This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Atmojo, Udayanto Dwi; Vyatkin, Valeriy

A Review on Programming Approaches for Dynamic Industrial Cyber Physical Systems

Published in:

Proceedings of the IEEE 16th International Conference on Industrial Informatics, INDIN 2018

DOI:

[10.1109/INDIN.2018.8471945](https://doi.org/10.1109/INDIN.2018.8471945)

Published: 24/09/2018

Document Version

Peer reviewed version

Please cite the original version:

Atmojo, U. D., & Vyatkin, V. (2018). A Review on Programming Approaches for Dynamic Industrial Cyber Physical Systems. In *Proceedings of the IEEE 16th International Conference on Industrial Informatics, INDIN 2018* (pp. 713-718). [8471945] (IEEE International Conference on Industrial Informatics). IEEE.
<https://doi.org/10.1109/INDIN.2018.8471945>

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.

This is the accepted version of the original article published by IEEE.

© 2018 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

A Review on Programming Approaches for Dynamic Industrial Cyber Physical Systems

Udayanto Dwi Atmojo, Valeriy Vyatkin

Dept. of Electrical Engineering and Automation, Aalto University, Finland

Email: udayanto.atmojo@ieee.org; vyatkin@ieee.org

Abstract—Industrial automation and control systems are going through a big change, adopting the modern computing and Internet technologies to give completely new agility and adaptability properties to production plants, transport and energy systems and infrastructures, often referred to as Industrial Cyber Physical Systems (ICPS). In manufacturing, they are expected to cope with product customizations, changes in the market or supply chain, or failures. However, the wide adoption of ICPS requires rethinking approaches to their design and programming. This article reviews the state of the art of programming approaches for ICPS and analyses their capabilities, in the hope that it will help researchers to identify potential opportunities of future development of software solutions for ICPS.

Keywords—*industrial cyber physical systems; industrial automation; reconfigurable manufacturing; software programming approaches*

I. INTRODUCTION

Industrial manufacturing is experiencing currently the next industrial revolution, popularly known as the Industry 4.0 which envisages that industrial machines and plants will include plethora of interconnected computing devices interacting while performing decision making and altering the physical world. Such systems are referred to as the Industrial Cyber Physical Systems (ICPS). Compared to traditional manufacturing systems which have simpler centralized computing architecture e.g. implemented with programmable logic controller (PLC), the ICPS paradigm move towards having (potentially Internet accessible) distributed controllers, where each is connected to an individual manufacturing machine or a part thereof.

According to [1], one challenge for ICPS is that production systems will have to adapt (ideally in real time) to changes in the supply chain and market and collaborate with ultra-flexibility potentially beyond company boundaries. This opens opportunities of enabling swift production according to individual customers' specifications and optimization in production procedures by organizing manufacturing/production units owned by different operators. Since the behaviour of manufacturing/production units (e.g. which include embedded controllers, sensor and actuator nodes/SANs, robots) are completely controlled by the software entities that run on them, the challenges and requirements of the design and programming of ICPS will be different to traditional industrial manufacturing systems and they have to be put into consideration for future software solutions.

This paper looks at the existing programming approaches amenable for ICPS and identifies their strength and weaknesses. The paper is organized as follows. Section II presents a

motivating example of an ICPS, which will be used to identify the programming challenges and requirements of ICPS described in Section III. Section IV presents the state of the art of programming approaches and assesses them with regard to the considered programming requirements and challenges, and finally, Section V concludes the article.

II. MOTIVATING EXAMPLE

Let's consider an example of integrated factory floor illustrated in Fig.1. In Fig.1, there are several facilities indicated as F1, F2, and F3. Geographically, these facilities may be located at the same vicinity or even more remotely. F1 is a facility which is mainly used to store raw materials and manufactured products. F2 provides an environment mainly for office-related activities, however, some manufacturing amenities are also stationed in F2, for example, 3D printers. F3 is a manufacturing facility which has manufacturing devices and machineries that perform manufacturing operations. These facilities utilize state of the art technologies which include SANs, embedded controllers, and mobile robots to achieve production and safety requirements. For example, facility F1, F2, and F3 employ stationary SANs S1-S8 equipped with sensors such as light intensity, temperature, proximity detectors, or cameras to monitor the ambient environment and detect certain events, e.g., occurrence of fire.

Apart from stationary SANs, terrestrial SANs such as TS1 and TS2 equipped with presence sensors or cameras are also utilized which can navigate around the environment to perform security checks. Also equipped with their own sensors, flying aerial SANs are also deployed and possess additional advantages in their degree of freedom of movement compared to terrestrial SANs. There are also service robots such as SR1 which assist in maintenance and office-related duties. F3 have embedded controllers run software entities which control the manufacturing machines to produce products, e.g., bottle drinks.

Some of the software entities may directly aim at implementing particular behaviours of the physical system, while others can play a purely supporting role, e.g., performing computations or enabling communications and interoperability. The current trends in software design emphasize autonomy of software entities through their encapsulation to components of sorts, often referred to as functional blocks [2], agents [3], services [4], or concurrent software entities such as those named reactions [5] or clock domains [6]. This is done to increase the reuse potential of software and to increase the overall system flexibility.

Fig. 1. An example of ICPS.

The use of Internet and cloud opens opportunities for interoperation between elements in the facilities, for example, remote monitoring/control, resource sharing, and supply chain management. Also, features of the Internet and cloud facilitate easier integration of new facilities or removal/detachment of existing facility and allow for elements intra and inter facilities to interoperate and form systems dynamically. The behaviours of the aforementioned elements (i.e. robots, SANs, manufacturing machines) in the systems are governed by the software entities which run on their corresponding execution platforms. Therefore, the amount and extent of effort in designing and programming systems like the motivating example will be affected by the underlying software programming framework.

III. IDENTIFIED PROGRAMMING REQUIREMENTS

Based on the example, the following programming requirements can be identified.

1) Concurrency

Software entities in systems like the motivating example are naturally running in parallel and deployed on more than one execution platforms (distributed). Thus, the software programming approach should facilitate programmers to design and deploy concurrent software entities potentially in more than one execution platforms.

2) Dynamicity

By taking into account the work in [7], the term dynamicity can be associated to the need to deal with dynamic behaviours and uncertainties. These uncertainties can come from either or both the physical world and software world, and they can be associated or will eventually lead to changes in the presence of software entities, i.e., software entities entering or leaving the system during runtime. From the software perspective, creating

new software entities, resuming the execution of suspended software entities, and inbound migrating software entities in the new location achieve addition/introduction of software entities. Suspending the execution of software entities, (permanently) terminating the execution of software entities, and outbound migrating software entities to a new location lead to removal of software entities. In general, software migration is categorized into two types, weak and strong migration. Weak migration permits the movement of software entities without retaining its execution and data state, while strong migration corresponds to the movement of software entities from one to another location with its execution and data state retained.

In addition, also by taking into account the work in [8], dynamicity is highly related to scalability, disturbance handling, and flexibility. Referring to the motivating example shown in Fig.1, the introduction or detachment of facilities or their elements therein (i.e. manufacturing machines, mobility of terrestrial and aerial SANs and robots) can cause the addition of new or removal of existing software entities.

Disturbances can be caused by failures, e.g., either in hardware controllers or in software controllers (e.g. due to errors, bugs, deadlocks), changes in production demand or manufacturing throughput, or human errors.

Flexibility is related to the need to cope with product customizations, e.g., variation in product designs, sizes, and mixes. Such product customizations may require changes ('updates') of existing software entities.

To cope with changes in system composition, having individual software entities which are aware of other software entities' presence and preferably also their capabilities is considered desirable. In this paper, this feature is referred to as *software environment awareness*.

3) Composability

Potentially large-scale and complex systems like the motivating example is subject to addition or removal of embedded controllers, robots, SANs. Thus, a programming approach which enables modular software design paradigm is crucial. Modular software design paradigm facilitates easy adjustments in case of changes or modifications.

IV. STATE OF THE ART AND ASSESSMENTS

The current state of the art encompasses a number of different software programming paradigms. Among the most popular software approaches are the Multi Agent Systems (MAS), Service Oriented Architecture (SOA), "Automation domain-specific languages", and those originally developed with inherent formal Model of Computations (MoC).

A. Multi Agent Systems (MAS)

MAS software paradigm provides abstraction where software entities are encapsulated as *agents*. Agents are naturally loosely-coupled, and some are designed to react (respond) to incoming inputs or events from the environment. MAS paradigm supports the execution of concurrent agents and facilitates their communication with each other through defined communication interfaces. Examples of MAS-based programming frameworks are JADE [9] and JIAC [10]. Especially in the industrial manufacturing domain, among the state of the art of MAS-based programming approaches, JADE is considered as the most popular and prominently used.

Fundamentally, MAS paradigm supports concurrency that agents can be executed in parallel and distributed across different computing machines. For example, with regard to JADE, agents may be executed by the same device or distributed across different runtime environments, where one computing machine possibly contains one or more runtime environments.

Some MAS frameworks, including JADE, enable hierarchical composition of software entities and allow for modular software design paradigm. For example, JADE agents execute concurrently, with individual agents potentially contain one or more concurrent ‘sub-behaviours’ called *behaviours* in JADE terminology. In addition, some MAS frameworks, particularly those which fully comply with the Foundation for Intelligent Physical Agents (FIPA) standard such as JADE and JIAC, have considerable support in coping with dynamicity. These MAS frameworks enable creation, suspension, resumption, termination, and migration of software agents. However, there are no built-in mechanisms in these MAS-based frameworks which assist in achieving proper system composition dynamically.

B. Service Oriented Architecture (SOA)

According to the SOA paradigm, service is defined as discrete functionality which can form applications or systems. The SOA paradigm offers the concept of loose-coupling, reconfigurability, and flexibility which allows service entities to be composed or orchestrated during runtime to create different systems. Looking at the SOA paradigm, a number of key features provide support in addressing dynamicity.

- Loose coupling: Individual service entities may be executed on their own without relying or dependent on other service entities.
- Mechanisms of information exchange: SOA defines an architecture such that service entities are informed (dynamically) of what and which services are ‘present’ and can be used. The information exchange mechanism provided by SOA paradigm supports software environment awareness.
- Flexible orchestration: With the SOA paradigm promoting loose coupling concept, systems may not necessarily be formed (or composed) during design time, but dynamically during runtime.

Several existing SOA-based software approaches are based on state of the art SOA specifications, for example, MCC-OSGi [11] which is based on the OSGi specification, UPnP Robot Middleware [12], which is based on the UPnP specification and designed specifically for robotic domain, SOA4D [13] which is based on the DPWS specification, and Arrowhead which utilizes DNS Service Discovery [14]. There is also the OPC-UA standard which defines its own SOA specification. A number of software libraries and frameworks based on OPC-UA have been developed, some have commercial license (e.g. developed by ProSys [15]) while some others are open source (e.g. open62541 [16]).

SOA-based approaches focus mainly on providing information exchange mechanisms which enable service entities to advertise, discover, and use services. Service entities are naturally concurrent, where they can be run in parallel on the same or also distributed across different computing machines. SOA paradigm doesn’t provide any specifications or describe any standards in dealing with dynamicity beyond the loose-

coupling feature and the information exchange mechanisms related to service discovery, advertisement/registration, and invocation.

Some SOA frameworks like WS4D provide the mechanism to dynamically introduce and remove service entities during runtime, however they don’t provide further support in allowing dynamic suspension, resumption, and migration. SOA frameworks typically leave how services are composed together at the misery of the programmers. One exception is the Arrowhead framework which provides service orchestration function. However, different from some SOA frameworks like WS4D and the SOA4D, Arrowhead and OPC-UA do not specify how software entities are described, that one may not consider them as software programming approaches.

C. Automation domain specific languages

In the manufacturing domain, among the most prominently known and used domain specific languages are the IEC 61131-3 [17] and IEC 61499 [18]. The IEC 61131-3 is dominating the automation world, but it is designed for centrally controlled systems. Thus, programmers are very likely to face difficult challenges when using this approach for modern ICPS, where it is expected that they are often distributed and decentralized. In addition, the standard also lacks features to cope with dynamicity.

On the other hand, the IEC 61499 is designed with consideration for distributed setting. Currently, there are more PLCs which are compatible with the IEC 61131-3 than the IEC 61499, although the industrial adoption of IEC 61499 is growing. Hierarchical composition is permitted in IEC 61131-3 and 61499, which can be achieved in function block form through the creation of the so-called compound or composite function blocks from multiple basic function blocks. Meanwhile, in MAS-based approaches like JADE, individual agents can contain one or more *behaviours*, with individual *behaviours* can contain one or more ‘sub-behaviours’. Compared to composite/compound function blocks and basic function blocks which are of the same encapsulation, agents and *behaviours* are not.

The IEC 61131-3 was designed to comply with traditional industrial manufacturing systems which are typically centralized, ‘static’ (i.e. remain unchanged during the entire time of their operation) and focus more on ‘mass production’ of the same products, thus very limited programming features/constructs are available for dealing with dynamicity. Meanwhile, the IEC 61499 have been experiencing improvements which also consider requirements and challenges of modern ICPS. It is possible to create, delete, start, and terminate IEC 61499 function blocks dynamically without restarting the system (‘on the fly’) through the IEC 61499 device manager. However, depending on how the IEC 61499 applications are implemented, particularly in certain cases when composite function blocks are involved, relatively simple changes such as replacing, adding, or removing individual function block within composite function block can become quite difficult and complicated for programmers [19]. In light to such challenges, there are works which propose solutions with regard to dynamic reconfiguration in IEC 61499. For example, the approach used in the eCedac project [20] proposes a composite function block that handles state transitions during the reconfiguration process and performs runtime checks to

detect failures. However, scenario for migration is not considered. Another work described in [21] proposed a new function block namely the Reconfigurable Function Block, however reconfiguration scenarios are limited depending on the individual execution control charts which are hardcoded in individual Reconfigurable Function Blocks (where each ECC corresponds to each possible scenario).

Originally, the IEC 61499 doesn't provide built-in features to support migration of function blocks (e.g. from one controller to another). An approach to achieve function blocks migration in IEC 61499 is presented in [22], however in this approach, migration cannot be triggered by the corresponding function blocks. On another note, programmers themselves are responsible to ensure that proper composition of function blocks are achieved when changes are made in an IEC 61499 application 'on the fly', which can also become challenging as an IEC 61499 application becomes more complex and increases in scale.

D. Formal MoC-based Approaches

Formal MoC-based approaches utilize formal semantics and MoC which assist programmers in designing software entities which are amenable for formal verification. Formal verification helps in minimizing programmers' efforts in software testing and avoiding errors/bugs in their implementation. Some examples of formal semantics and MoCs-based software approaches are Esterel [5], DSystemJ [23], SystemJ [24], LibDGALS [25], and LibGALS [26].

Formal MoC-based approaches, like the aforementioned ones, support concurrency. Esterel supports synchronous concurrency, where software entities execute in lock-step in accordance to a notion of time of logical discrete clock (*tick*). However, it is not designed for distributed setting and doesn't support asynchronous concurrency (i.e. where concurrent software entities have their own notion of tick). Also, Esterel programs are usually compiled to C/C++ making it platform dependent. The use of C/C++ means programmers are responsible for ensuring that memory allocation is handled properly during runtime, which can become challenging especially in complex systems with distributed setting. Esterel is also not designed for coping with dynamicity: E.g. no support is given to allow for introduction and removal of software entities during runtime. Similar issues are present in LibGALS and LibDGALS due to the use of C/C++. However, LibDGALS has the advantage over Esterel and LibGALS, that it provides certain degree of support to handle dynamicity which includes creation and termination of software entities. Esterel, LibGALS, and LibDGALS support hierarchical composition, where in Esterel, synchronous software entities (called *reactions*) can contain multiple 'sub'-*reactions*. Meanwhile, LibGALS and LibDGALS have asynchronous software entities (called *clock domains*) at the top level, where individual clock domains may have hierarchical *reactions*.

SystemJ and DSystemJ languages are similar to LibGALS and LibDGALS as they support both asynchronous and synchronous concurrency and based on the same formal MoC namely GALS (Globally Asynchronous Locally Synchronous) [6]. Hierarchical composition with clock domains on the top-level and reactions on the bottom-level is also supported. However, unlike LibGALS and LibDGALS, SystemJ and DSystemJ are based on the use of Java. SystemJ and DSystemJ

code is compiled into Java which enhances code portability as compared to LibGALS, LibDGALS, and Esterel. Also, the use of Java saves the programmers' effort on handling memory allocation during runtime. The formal MoC-based approaches provide abstracted communication mechanisms for software entities to communicate among each other or with the environment. All of the considered formal MoC-based approaches utilize mechanism called *signal* to interact with the environment and between synchronous software entities, meanwhile the mechanism called *channel* is used by SystemJ, DSystemJ, LibGALS, and LibDGALS for communication between asynchronous software entities. Abstracted communication mechanisms such as *signals* or *channels* allow programmers to focus on system-level function instead of implementation details of communication and physical interfacing.

Both SystemJ and LibGALS are similar in that both are designed for the design and programming of 'static' systems and lack programming features to handle dynamicity, while DSystemJ and LibDGALS are dynamic extensions from SystemJ and LibGALS, respectively. DSystemJ supports creation and weak migration of software entities, while LibDGALS support creation and termination of software entities. Still both DSystemJ and LibDGALS lack many features for supporting proper dynamic system composition and software environment awareness.

E. Combination of Paradigms

Apart from the aforementioned paradigms, there are efforts in developing new paradigms by synergizing different paradigms in order to benefit from the features provided by the individual paradigms at the same time. In this section, some of such combinations will be briefly discussed.

1) MAS and SOA

There have been attempts to combine both SOA-based approaches and MAS-based approaches. Looking at the state of the art, three methodologies are identified in which the combined SOA and MAS is achieved. First is to wrap (software) agents as service entities, for example, by wrapping agents as web services [27]. Second is to introduce what is called as the Service Oriented Multi Agent System (SOMAS), such as Thomas [28]. Third is to introduce a proxy function to enable interoperability. One example which uses this methodology is JADE-JBossESB [29]. With the development of the FIPA standard, FIPA-compliant MAS-based frameworks like JADE and JIAC V [30], the latest version of JIAC, incorporate some degree of SOA-like functions using the Directory Facilitator (DF), which allows agents to register their services and perform query to the DF to obtain list of registered services.

The introduction of the SOA paradigm into MAS-based frameworks adds further support in dealing with dynamicity, which aims at enabling *software environment awareness* and introduce loose-coupling found originally in SOA. Still, it is the responsibility of programmers to ensure the composition is achieved properly.

2) Automation domain-specific and SOA

With appealing features of loose coupling and enabling reconfiguration, the SOA paradigm attracts attention and leads to efforts in combining SOA and the domain specific approaches of IEC 61131-3 and 61499. For example, the work [31] proposes

a conceptual IEC 61499-compatible approach that adopts the SOA paradigm. The work [32] implemented SOA using IEC 61499 function blocks, however no SOA functions were introduced at the runtime-level. Then, the work [33] attempted to introduce SOA functions into IEC 61499 at the runtime environment level. Another approach in [34] used the DPWS stack to implement SOA in IEC 61499. Introducing the SOA paradigm to the IEC 61131-3 was also considered, but the fact that the IEC 61131-3 isn't designed for distributed setting hinders any significant developments in introducing SOA into the IEC 61131-3 [17]. Similar to the introduction of SOA into MAS approaches, the introduction of the SOA paradigm to the IEC 61499 adds the feature of loose coupling and incorporate *software environment awareness*. However, this approach doesn't introduce functions which support proper dynamic composition of IEC 61499 application. Programmers are still responsible in ensuring that such composition is achieved.

3) 'Automation domain-specific' and Formal MoC-based

While the automation domain-specific approaches like the IEC 61131-3 and 61499 are considered the most prominently used in PLC-based manufacturing systems, insufficient concern is given in ensuring that the standards have unambiguous semantics. In the attempt to alleviate this issue, there have been attempts to use formal semantics and model of computations. The work described in [35] makes use of the synchronous reactive semantics and applies them to the IEC 61499 by generating the corresponding Esterel code from IEC 61499 function blocks. Also, the attempt [36] uses the formal GALS MoC to model IEC 61499-based systems. Other formal MoCs, such as Net Condition/Event Systems [37] and timed automata [38], were also considered. Meanwhile, a few attempts consider introducing formal semantics and MoCs into the IEC 61131-3. An example is [39], which allow mapping of IEC 61131-3 into timed automata. The aforementioned efforts in incorporating formal MoCs into the IEC 61499 and IEC 61131-3 emphasize on enabling programmers to develop verifiable software entities based on these standards.

4) Formal MoC-based and SOA

Attempts are also being made in realizing the synergy between the SOA paradigm and formal MoC-based programming approaches. One successful attempt in this domain is the SOSJ framework [41], which extends the formal MoC-based language SystemJ with the SOA paradigm. The SOSJ framework has been further enriched with features that enable creation, suspension, resumption, termination, and migration of software entities 'on the fly' without requiring system restart and demonstrated in a manufacturing scenario [42]. The framework benefits from the features provided by SystemJ, which include the use of abstracted communication mechanisms of signal and channel, hierarchical composition with clock domains and reactions, and formal MoC that guarantees functional correctness and allows for verifiable software entities. However, the feature that support proper dynamic composition of systems is not present in SOSJ.

5) MAS and the automation domain-specific

The features offered by the MAS software paradigm attract people to use MAS-based programming approaches in the automation domain-specific languages environment. An example is the works [47], where software agents are utilized to support dynamic reconfiguration in IEC 61499-based

manufacturing scenarios. In these examples, some extent of formal model was used, however it is not inherent in the programming frameworks and applies only to certain extent, e.g., synchronization and coordination between software agents and the IEC 61499. This is different from the approaches in Section IV.D or IV.E.4, where the formal models in these approaches are inherent in the software frameworks and more extensive that they can guarantee the functional correctness of the software entities.

Both the holonic and MAS paradigm is also used alongside the IEC 61131-3, for example, in the work [48], where the ADACOR framework is used to implement holons in the IEC 61131-3 environment. The aforementioned works are based on the approach that essentially uses separate MAS (or MAS and holonic) and IEC 61499 and or IEC 61131-3 based programming frameworks. Interestingly, different combination of these individual approaches can introduce distinct combination of advantages and disadvantages in addressing the programming requirements due to their respective programming features and limitations. For example in [48], while MAS-based frameworks generally support introduction and removal of software entities during runtime, the use of IEC 61131-3 prevents the use of the above features.

V. CONCLUDING REMARKS

Because of space limitations, this paper is unable to present all developments in software solutions for ICPS, however the observed developments may expose possible ways along which the legacy technology would transform to satisfy new requirements.

REFERENCES

- [1] A. Hellinger and H. Seeger, "Cyber-Physical Systems. Driving force for innovation in mobility, health, energy and production," *Acatech Position Paper, National Academy of Science and Engineering*, 2011.
- [2] D. H. Hanssen, "Function Block Diagram (FBD)," in *Programmable Logic Controllers*, ed: John Wiley & Sons, Ltd, 2015, pp. 262-277.
- [3] J. Ferber, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*: Addison-Wesley Longman Publishing Co., Inc., 1999.
- [4] M. P. Papazoglou, "Service-oriented computing: concepts, characteristics and directions," in *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, 2003, pp. 3-12.
- [5] G. Berry and G. Gonthier, "The Esterel synchronous programming language: design, semantics, implementation," *Science of Computer Programming*, vol. 19, pp. 87-152, 1992.
- [6] D. M. Chapiro, "Globally-asynchronous locally-synchronous systems," PhD, Department of Computer Science, Stanford University, California, 1984.
- [7] R. Baldoni, M. Bertier, M. Raynal, and S. Tucci-Piergiovanni, "Looking for a Definition of Dynamic Distributed Systems," in *Parallel Computing Technologies*. vol. 4671, V. Malyshev, Ed., ed: Springer Berlin Heidelberg, 2007, pp. 1-14.
- [8] J. H. Christensen, "Holonic Manufacturing Systems: Initial Architecture and Standards Directions," in *Proceeding of 1st Euro Workshop on Holonic Manufacturing Systems*, 1994.
- [9] F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi, "JADE—a java agent development framework," *Multi-Agent Programming*, pp. 125-147, 2005.
- [10] M. Lützenberger, T. Konnerth, T. Küster, J. Tonn, N. Masuch, and S. Albayrak, "Engineering JIAC multi-agent systems," presented at the Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, Paris, France, 2014.
- [11] F. Houacine, S. Bouzeffrane, L. Li, and D. Huang, "MCC-OSGi: An OSGi-based mobile cloud service model," in *Autonomous Decentralized*

- Systems (ISADS), 2013 IEEE Eleventh International Symposium on*, 2013, pp. 1-8.
- [12] S. Ahn, K. Lim, J. Lee, H. Ko, Y. Kwon, and H. Kim, "UPnP robot middleware for ubiquitous robot control," in *The 3rd International Conference on Ubiquitous Robots and Ambient Intelligence (URAI 2006)*, 2006.
 - [13] (2017, 27 December). *Service-Oriented Architecture for Devices (SOA4D)*. Available: <https://forge.soa4d.org/>
 - [14] S. Cheshire and M. Krochmal. (2006). *DNS-Based Service Discovery*. Available: <http://www.dns-sd.org/>
 - [15] (2017, 27 December). *ProSys OPC*. Available: <http://www.prosysopc.com>
 - [16] (2017, 27 December). *open62541: an open source implementation of OPC UA* Available: <https://open62541.org/>
 - [17] K. H. John and M. Tiegkamp, *IEC 61131-3: Programming Industrial Automation Systems Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids*: Springer Publishing Company, Incorporated, 2010.
 - [18] V. Vyatkin, "The IEC 61499 standard and its semantics," *IEEE Industrial Electronics Magazine*, vol. 3, pp. 40-48, 2009.
 - [19] R. W. Brennan, P. Vrba, P. Tichy, A. Zoitl, C. Sünder, T. Strasser, *et al.*, "Developments in dynamic and intelligent reconfiguration of industrial automation," *Computers in Industry*, vol. 59, pp. 533-547, 2008/08/01/ 2008.
 - [20] M. Rooker, C. Sünder, T. Strasser, A. Zoitl, O. Hummer, and G. Ebenhofer, "Zero Downtime Reconfiguration of Distributed Automation Systems: The eCEDAC Approach," in *Holonic and Multi-Agent Systems for Manufacturing*, vol. 4659, V. Mafik, V. Vyatkin, and A. Colombo, Eds., ed: Springer Berlin Heidelberg, 2007, pp. 326-337.
 - [21] S. Guellou, A. Benzina, M. Khalgui, and G. Frey, "Reconfigurable function blocks: Extension to the standard IEC 61499," in *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, 2016, pp. 1-8.
 - [22] J. Yan, C. Pang, C. W. Yang, and V. Vyatkin, "Adaptable software components: Towards digital ecosystems and software evolution in the industrial automation domain," in *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, 2014, pp. 2512-2518.
 - [23] A. Malik, A. Girault, and Z. Salcic, "Formal Semantics, Compilation and Execution of the GALS Programming Language DSystemJ," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, pp. 1240-1254, 2012.
 - [24] A. Malik, Z. Salcic, P. S. Roop, and A. Girault, "SystemJ: A GALS language for system level design," *Computer Languages, Systems, & Structures*, vol. 36, pp. 317-344, 2010.
 - [25] W.-T. Sun, Z. Salcic, A. Girault, and A. Malik, "libDGALS: A library-based approach to design dynamic GALS systems," in *Industrial Embedded Systems (SIES), 2014 9th IEEE International Symposium on*, 2014, pp. 104-111.
 - [26] W.-T. Sun, Z. Salcic, and A. Malik, "LibGALS: A library for GALS systems design and modeling," in *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, 2010, pp. 107-112.
 - [27] M. N. Huhns, "Agents as Web services," *IEEE Internet Computing*, vol. 6, pp. 93-95, 2002.
 - [28] C. Carrascosa, A. Giret, V. Julian, M. Rebollo, E. Argente, and V. Botti, "Service oriented MAS: an open architecture," presented at the Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2, Budapest, Hungary, 2009.
 - [29] P. Vrba, M. Fuksa, and M. Klima, "JADE-JBossESB gateway: Integration of multi-agent system with enterprise service bus," in *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, 2014, pp. 3663-3668.
 - [30] M. Lützenberger, T. Küster, T. Konnerth, A. Thiele, N. Masuch, A. Heßler, *et al.*, "JIAC V: A MAS framework for industrial applications," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 2013, pp. 1189-1190.
 - [31] K. Thramboulidis, G. Koumoutsos, and G. Doukas, "Towards a Service-Oriented IEC 61499 compliant Engineering Support Environment," in *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on*, 2006, pp. 758-765.
 - [32] W. W. Dai, J. H. Christensen, V. Vyatkin, and V. Dubinin, "Function block implementation of service oriented architecture: Case study," in *Industrial Informatics (INDIN), 2014 12th IEEE International Conference on*, 2014, pp. 112-117.
 - [33] W. Dai, V. Vyatkin, J. H. Christensen, and V. N. Dubinin, "Bridging Service-Oriented Architecture and IEC 61499 for Flexibility and Interoperability," *IEEE Transactions on Industrial Informatics*, vol. 11, pp. 771-781, 2015.
 - [34] W. Dai, H. Wanqi, and V. Vyatkin, "Enabling plug-and-play software components in industrial cyber-physical systems by adopting service-oriented architecture paradigm," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, 2016, pp. 5253-5258.
 - [35] L. H. Yoong, P. S. Roop, V. Vyatkin, and Z. Salcic, "A Synchronous Approach for IEC 61499 Function Block Implementation," *Computers, IEEE Transactions on*, vol. 58, pp. 1599-1614, 2009.
 - [36] L. H. Yoong, G. D. Shaw, P. S. Roop, and Z. Salcic, "Synthesizing Globally Asynchronous Locally Synchronous Systems With IEC 61499," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, pp. 1465-1477, 2012.
 - [37] V. Vyatkin, "Execution Semantic of Function Blocks based on the Model of Net Condition/Event Systems," in *Industrial Informatics, 2006 IEEE International Conference on*, 2006, pp. 874-879.
 - [38] M.-P. Stanica and H. Guéguen, "A Timed Automata Model of IEC 61499 Basic Function Blocks Semantic," in *Proceedings of the 7th International Workshop on Discrete Event Systems*, Reims, France, 2004, pp. 385-90.
 - [39] E. P. Enoiu, D. Sundmark, and P. Pettersson, "Model-Based Test Suite Generation for Function Block Diagrams Using the UPPAAL Model Checker," in *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*, 2013, pp. 158-167.
 - [40] M. Pereplechikov, C. Ryan, K. Frampton, and H. W. Schmidt, "A Formal Model of Service-Oriented Design Structure," in *Software Engineering Conference, 2007. ASWEC 2007. 18th Australian*, 2007, pp. 71-80.
 - [41] U. D. Atmojo, Z. Salcic, and K. I. K. Wang, "SOSJ: A new programming paradigm for adaptive distributed systems," in *Industrial Electronics and Applications (ICIEA), 2015 IEEE 10th Conference on*, 2015, pp. 978-983.
 - [42] U. D. Atmojo, Z. Salcic, and K. I. K. Wang, "Dynamic Reconfiguration and Adaptation of Manufacturing Systems Using SOSJ Framework," *IEEE Transactions on Industrial Informatics*, vol. PP, pp. 1-1, 2018.
 - [43] A. Koestler, "Beyond atomism and holism—the concept of the holon," *Perspectives in Biology and Medicine*, vol. 13, pp. 131-154, 1970.
 - [44] H. Van Brussel, J. Wyls, P. Valckenaers, L. Bongaerts, and P. Peeters, "Reference architecture for holonic manufacturing systems: PROSA," *Computers in Industry*, vol. 37, pp. 255-274, 1998/11/01/ 1998.
 - [45] V. Botti and A. Giret, *ANEMONA: A Multi-agent Methodology for Holonic Manufacturing Systems*: Springer Publishing Company, Incorporated, 2008.
 - [46] P. Leitão and F. Restivo, "ADACOR: A holonic architecture for agile and adaptive manufacturing control," *Computers in Industry*, vol. 57, pp. 121-130, 2006.
 - [47] M. Khalgui and H. M. Hanisch, "Reconfiguration Protocol for Multi-Agent Control Software Architectures," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 41, pp. 70-80, 2011.
 - [48] K. Kruger and A. H. Basson, "Multi-agent Systems vs IEC 61499 for Holonic Resource Control in Reconfigurable Systems," *Procedia CIRP*, vol. 7, pp. 503-508, 2013/01/01/ 2013.