expertise gap, and transition costs. Many industrial systems are already established and may not easily integrate with multiagent technologies. Retrofitting existing systems to accommodate multiagent interactions could be technically complex and costly. At the same time, industries often prioritize reliability, stability, and safety. Implementing multiagent systems, especially if they involve autonomous decision-making, could raise concerns about system behavior predictability and potential risks. Moreover, importantly, developing and maintaining complex MAS requires specialized knowledge in areas such as distributed systems, artificial intelligence, and control theory. A shortage of experts in these domains within industrial organizations may limit the adoption of such systems.

Despite these challenges, as technologies evolve and solutions to these challenges emerge, we claim that the adoption of multiagent technologies in industrial contexts might become more and more feasible and beneficial. As Industry 4.0 continues to reshape the industrial landscape, the use of multiagent systems emerges as a compelling solution to address the challenges of complexity, dynamism, and decentralization. These systems offer the potential to create adaptive, collaborative, and efficient manufacturing environments by enabling the seamless interaction of heterogeneous components. As research and implementation of multiagent systems progress, they are poised to play a pivotal role in shaping the factories of the future, where intelligent automation and human expertise converge to redefine industrial production.

In this paper we have presented a simulator of a real-world production cell for use in multiagent settings. Even if this is a starting point, its aim is to provide a realistic testbed for multiagent applications. We have illustrated the prototypal implementation of a JaCaMo MAS that is able to effectively control the production cell in an autonomous way. Although some simplifications have been made, this proves the applicability of an agent-oriented approach to the problems and domains that characterize modern industrial infrastructures. We, thus, claim that MAS can play a primary role in reshaping the industry of the future.

Future work will develop along several directions. First of all, we aim at enriching the simulator in order to make it reproduce a real production cell in a more and more accurate way. This also includes tackling perturbations and fault tolerance. Indeed, it would be interesting to evaluate the ability of the system to reorganize and adapt in presence of, e.g., malfunctioning machines, stressful environmental conditions, etc. To this end, the notions of accountability and exception handling may play an important role. Some results along this line can be found in [19, 20, 21]. All such attempts leverage the metaphor of the *organization*. Indeed, it's worth noting that JaCaMo's conceptual model also includes an organizational dimension, which allows defining and managing how agents ought to coordinate in order to achieve a common organizational goal. A further development could be related to evaluating the benefits of adding an organizational layer to the production cell scenario in terms of system development, maintainability, and performance.

For what concerns the simulator development, Java and JavaFX were chosen for the a first proof of concept. As a future work, it would be interesting to evaluate the use of other solutions such as the Robot Operating System (ROS)[2]. The integration of Jason agents with ROS has been investigated in [22, 23]. In [24], in turn, the authors propose a framework for developing autonomous mobile robots using BDI agents. The framework provides the means of connecting the agent reasoning system to simulated ROS environments. Another promising future direction

[2]https://ros.org/

concerns the integration inside the production cell of concepts from the field of the Web of Things. The term Web of Things (WoT)[3] describes a series of standards of the World Wide Web Consortium (W3C) which aim to reduce interoperability problems between Internet of Things (IoT) platforms in different application domains. The approach is based on the concept of Thing Description (TD), i.e., a standardized description of a virtual or physical device (Thing). A TD describes in a machine-interpretable way all the actions, events and properties attributable to a specific Thing. Since each TD can be uniquely identified on the Web, the Web of Things makes it possible to use the Web as an application middleware for a set of heterogeneous connected objects, significantly reducing their fragmentation and allowing their integration into complex systems, spread all over the Internet.

Finally, it would be interesting to integrate heterogeneous agents in the system, which could be eventually developed in isolation by means of different programming languages and platforms. One promising candidate is the SARL framework [12]. SARL is a Java-based programming language and platform that allows deploying agents that follow an event-driven approach. As a future development, we would like to integrate both agents programmed in JaCaMo and in SARL within the same production cell.

Acknowledgments

This publication is part of the project NODES which has received funding from the MUR – M4C2 1.5 of PNRR funded by the European Union - NextGenerationEU (Grant agreement no. ECS00000036).

References

[1] V. Roblek, M. Meško, A. Krapež, A Complex View of Industry 4.0, SAGE Open 6 (2016). doi:10.1177/2158244016653987.

[2] B. Vogel-Heuser, D. Hess, Guest Editorial Industry 4.0–Prerequisites and Visions, IEEE Transactions on Automation Science and Engineering 13 (2016) 411–413. doi:10.1109/TASE.2016.2523639.

[3] A. Ustundag, E. Cevikcan, Industry 4.0: Managing The Digital Transformation, Springer Series in Advanced Manufacturing, Springer International Publishing, 2017.

[4] K. Schwab, The fourth industrial revolution, World Economic Forum, 2017.

[5] X. Yao, J. Zhou, J. Zhang, C. R. Boër, From Intelligent Manufacturing to Smart Manufacturing for Industry 4.0 Driven by Next Generation Artificial Intelligence and Further On, in: 2017 5th International Conference on Enterprise Systems (ES), 2017, pp. 311–318. doi:10.1109/ES.2017.58.

[6] J. Lee, H. Davari, J. Singh, V. Pandhare, Industrial Artificial Intelligence for industry 4.0-based manufacturing systems, Manufacturing Letters 18 (2018) 20–23. doi:https://doi.org/10.1016/j.mfglet.2018.09.002.

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

[3]https://www.w3.org/WoT/

[8] M. Wooldridge, An introduction to multiagent systems, John wiley & sons, 2009.

[9] F. Bellifemine, A. Poggi, G. Rimassa, JADE - a FIPA-compliant agent framework, in: Proceedings of the Practical Applications of Intelligent Agents, 1999.

[10] R. H. Bordini, J. F. Hübner, M. Wooldridge, Programming multi-agent systems in AgentSpeak using Jason, John Wiley & Sons, 2007.

[11] R. W. Collier, S. Russell, D. Lillis, Reflecting on agent programming with AgentSpeak (L), in: PRIMA 2015: Principles and Practice of Multi-Agent Systems: 18th International Conference, Bertinoro, Italy, October 26-30, 2015, Proceedings, 2015, pp. 351–366.

[12] S. Rodriguez, N. Gaud, S. Galland, SARL: A General-Purpose Agent-Oriented Programming Language, in: 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), volume 3, 2014, pp. 103–110. doi:10.1109/WI-IAT.2014.156.

[13] A. Ricci, M. Piunti, M. Viroli, A. Omicini, Environment Programming in CArtAgO, Springer US, 2009, pp. 259–288.

[14] O. Boissier, R. H. Bordini, J. Hübner, A. Ricci, Multi-agent oriented programming: programming multi-agent systems using JaCaMo, MIT Press, 2020.

[15] C. Lewerentz, T. Lindner, Formal development of reactive systems: case study production cell, volume 891, Springer Science & Business Media, 1995.

[16] A. Brauer, C. Lewerentz, T. Lindner, Implementing a visualization of an industrial production cell using Tcl/Tk, in: Proceedings of the first Tcl/Tk Workshop (Berkeley, California, USA), 1993.

[17] J. K. Ousterhout, Tcl and the Tk Toolkit, Addison-Wesley Professional, 1994.

[18] J. F. Hübner, J. S. Sichman, O. Boissier, Developing organised multiagent systems using the MOISE+ model: Programming issues at the system and agent levels, Int. J. Agent-Oriented Softw. Eng. 1 (2007) 370–395. doi:10.1504/IJAOSE.2007.016266.

[19] M. Baldoni, C. Baroglio, R. Micalizio, S. Tedeschi, Reimagining Robust Distributed Systems through Accountable MAS, IEEE Internet Computing 25 (2021) 7–14. doi:10.1109/MIC.2021.3115450.

[20] M. Baldoni, C. Baroglio, R. Micalizio, S. Tedeschi, Exception Handling as a Social Concern, IEEE Internet Computing 26 (2022) 33–40. doi:10.1109/MIC.2022.3216272.

[21] M. Baldoni, C. Baroglio, R. Micalizio, S. Tedeschi, Accountability in multi-agent organizations: from conceptual design to agent programming, Autonomous Agents and Multi-Agent Systems 37 (2023). doi:10.1007/s10458-022-09590-6.

[22] G. R. Silva, L. B. Becker, J. F. Hübner, Embedded architecture composed of cognitive agents and ros for programming intelligent robots, IFAC-PapersOnLine 53 (2020) 10000–10005. doi:10.1016/j.ifacol.2020.12.2718, 21st IFAC World Congress.

[23] I. Silvestre, B. de Lima, P. H. Dias, L. Buss Becker, J. F. Hübner, M. de Brito, Uav swarm control and coordination using jason bdi agents on top of ros, in: P. Mathieu, F. Dignum, P. Novais, F. De la Prieta (Eds.), Advances in Practical Applications of Agents, Multi-Agent Systems, and Cognitive Mimetics. The PAAMS Collection, Springer Nature Switzerland, 2023, pp. 225–236.

[24] P. Gavigan, B. Esfandiari, Agent in a box: A framework for autonomous mobile robots with beliefs, desires, and intentions, Electronics 10 (2021). doi:10.3390/electronics10172136.