Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 107 (2022) 1257–1262

www.elsevier.com/locate/procedia

55th CIRP Conference on Manufacturing Systems

Towards Modular and Plug-and-Produce Manufacturing Apps

Agajan Torayev[a], Giovanna Martínez-Arellano[a], Jack C Chaplin[a], David Sanderson[a], Svetan Ratchev[a]

[a]*Institute for Advanced Manufacturing, Jubilee Campus, University of Nottingham, Nottingham, NG7 2RD, UK*

* Corresponding author. Tel.: +447799077892. *E-mail address:* agajan.torayev@nottingham.ac.uk

Abstract

Industry 4.0 redefines manufacturing systems as smart and connected systems where software solutions provide additional capabilities to the manufacturing equipment. However, the connection of manufacturing equipment with software solutions is challenging due to poor interoperability between different original equipment manufacturers (OEMs), making it difficult to integrate into the manufacturing system. Hence, there is a need for a methodology to develop modular "plug-and-produce" applications in the manufacturing domain to meet the requirements of Industry 4.0. This work investigates the "appification" of manufacturing processes where the goal is to sub-divide the process into independent, re-configurable digital manufacturing applications. In this context, "appification" means separating the digital implementation from the physical implementation of the system by making the former modular and independent so that digital implementations can be re-used without depending on the physical parts of the system. In this paper a framework for the development of such manufacturing "apps" is presented. This framework consists of four main elements: a modular plug-and-produce architecture, a manufacturing apps development kit, a communication protocol, and a construction methodology. The modular plug-and-produce architecture is developed using the recent advances in microservices, containerization, and communication technologies. The manufacturing apps development kit (MAPPDK) has been developed to facilitate the implementation of manufacturing apps using high-level programming languages. MAPPDK allows to control manufacturing equipment from external computational devices. The methodology for developing different modules for different types of manufacturing processes is also provided. The proof of concept is shown experimentally by the "appification" of a sorting process using an industrial robot arm, a gripping end-effector, a third-party vision camera, and an intelligent vision module.

Keywords: manufacturing apps; digital manufacturing; industry 4.0; smart manufacturing.

1. Introduction

The manufacturing industry is shifting from mass production towards mass customization [6], which opens new challenges for existing manufacturing systems in adapting to changing environments and product specifications, increasing reconfiguration time and efforts. These challenges arise from existing manufacturing systems' tightly interconnected structure that hinders flexibility and adaptability to new requirements.

The current approaches to developing manufacturing software solutions use a monolithic approach where different functionalities are combined, limiting the "plug-and-produce" functionality of the whole system. Plug-and-produce functionality is a concept derived from plug-and-play self-configuring devices used in computing. In manufacturing, it aims to improve the interoperability and reusability of manufacturing modules by automatically identifying new components and integrating them into a production system without manual efforts [1, 10].

In current approaches, if some part of the system changes, the software solutions must be re-written to reflect the changes. For example, robot controllers have been available for more than 60 years, but they do not have a plug-and-produce functionality. Instead, robot controllers need to be manually and laboriously configured to support adding new functionalities and connected devices, limiting their ability to adapt to new environments and tasks.

Instead of developing a monolithic software solution for the whole task, we propose developing software solutions with independent and modular components. Manufacturing systems should take inspiration from IT computer systems and support modular and plug-and-produce functionalities to meet the changing market requirements and stay competitive in the global industrial marketplace.

10.1016/j.procir.2022.05.141

This work investigates the modularization of manufacturing systems from a software perspective. Our goal is to provide a methodological approach for developing software solutions for manufacturing equipment. We investigate the microservices and containerization technologies in the manufacturing domain. Based on our findings, we propose:

- a new method of developing modular and plug-and-produce software solutions for manufacturing systems;
- an architecture with the necessary characteristics to develop modular and plug-and-produce software solutions;
- a manufacturing app development kit (MAPPDK) as a solution for developing interoperable, modular, and plug-and-produce manufacturing apps.

The remainder of the paper is organised as follows: In the Related Work section, we compare existing solutions in the literature and analyze their advantages and limitations. In the Methodology section, we define the appification of manufacturing processes, define a conceptual architecture for modular and plug-and-produce manufacturing apps, and introduce a software development kit. In the Experiment section, we show the experimental proof of concept using a sorting process that involves an industrial robotic arm, vision app, and intelligence app. The manuscript ends with a Conclusions section.

2. Related Work

Microservices is an architectural approach in which software is composed of a loosely coupled and independently deployable collection of services. Unlike a monolithic architecture where all processes are tightly coupled and run as a single service, applications in a microservices architecture are broken into small and independent components to accomplish the same task. The microservices architecture tries to remove the unnecessary complexities that exist in a service-oriented architecture to achieve flexibility, interoperability [2], independence, and scalability [5].

Containerization is a method of encapsulating a software package with source code, related libraries, configuration files, and required dependencies to run it on the host operating system. Containers allow the “write once, run anywhere” paradigm, removing operating system and software compatibility problems.

The microservices architecture and containerisation technologies are popular choices for developing manufacturing architectures and frameworks with different purposes and goals. Ibarra-Junquera et al. introduced a flexible, scalable and robust framework based on container technology and microservices for real-time control capabilities in the automation of industrial bioprocesses [7]. Goldschmidt et al. used a container-based solution to build a multi-purpose industrial controller which addresses legacy hardware emulation and flexible function deployment [3]. Gonzalez-Nalda et al. introduced a generic modular architecture for cyber-physical Systems based on the Robot Operating System (ROS) and Docker [4]. Thramboulidis et al. proposed a framework for assembly systems based on microservices and the Internet of things (IoT) [12]. Senington et al. employed container technologies such as Docker, Docker registries, and Docker-compose registries to solve software management, deployment, configuration and integration in smart factories in four locations around the European Union [11].

Interoperability between devices and machines often relies on multiple heterogeneous communication protocols, which hinders the development of modular manufacturing architectures. The presence of heterogeneous technologies in manufacturing systems makes manufacturing processes complex and difficult to control. To this end, Nikolakis et al. proposed an end-to-end software framework to bridge the high-level planning and the low-level execution control using IEC61499 compliant function blocks modelling of manufacturing operations executed using docker containers [8]. Rufino et al. considered end-devices as cyber-physical systems capable of running containerised services and abstracted interoperability using Representational State Transfer (REST) protocols and a distributed database at the gateway for communication mediation [9].

The above research publications show that microservices architectures and containerization are often adopted together to achieve flexible and agile automation systems in manufacturing. Microservices architectures and containerization help deal with multiple constraints such as ease of installation and deployment, management, security, and scalability in modern industrial automation systems. However, some limitations hinder the further development and adoption by the manufacturing community in the existing approaches.

The first research gap in the existing literature is the lack of methodology for developing manufacturing solutions. Although all authors use microservices and containerization technologies for developing manufacturing software solutions, none of the authors provide a methodological way of developing such manufacturing applications. This lack of a unifying method of developing manufacturing applications makes it difficult for practitioners to develop interoperable, modular, and plug-and-produce solutions. No software development kit in the existing literature is specially designed for developing manufacturing software solutions. We address this issue by proposing MAPPDK.

The second research gap that we address in our work, which is not addressed in the existing literature, is the re-usability and distribution of developed software solutions for a similar manufacturing process. We treat this problem similar to mobile application development practice, where the developed apps are distributed through the central hub.

The third research gap in the literature is the choice of developing manufacturing software solutions. The authors above use a top-down approach for developing manufacturing solutions. In the top-down approach, a high-level definition of a problem is subdivided into smaller problems and the respective solutions developed for the whole problem. While this method is easy to implement, extending it to other processes is difficult because of the problem-specific solutions developed in a top-down approach. Thus, we use a bottom-up approach where each system

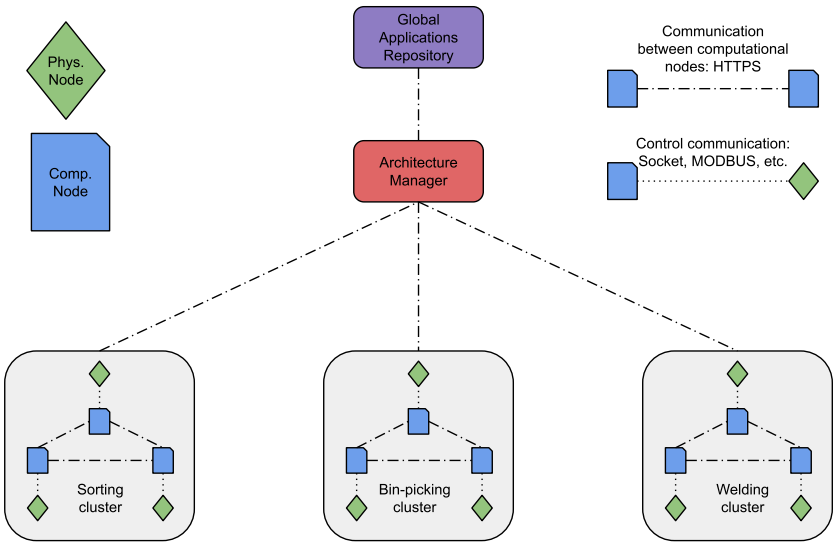


Fig. 1: Conceptual architecture that enables the development of modular and plug-and-produce manufacturing software solutions. This diagram serves as a guideline for implementing and extending a modular architecture presented in Section 3.2.

module is developed individually. This method allows re-using and distributing developed solutions for similar use-cases.

In the next section, we discuss our methodology for developing modular and plug-and-produce manufacturing software solutions.

3. Methodology

3.1. Appification of manufacturing processes

In a manufacturing process such as assembling parts to make a final product, the whole process can be divided into smaller operations. Each operation in this process involves some manufacturing equipment that depends on a software solution to control the equipment. With the connectivity offered by Industry 4.0, stand-alone control code is no longer enough to control equipment, and instead the software solution includes control code as well as access to other data sources to coordinate production. We call these software solutions 'manufacturing apps' and give the following definition to "appification" of the manufacturing process:

The "appification" of the manufacturing process refers to separating digital and physical implementations of the manufacturing system to make implementations modular and reusable for similar manufacturing processes.

By "digital implementation" we mean the software component, and by "physical implementation" we mean the specific hardware in the manufacturing system.

The above definition assumes that each physical and digital component of the manufacturing system can be modularized to enable the development of apps for each module of the system.

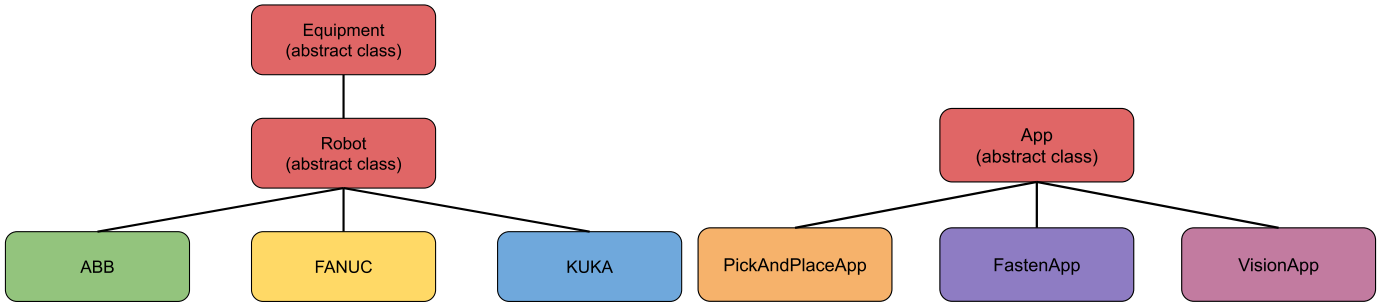
The appification allows the reusability of developed solutions. For instance, a pick-and-place process that consists of an industrial robot arm, an end-effector, and a vision system can be modularized into three independent modules with their apps that provide the required capabilities. A vision system can be considered a separate system with autonomous object detection capability using machine learning methods. If, for some reason, the vision system needs to be replaced during production, the other two modules will not be affected by this change; or if there is a need to get extra capability such as pose estimation, it should be possible to do so by implementing the necessary app. Similarly, if the robot or end-effector changes, it should be possible to replace them without spending too much time on re-adjusting all three modules.

3.2. Modular architecture

However, to be able to develop manufacturing apps, there is a need for an architecture that meets certain requirements; in particular, a conceptual architecture must have the following inherent characteristics:

1. An architecture must enable modular physical implementations, i.e., physical components must be independent of each other.
2. An architecture must enable modular digital implementations, i.e., software components must be independent of each other.
3. An architecture must enable plug-and-produce properties, i.e., physical and digital components must be replaceable and reusable with minimum necessary configuration.
4. An architecture must enable the distribution of developed solutions by publishing to the central hub to enable the reusability of developed solutions in similar scenarios.

In this work, we propose a modular and plug-and-produce architecture that has the above inherent characteristics. Before giving implementation details, we first define the below terms that are necessary for an architecture to be modular and enable plug-and-produce:



(a) Abstract Equipment class (b) Abstract App class

Fig. 2: Manufacturing App Development Kit interfaces. (a) demonstrates the object-oriented way of developing manufacturing equipment software. (b) demonstrates the object-oriented way of developing manufacturing apps.

- **Atomic device**. An atomic device is a single physical device intended to perform one specific operation such as gripping with a gripper, capturing an image with a camera, or measuring force with a force sensor. Dividing every physical device of the manufacturing system into atomic devices with one specific goal enables the physical modularity of the architecture.
- **Physical node**. A physical node consists of one or more atomic devices to perform a manufacturing process. For example, an industrial robot arm, camera, and gripper may form one physical node to sort products. The definition of the physical node is necessary for managing atomic devices that are grouped to perform one manufacturing process.
- **Computational node**. A computational node is a device such as a Raspberry Pi that can control the physical nodes or the atomic devices in the physical nodes. A computational node must have the ability to host variable software solutions using microservices and containerization technologies. The definition of computational nodes helps solve the interoperability bottleneck in the existing manufacturing systems by lifting the communication and digital implementations to high-level computational devices. Computational nodes enable digital modularity of the manufacturing systems.
- **Manufacturing process cluster**. A manufacturing process cluster is a group of physical and computational nodes with one specific goal: sorting, bin-picking, palletizing, welding, drilling. The manufacturing process cluster helps organize physical nodes and computational nodes for management.
- **Architecture manager**. Some of the functions of an architecture manager are installing/updating/removing apps, assigning and re-assigning computational nodes and physical nodes to manufacturing process clusters. The architecture manager enables the plug-and-produce property of the manufacturing systems by giving manufacturing engineers high-level management tools and an interface for developers to develop software solutions without considering management-related issues.
- **Global applications repository**. The global application repository is a central hub for the publishing and delivering manufacturing apps. Global applications repository enables the reusability of developed solutions for similar processes.

This conceptual architecture is depicted in Figure 1. In the above architecture, atomic devices, physical nodes, and manufacturing process clusters are concepts that serve as the basis for the architecture; computational nodes, the architecture manager, and the global applications repository can be developed using the recent advances in information technologies such as microservices and containerization.

Docker was used for containerizing manufacturing apps. We developed a computational node using Raspberry Pi and used Ubuntu IoT as an operating system; however, any computational device with the minimum requirements to run container technologies can serve as a computational node for this architecture.

We also developed an architecture manager using an in-house container management software solution. This solution is written using NodeJS and has a web interface for managing different manufacturing apps. Computational nodes can also be managed using more advanced orchestration technologies such as Kubernetes. For the sake of simplicity, we used our custom-developed approach for this work.

3.3. Manufacturing Apps Development Kit

The modular architecture acts as an enabler for developing manufacturing apps. However, another challenge that currently hinders the development of manufacturing apps is the lack of a software development kit (SDK) specially designed for this purpose. An SDK is a collection of tools that facilitate the creation of applications for a given environment. Every OEM has an SDK suitable only for that hardware. For example, FANUC has FANUC PC Developers Kit, ABB has RobotStudio SDK, and KUKA has its software collection for developing robotic applications. It is extremely difficult to make them work together because of different vendor-specific standards [13]. However, encapsulating the manufacturing process into a computational

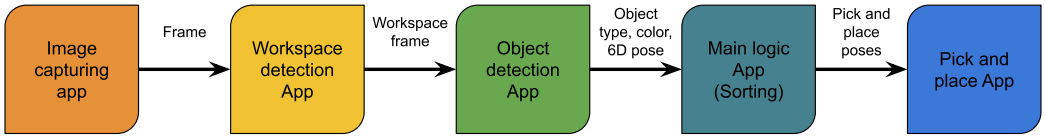


Fig. 3: Sorting process. In this diagram, every operation is appified to make it modular and plug-and-produce. In our experiments, we implement each component in the above diagram as a Python module, making the code reusable for similar scenarios.

node that runs containerized software solutions allows us to overcome this existing limitation.

We developed MAPPDK following an object-oriented programming paradigm, i.e., communication, manufacturing equipment, capability are objects, and particular instances inherit parent properties. The high-level manufacturing equipment class is shown in Figure 2a. This abstract class defines common methods such as data logging and error handling. For instance, Robot is a child class that inherits properties from the manufacturing equipment class and has robot-specific methods such as getting the current position of end-effector, getting current joint values, moving to the user-defined position. In MAPPDK, the abstract Robot class defines all robots' necessary methods and general properties and can be further extended by developing robot-manufacturer-specific properties. For example, ABB, KUKA, and FANUC all have different Euler-angle conventions. These robot-manufacturer-specific details can be integrated by inheriting the abstract Robot class and overwriting the necessary functions. This development approach allows changing robots for the manufacturing process without re-implementing the same solution for each robot separately. Similarly, other manufacturing equipment can be integrated by inheriting abstract manufacturing equipment class.

MAPPDK implements an abstract App class as shown in 2b for developing manufacturing apps, defining configurations management, errors, and exception handling methods common to all manufacturing apps. Thus, manufacturing apps developed by inheriting abstract App class have access to the common interface, facilitating plug-and-produce and modular development. For example, pick-and-place, fastening, vision, drilling can all be implemented by inheriting the App class.

Furthermore, we have developed a fast and lightweight communication protocol that can be adapted to new robot manufacturers only by writing the driver for the robot controller. We implemented our protocol because not all robot controllers support the current standards. The MAPPDK side will stay the same for all robots to enable easy replacement and reusability of developed solutions. In this implementation, the robot controller acts as a server, and MAPPDK acts as a client requesting commands to execute. The server implementation runs until an operator stops, while client implementation runs only for one command execution.

4. Experiments

In this section, we experimentally demonstrate the proof of concept of the appification of manufacturing processes. As a use case, we selected a simple yet industrially relevant process of sorting products. In this setting, an industrial robot arm needs to sort products based on their colors. In particular, there are three different cylinders with different shapes and colors. The task for an industrial robot arm is to pick these cylinders from random positions and place them in their respective containers, as shown in Figure 4.

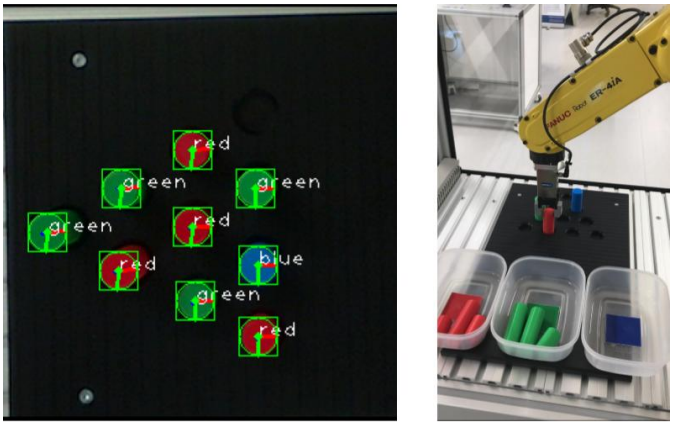


Fig. 4: The demonstration on real manufacturing equipment. In this experiment, a FANUC industrial robot utilizes the manufacturing apps depicted in Figure 3 to sort cylinders with different heights and colors. The Figure on the left shows the final output after image capturing, workspace detection, and object detection. The Figure on the right shows the actual robot execution.

We used a 6-axis FANUC ER-4iA industrial robot, which has an R-30iB Mate Plus Controller, a Schunk gripper, and an Intel RealSense L515 LiDAR camera. Also, the industrial robot cell has a built-in FANUC iRVision camera. The heterogeneity of devices, i.e., different OEMs, is a good example for demonstrating the modular manufacturing app development using MAPPDK.

In this setup, the industrial robot arm, gripper, and vision camera are all considered atomic devices which make one physical node with one manufacturing process goal - sorting cylinders. A Raspberry Pi Model 4 acts as the computational node that runs the Docker container on top of the Ubuntu IoT operating system. The whole manufacturing process cluster is controlled by the architecture manager developed using NodeJS.

In our experiments, we developed the sorting process in a bottom-up approach to demonstrate the modularity of apps. Initially, we developed a simple pick and place app that picks an object from any given pick position and places it to a predefined place. This app receives a 6D pose of an object for picking and placing.

After verifying the correctness of the pick and place app, we integrated a built-in cell camera to detect one type of cylinder.

For this, we developed the image capturing, workspace detection, and cylinder detection apps.

The image capturing app is only responsible for capturing images with an external camera. This app is device-agnostic and can work with different types of cameras. The goal of the workspace detection app is to detect the robot's workspace. The task of the object detection app at this stage was to detect the position of one type of cylinder in the workspace.

In the next step, we changed the built-in camera with a third-party LiDAR camera. The introduction of a new camera did not introduce any changes to other apps in the system. Hence, we achieved the modularity and plug-and-produce for integrating a vision module.

After integrating a new camera, we changed the types of objects, i.e., we introduced cylinders with different shapes and colors. This new change required only change in the object detection app, proving the modularity and plug-and-produce of integrating an intelligence module. We used the OpenCV library for implementing the vision-based logic. We used RANSAC based Perspective-n-Point algorithm for detecting 6D poses of cylinders. However, the object detection app can further be expanded using more advanced machine learning methods, such as deep learning-based 6D pose estimation methods.

In the next step, we integrated more intelligent capability - sorting the object in the workspace. For this, we needed to change only the main logic app, which has a sorting function. The final sorting process app is shown in Figure 3.

Our aim with these experiments is to show the possibility of developing manufacturing apps using high-level programming languages. In particular, we used the Python programming language for performing the defined tasks.

5. Conclusions

In this work, we investigated the appification of manufacturing processes to develop modular and plug-and-produce manufacturing apps. We defined the methodology for developing and deploying apps in an industrial environment.

The concept of appification of manufacturing processes was shown experimentally by developing a sorting app that involves heterogeneous devices. Existing solutions implement this in a monolithic approach where all the required steps are tightly dependent on each other. The appification of manufacturing processes helps to divide all the steps into modular apps independent from each other. Plug-and-play has proven extremely successful in the computer science domain, and plug-and-produce achieves a similar goal by making it easier for manufacturers to introduce new equipment from new technology suppliers, thus incentivizing technology providers to support the approach.

Plug-and-produce has previously been addressed in terms of modularity of physical resources and associated interfaces (e.g., Antzoulatos et al. [1], Sanderson et al. [10]). In this work, we have addressed plug-and-produce from the perspective of modular cyber-physical capabilities, where the software components and the hardware components are not intrinsically locked together, and instead, we allow for different software components to be chosen depending on the application at hand and the range of available hardware components.

One approach to achieving a plug-and-produce paradigm and simultaneously securing intellectual property is ensuring syntactic and semantic interoperability [2], where the OEMs agree on a common data format and disclose other things related to securing IP. IP security is a key concern, and a data access model that defines what information is required for the plug-and-produce will be a future development.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 814078.

References

[1] Antzoulatos, N., Castro, E., Scrimieri, D., Ratchev, S., 2014. A multi-agent architecture for plug and produce on an industrial assembly platform. Production Engineering 8, 773–781.

[2] Estrada-Jimenez, L.A., Pulikottil, T., Hien, N.N., Torayev, A., Rehman, H.U., Mo, F., Hojjati, S.N., Barata, J., 2021. Integration of cutting-edge interoperability approaches in cyber-physical production systems and industry 4.0, in: Design, Applications, and Maintenance of Cyber-Physical Systems. IGI Global, pp. 144–172.

[3] Goldschmidt, T., Hauck-Stattelmann, S., Malakuti, S., Grüner, S., 2018. Container-based architecture for flexible industrial control applications. Journal of Systems Architecture 84, 28–36.

[4] González-Nalda, P., Etxeberria-Agiriano, I., Calvo, I., Otero, M.C., 2017. A modular cps architecture design based on ros and docker. International Journal on Interactive Design and Manufacturing (IJIDeM) 11, 949–955.

[5] Homay, A., Zoitl, A., de Sousa, M., Wollschlaeger, M., 2019. A survey: Microservices architecture in advanced manufacturing systems, in: 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), IEEE. pp. 1165–1168.

[6] Hu, S.J., 2013. Evolving paradigms of manufacturing: From mass production to mass customization and personalization. Procedia Cirp 7, 3–8.

[7] Ibarra-Junquera, V., González, A., Paredes, C.M., Martínez-Castro, D., Nuñez-Vizcaino, R.A., 2021. Component-based microservices for flexible and scalable automation of industrial bioprocesses. IEEE Access 9, 58192–58207.

[8] Nikolakis, N., Senington, R., Sipsas, K., Syberfeldt, A., Makris, S., 2020. On a containerized approach for the dynamic planning and control of a cyber-physical production system. Robotics and computer-integrated manufacturing 64, 101919.

[9] Rufino, J., Alam, M., Ferreira, J., Rehman, A., Tsang, K.F., 2017. Orchestration of containerized microservices for iiot using docker, in: 2017 IEEE International Conference on Industrial Technology (ICIT), IEEE. pp. 1532–1536.

[10] Sanderson, D., Shires, E., Chaplin, J.C., Brookes, H., Liaqat, A., Ratchev, S., 2020. Context-aware plug and produce for robotic aerospace assembly, in: International Precision Assembly Seminar, Springer. pp. 184–199.

[11] Senington, R., Pataki, B., Wang, X.V., 2018. Using docker for factory system software management: Experience report. procedia CIRp 72, 659–664.

[12] Thramboulidis, K., Vachtsevanou, D.C., Kontou, I., 2019. Cpus-iot: A cyber-physical microservice and iot-based framework for manufacturing assembly systems. Annual reviews in control 47, 237–248.

[13] Vicente, L., Lomelino, P., Carreira, F., Campos, F.M., Mendes, M.J., Osório, A.L., Calado, J.M.F., 2021. Industrial collaborative robotics platform, in: Working Conference on Virtual Enterprises, Springer. pp. 567–576.