Utilizing software design patterns in product-driven manufacturing system: a case study

Dmitrii Drozdov, Udayanto Dwi Atmojo, Cheng Pang, Sandeep Patil, Muhammad Irfan Ali, Antti Tenhunen, Taavi Oksanen, Kiril Cheremetiev, and Valeriy Vyatkin

Abstract This paper presents the development of a flexible product-driven manufacturing case study. The case study is based on Festo EnAS (Energieautarke Aktoren und Sensoren" or "energy efficient actuators and sensors" in English) platform, enhanced with mobile robot for logistics and a camera for automated visual-based product inspection. IEC 61499-based and wireless-capable embedded controllers are used to enable decentralized control architecture. In this case study, a software design pattern is considered to enable easier development of modular production systems with physically reconfigurable layout during the production process. This paper applies the software design pattern into production systems where product-driven approach is adopted, i.e. the overall production process is determined based on the product specifications/requirements from the customer.

Key words: design patterns, IEC 61499, product-driven manufacturing.

Dmitrii Drozdov
Luleå University of Technology, Luleå, Sweden and Penza State University, Penza, Russia; e-mail: dmitrii.drozdov@ltu.se

Udayanto Dwi Atmojo
Aalto University, Helsinki, Finland; e-mail: udayanto.atmojo@ieee.org

Cheng Pang
Jiangmen Goobotics Research Institute and Googol Technology (Shenzhen) Ltd.,Guangdong, China; e-mail: cheng.pang@ieee.org

Sandeep Patil
Luleå University of Technology, Luleå, Sweden; e-mail: sandeep.patil@ltu.se

Muhammad Irfan Ali, Antti Tenhunen, Taavi Oksanen, Kiril Cheremetiev
Aalto University, Helsinki, Finland;

Valeriy Vyatkin
Luleå University of Technology, Luleå, Sweden and Aalto University, Helsinki, Finland; e-mail: vyatkin@ieee.org

1 Introduction

Traditionally, industrial manufacturing has been focusing on manufacturing large quantity of uniform, homogeneous products (mass production). However, recently there is an increasing trend in manufacturing which aims at customized products as requested or specified by the customer (mass customization). This creates new challenges which are not encountered in the traditional factory floor/ manufacturing setting. In this new setting, the factory floor needs to be flexible and agile to perform different scheduling and sequences of production, or even changing physical layout of machines on the fly in order to cope with various product requirements and specifications while also taking into account the availability of required resources, timeliness of completion, and also non-critical factors such as energy consumption. Product-driven manufacturing [1] paradigm can become a solution for these new demands, where the overall production processes/sequences are determined based on requirements or specifications of the product requested by the customer/client. In the product-driven manufacturing, the flow in the production line changes and is often determined on the fly.

One possible consequence of the product-driven manufacturing approach is a possible need to install new machines/stations, in addition to taking into account failures which may happen at any time. This requires smarter machines that are equipped with embedded control devices, implying, in turn, decentralized control architecture.

Most of automation systems currently have centralized control architecture, supported by mainly IEC 61131-3 standard [2]. Along with insufficient flexibility, such control architecture poses the risk of single point of failure, i.e., the production system will halt if the sole controller fails or becomes inoperable. There is a trend in shifting towards decentralized/distributed architecture, where software control for of mechatronic devices is executed on multiple interconnected hardware controllers, e.g., Programmable Logic Controllers (PLCs). The IEC 61499 standard [3] extends IEC 61131-3 standard with decentralized control architecture and enables modularity and reconfigurability in software design.

This paper reports the use of software design patterns in a product-driven manufacturing case study which is based on laboratory-scale model of assembly system called EnAS demonstrator [4] ("Energieautarke Aktoren und Sensoren" or "energy efficient actuators and sensors" in English). In this case study, the EnAS demonstrator is enhanced with the layout reconfiguration feature implemented by placing a part of production and material handling units on top of an autonomous mobile robot. Other flexibility support features include the use of wirelessly communicating, internet-addressable, and IEC 61499-enabled distributed controllers embedded into all basic mechatronic units. To enable easier development of software control and promote reusability of software components, a software design pattern called "product-driven software design pattern" is proposed and will be described in this paper.

The rest of this paper is structured as follows. Section 2 briefly presents related work of existing design patters, it will specifically address IEC 61499 applications. Section 3 presents the description of the case study which utilized the "product-driven software design pattern". Section 4 elaborates the "product-driven software design pattern" architecture, and finally, some conclusions and future works are given in Section 5.

2 Related Work

Software accounts for more than 50% of the effort of building an industrial automation systems. In order to build systems that efficient, robust and scalable a software design and implementation methodology is needed. In the context of industrial automation system software, the methodologies used and researched today are mainly dealing with very high level architecture. On the other hand, in the pure software engineering domain (Java, C, C++ programming) there exists mature research for using design patterns. Design patterns in software engineering presented in [5] have taken an important place in the traditional field mainly because they come from expert system designers. There is need for such systematic low level software design in the context of software for industrial automation systems, a fact well highlighted in [6, 7].

At the core of both IEC 61499 [3] and IEC 61131-3 [2] industrial control software standards is the concept of Function Block(FB) which supports modularity, hence reusability. FB can be treated as representing an object because FB's have types (encapsulate data and algorithms) and they are instantiated in order to be used. Their interface is well defined making it easier to communicate with other FB's (objects). Given these features, programming using these standards can be considered object-oriented. However, strictly speaking, these standards are not object-oriented programming since they don't support *inheritance*. (With exception of the object-oriented extension to IEC 61131-3 implemented in CoDeSys tool.)

Different categories exist, to which a design pattern can belong to. [8] presents these categories. [8] also presents a good overview of current design patterns in the field of industrial automation systems, Table. 1 summarizes the review. It can be safely said that the majority of the works deal with model-driven design and engineering applicable to industrial automation systems. The patterns presented are mainly for function block application(FBA) architecture and do not address patterns for the design of basic function blocks(BFB) and composite function blocks(CFB). This is due to the fact how traditionally control engineers design and implement control software currently. There is lack of implementation patterns as most works deal with high-level design. The contribution of this paper's design pattern can be classified as Compositional design pattern.

Table 1 Design patterns summary

Pattern Name	**Presented in**	Category Name
Distributed Application	[9, 10]	Architectural
Proxy	[9, 10]	Architectural
Model-View-Control(MVC)/Model-View-Control-Diagnostics(MVCD)	[9, 10]	Architectural
Ordered Synchronous	[10]	Architectural
Delayed Synchronous	[10]	
Hierarchical Control	[11]	Architectural
Motion control	[11]	Architectural
Alarm handling pattern	[11]	Architectural
Generic Model-driven design patterns	[12–16]	Architectural
Intelligent Mechatronic components	[17]	Structural
Peer-to-Peer	[18]	Behavioral
Master-Slave	[18]	Behavioral
"Channel"-based pattern	[19]	Handshake

3 EnAS Demonstrator: Case Study

EnAS demonstrator, a lab scale model of an automated assembly system, has its mechatronic components supplied by Festo. The physical layout of the EnAS demonstrator is shown in Fig. 1. The demonstrator consists of a conveyor system constructed from six individual conveyors, a set of mechatronic stations consisting of two gripper stations, two jack stations, and ten photocell sensors to detect the position of workpieces on the conveyor system. Each jack station has a sledge, Sledge 1 for Jack Station 1 and Sledge 2 for Jack Station 2, which are used to put workpieces to be taken from or to be put inside a container on the conveyor. Gripper station (Gripper 1 and Gripper 2) can grip and lift workpieces from the container and place them on a different container on the conveyor.

3.1 Previous Configuration

The EnAS demonstrator utilized metal pallets to transport workpieces. Each pallet has two pairs of larger and smaller slots, totaling four slots altogether. Metal pallets which carry workpieces are then transported on the conveyor system in a clock-wise direction. The movement of the workpiece on the conveyor may be stopped when the workpiece reaches certain position, e.g., when it is adjacent to a mechatronic station.

Prior to the case study presented in this paper, several case studies based on the EnAS demonstrator have been developed. For example, Gröhn et al [20] deploys six Nxt DCS Mini controllers (developed by NxtControl) to run IEC 61499 control logic that governs the mechatronic machines through wireless communication. Then, the next development phase was reported

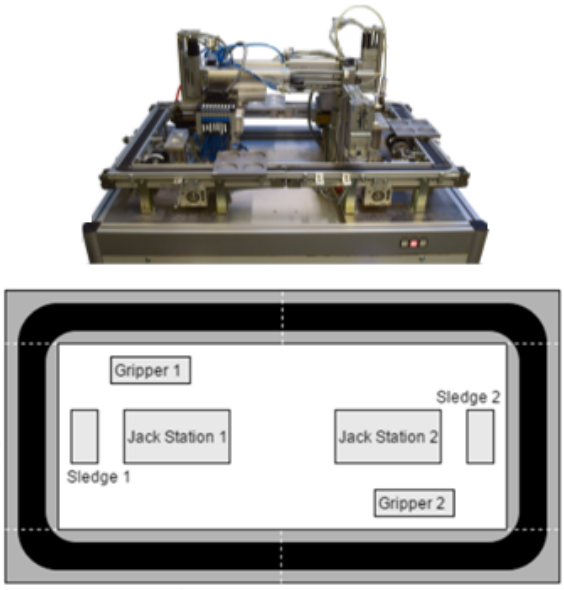

Fig. 1 The physical platform of EnAS (top) and layout of EnAS (bottom)

in [21], where the EnAS demonstrator was extended with several features, including more distribution of the software control logic on 10 Nxt DCS Mini controllers, the capability to produce multiple types of products, a camera for workpiece inspection, and a machine to machine (M2M)-typed interaction to incorporate certain degree of flexibility and fault tolerance. The camera is connected to a Raspberry Pi 3 model B embedded controller which runs a Python-based script that performs color detection to inspect workpiece. The most recent development which follows the aforementioned works is described in this paper.

3.2 Modifications to accommodate product-driven manufacturing paradigm

Physical Modifications. From the physical aspect, several changes are introduced from the previous development phases.

- Metal pallets that were used in previous development efforts are not used. Instead, custom 3D-printed container ("cup") are created, where workpieces of different types (indicated by their different colours) can be placed upon. In this case study, workpiece can have red or green colour as shown in Fig. 2.

Fig. 2 Red and green workpiece and custom container/cup

- In this case study, Mobile Production Platform (MPP) is introduced to support production. The MPP is constructed from a Festo Robotino mobile robot equipped with a conveyor to support logistics. Here, the Festo Robotino is programmed using Robotino View language, and the robot platform is connected to the EnAS demonstrator via an IEC 61499-enabled embedded controller called ICEBlock (which will be described in the subsequent items). (Fig. 3).

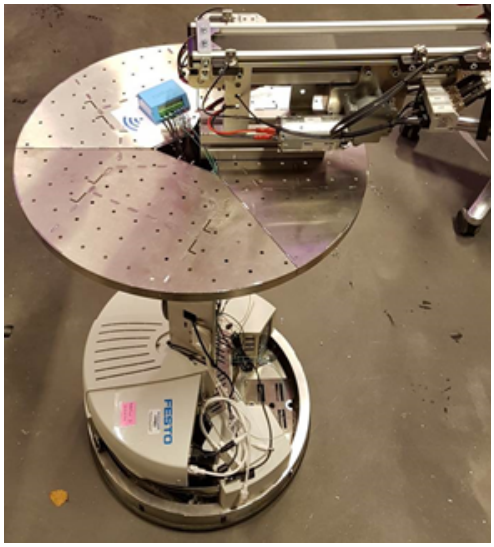

Fig. 3 Mobile production platform based on Festo Robotino

- Two corners of the EnAS demonstrator's conveyor system are opened to become "docking points" for the MPP to transfer workpiece in and out of the EnAS demonstrator. This is different than the previous development phases, where all corners of the EnAS demonstrator's conveyor system are closed as a closed-loop assembly line.
- A diverting actuator ("gate") is installed on one of the two "opened" corners of the EnAS demonstrator (top sides of sections S1 and S5 on Fig. 6). The gate can be opened or closed at any point in time. The gate can

be opened when workpiece is to be transferred to the MPP and the MPP is currently docked to the corner, otherwise it is closed if the workpiece should remain on the conveyor system.
- The gripper stations are not used.
- The Nxt DCS Mini controllers which were used in the previous development have been replaced with wireless-capable and internet-addressable embedded controller called IceBlock. IceBlock has NXT 61499 runtime running on Linux OS, which allows it to execute IEC 61499 applications developed using Nxt Studio tool.

Production Scenario. Production is started "on demand" when an order for a particular product has been placed. A product consists of a cup and a workpiece, which can be in red or green colour. Multiple orders are processed in batches, i.e., subsequent order can be placed only when the previous one is completed. The EnAS assembly line is supplied with the correct number of cups by the Robotino and the assembly of a product is deemed complete only when a product is valid, e.g., it has a correct workpiece colour. This case study considers the following assumptions:

- EnAS shall have simultaneously at most two cups on it, at any given time.
- Workpieces come only in two colours: red or green.
- Sledge 2 is always able to provide coloured workpieces. It is always restocked in time.
- Sledge 1 can always take workpieces from the cup on the conveyor. When the slots in the sledge are occupied, they will be emptied.
- Cups are supplied by the MPP, at most two cups, to the EnAS assembly line.

A video demonstration on how some production scenarios in the case study are performed can be seen in https://www.youtube.com/watch?v=Rp6lfUOyQ1o.

4 Software Design and Implementation

The software control logic defining operation of the entire production process in the EnAS extended by mobile production unit is implemented as an application in IEC 61499 function blocks (FBs) using NxtStudio tool from NxtControl, with the exception of the Festo Robotino which is programmed in Robotino View graphical language tool from Festo and color sensing which is implemented on a raspberry pi (note that this can be implemented in function blocks, the reason we used raspberry pi is to show how function block can interact with external services, in general off-the-shelf solution).

4.1 Product-driven software design pattern

The paper presents a five-layer design and implementation pattern as shown in Fig 4. The pattern described works on the concept of services and service providers, where every small action/activity is performed by a minimalistic reusable function blocks. Each of these function blocks is considered to be a service provider of some predefined service. The goal of the design pattern is to describe how to design function block applications such that they are reusable and are easy to scale with minimum effort.

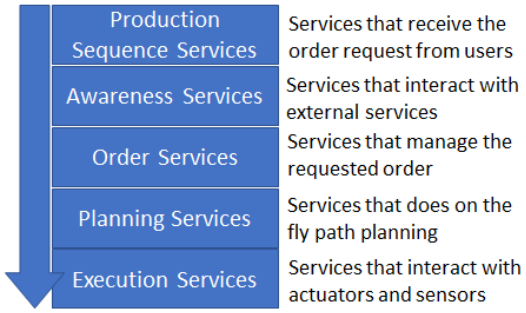

Fig. 4 Five-Layer design pattern for function block composition for distributed control software

This design pattern is inspired and builds on the design patterns presented in [8]. According to [8], the design pattern in this paper can be classified as a compositional pattern. However individual layers themselves use design patterns in [8] to design the function blocks in respective layers.

- ***Production Sequence Services***: In this layer, the function blocks are responsible for defining different products that users can choose to produce in the system, they define services that hold recipes for production of custom user products. The presented design pattern suggests that there is one function block responsible for handling each product requirements ("recipes"). In our use case, there are two function blocks in this layer, the "Produce Green" and "Produce Red" function blocks representing the two different product recipes that can be produced.
- ***Awareness Services***: This layer implements interfacing services to interact with other services described in non IEC 61499 environment. For example, the service that determines the colour of the work pieces (which in this case study is based on Python) or services that provides docking and undocking of robot services (which is based on Robotino View). In short, the function blocks that interact with systems that are not programmed in IEC 61499 are classified to be in this layer. The function blocks can use various communication methods, such as OPC-UA, REST API and stan-

dard socket communications. In Fig 4 we refer to such services as external services.

- ***Order Services***: In this layer, the services that interact with the user are designed and implemented. All the HMI actions related to order management fall under this layer. In this case study, the HMI function blocks that show order buttons and status messages are part of this layer.
- ***Planning Services***: This layer implements scheduling and path planning services. In our use, the services that the work piece cup takes in the production line, services that are responsible for scheduling of the conveyor operations and services that decide which of two jacks (jack1 or jack2) will be used in the work-piece production are classified to be part of this layer.
- ***Execution Services***: includes function blocks that interact with the actuators and sensors. These are the services whose simple operation is to turn on/off actuators that the Planning Services layer function blocks decided to be used in production of a product.

4.2 Implementation of design pattern for our case study

Fig. 5 shows the mapping of the different layers and the function block instances of the proposed design pattern and the use case.

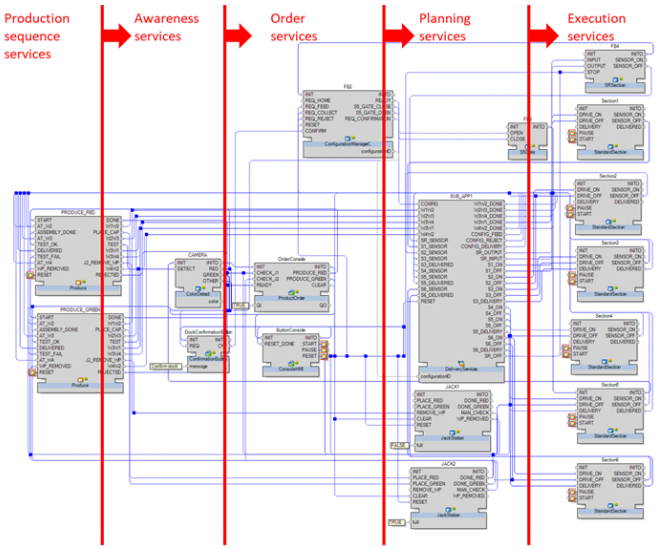

Fig. 5 Mapping of different function blocks to 5 layers of the proposed design pattern

The conveyor section mounted on the mobile platform and the diverter allow for on-demand reconfiguration of the production system's transport line during production process, and the transport system's logic is heavily relying on transportation service concept. Fig. 6 shows a schematic picture of EnAS transport system. On this picture, one can see four "waypoints": w1-w4, that correspond to places where a workpiece should be delivered for some action. For example, w1 is a terminal input-output point, w2 - product assembly, w3 - colour check, w4 - disassembly (of a product with wrong coloured workpiece). S1-S6 and SR represent low-level transport services, related to individual conveyor sections, and on top of that, five "planning" services are deployed to control transportation between waypoints w1-w2, w2-w3, w3-w1, w3-w4 and w4-w2.

Planning services are also responsible for requesting certain transport line

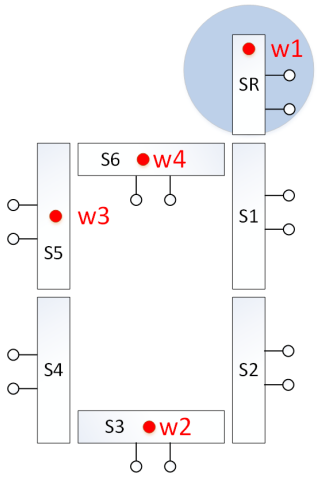

Fig. 6 EnAS transport system

configurations from configuration manager. For example, when w3-w1 transportation is requested, the corresponding planning service requests such configuration, where the mobile platform is in front of section S5 and diverter is open. Or w3-w4 transportation requires the diverter to be closed and at the same time is indifferent to the mobile platform position.

5 Conclusion and Future work

The paper presented a design pattern for design and implementation of IEC 61499-based distributed control application for a production line and was

demonstrated on a laboratory scale mechatronic system. By classifying and categorizing the different functional blocks that implement specific system requirements, the paper shows how it is to scale and also adopt new physical layouts quickly. The proposed design pattern is an extension of existing design patterns reported in the previous works.
Future work will take to extend this pattern to other domains and pillars of the factory floor such as SCADA and MES systems. Future work will also define the pattern using formal methods, so that formal verification of systems designed using this pattern can be easily verified. The product driven pattern may also be investigated in different programming language setting which have underlying formal model of computations, e.g., Service Oriented SystemJ (SOSJ) [22] [23].

Acknowledgment

The authors would like to thank Mr. Vesa Korhonen, the EEK department technician who have helped us a lot in technical and logistical support. The authors would like to also acknowledge our industrial partners who have provided their support on this project, to name a few, NxtControl that has provided NxtStudio software and to Festo that has provided a space to exhibit this project during the ScanAutomatic 2018 event in Gothenburg, Sweden. The part of the project was funded by European project DAEDALUS (H2020 Grant Agreement n°: 723248) and Aalto Factory of the Future initiative from School of Electrical Engineering of Aalto University.

References

1. D. Trentesaux and A. Thomas, *Product-Driven Control: Concept, Literature Review and Future Trends.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 135–150. [Online]. Available: https://doi.org/10.1007/978-3-642-35852-4_9
2. "Programmable Logic Controllers - Part 3: Programming Languages, IEC Standard 61131-3," 2013.
3. "IEC 61499-1: Function Blocks Part 1: Architecture," 2012.
4. C. Gerber, M. Hirsch, and H.-m. Hanisch, "Automatisierung einer energieautarken Fertigungsanlage nach IEC 61499," *atp magazin*, vol. 51, no. 03, pp. 44–52, 2013.
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, ser. Addison-Wesley Professional Computing Series. Pearson Education, 1994. [Online]. Available: https://books.google.se/books?id=6oHuKQe3TjQC
6. B. Vogel-Heuser, J. Fischer, S. Rösch, S. Feldmann, and S. Ulewicz, "Challenges for maintenance of PLC-software and its related hardware for automated production systems: Selected industrial Case Studies," in *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on.* IEEE, 2015, pp. 362–371.

7. B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy, "Evolution of software in automated production systems: Challenges and research directions," *Journal of Systems and Software*, vol. 110, pp. 54–84, 2015.
8. S. Patil, D. Drozdov, and V. Vyatkin, "Adapting Software Design Patterns To Develop Reusable IEC 61499 Function Block Applications," in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*. IEEE, 2018, pp. 725–732.
9. J. H. Christensen, "Design patterns for systems engineering with IEC 61499," in *Verteilte Automatisierung - Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung (VA2000)*, 2000, pp. 63–71.
10. A. Zoitl and T. Strasser, Eds., *Distributed control applications: guidelines, design patterns, and application examples with the IEC 61499*. CRC Press, 2016, vol. 9.
11. M. Bonfè, C. Fantuzzi, and C. Secchi, "Design patterns for model-based automation software design and implementation," *Control Engineering Practice*, vol. 21, no. 11, pp. 1608–1619, 2013.
12. D. Brandl, *Design Patterns for Flexible Manufacturing*, ser. EngineeringPro collection. ISA, 2006. [Online]. Available: https://books.google.se/books?id=136rPmt-K5UC
13. F. Serna, C. Catalán, A. Blesa, and J. M. Rams, "Design patterns for Failure Management in IEC 61499 Function Blocks." in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*. IEEE, 2010, pp. 1–7.
14. G. Cengic, O. Ljungkrantz, and K. Akesson, "A framework for component based distributed control software development using IEC 61499," in *Emerging Technologies and Factory Automation, 2006. ETFA'06. IEEE Conference on*. IEEE, 2006, pp. 782–789.
15. V. Vyatkin, S. Karras, and T. Pfeiffer, "Architecture for automation system development based on IEC 61499 standard," in *Industrial Informatics, 2005. INDIN'05. 2005 3rd IEEE International Conference on*. IEEE, 2005, pp. 13–18.
16. R. Hametner, A. Zoitl, and M. Semo, "Automation component architecture for the efficient development of industrial automation systems," in *Automation Science and Engineering (CASE), 2010 IEEE Conference on*. IEEE, 2010, pp. 156–161.
17. V. Vyatkin, "Intelligent mechatronic components: control system engineering using an open distributed architecture," in *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA'03. IEEE Conference*, vol. 2. IEEE, 2003, pp. 277–284.
18. M. Sorouri, S. Patil, and V. Vyatkin, "Distributed control patterns for intelligent mechatronic systems," in *Industrial Informatics (INDIN), 2012 10th IEEE International Conference on*. IEEE, 2012, pp. 259–264.
19. U. D. Atmojo and V. Vyatkin, "A design pattern for systems composed from intelligent mechatronic modules with wireless communication: a case study," in *2019 IEEE 24th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019 (Accepted).
20. L. Gröhn, S. Metsälä, M. Nyholm, L. Saikko, E. Väänänen, K. Gulzar, and V. Vyatkin, "Manufacturing system upgrade with wireless and distributed automation," *Procedia Manufacturing*, vol. 11, pp. 1012 – 1018, 2017, 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2351978917304158
21. U. D. Atmojo, K. Gulzar, V. Vyatkin, R. Ma, A. Hopsu, H. Makkonen, A. Korhonen, and L. T. Phu, "Distributed control architecture for dynamic reconfiguration: Flexible assembly line case study," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, May 2018, pp. 690–695.
22. Z. Salcic, U. D. Atmojo, H. Park, A. T. Chen, and K. I. Wang, "Designing dynamic and collaborative automation and robotics software systems," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 1, pp. 540–549, Jan 2019.
23. U. D. Atmojo, Z. Salcic, and K. I. Wang, "Dynamic online reconfiguration in manufacturing systems using sosj framework," in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, July 2016, pp. 695–698.