Automation Software Architecture in CPPS – Definition, Challenges and Research Potentials

Birgit Vogel-Heuser
Institute of Automation and Information Systems
TUM School of Engineering and Design
Core Member of MDSI, Member of MIRMI
Technical University of Munich
Garching, Germany
vogel-heuser@tum.de

Eva-Maria Neumann
Institute of Automation and Information Systems
TUM School of Engineering and Design
Technical University of Munich
Garching, Germany
eva-maria.neumann@tum.de

Juliane Fischer
Institute of Automation and Information Systems
TUM School of Engineering and Design
Technical University of Munich
Garching, Germany
juliane.fischer@tum.de

Marga Marcos
Department of Automatic Control and System Engineering
ETSI Bilbao
Basque Country University
Bilbao, Spain
marga.marcos@ehu.eus

Elisabet Estévez Estévez
Electronics and Automation Engineering Department
University of Jaén
Jaén, Spain
eestevez@ujaen.es

Giacomo Barbieri
Department of Mechanical Engineering
Universidad de los Andes
Bogotá, Colombia
g.barbieri@uniandes.edu.co

Lisa Sonnleithner
CDL VaSiCS, LIT CPS Lab, Johannes Kepler University Linz
Linz, Austria
lisa.sonnleithner@jku.at

Rick Rabiser
CDL VaSiCS, LIT CPS Lab, Johannes Kepler University Linz
Linz, Austria
rick.rabiser@jku.at

***Abstract*—The importance of a mature, maintainable design of automation software in Cyber-Physical Production Systems (CPPS) is continuously increasing since a growing proportion of the system functionality is implemented via software. As a result, system complexity is shifting more and more towards the software side. Quality characteristics such as flexibility, maintainability, and extensibility, whose importance is increasing in the context of Industry 4.0, are thus becoming harder to achieve. These challenges can only be solved through sound architectural design. While there is a common understanding of good software architecture and appropriate methods to achieve it in computer science, a suitable architecture definition for automation software that takes all relevant factors into account is still missing in the field of CPPS. Therefore, based on a workshop with international scientists with different perspectives on CPPS, this paper presents influencing factors on automation software architecture. On this basis, a joint definition for software architecture in CPPS, as well as challenges and future research potentials, are derived.**

Keywords—Cyber-Physical Production Systems, Automation Software Architecture, Programmable Logic Controllers

I. Motivation and Introduction

Cyber-Physical Production Systems (CPPS) represent a special type of mechatronic system characterized by high complexity in hardware and software. The development of automation software in CPPS is strongly influenced by other disciplines [1], mainly including electrics/electronics and mechanics, which usually determine the system architecture in a sequential development process, whereby software development often takes place last. CPPS often have lifetimes of up to several decades, and during that time, the automation software has to be maintainable and adaptable to changing requirements [2]. However, systematic change management is barely applied in industry, and software is still mainly reused using *Copy, Paste & Modify,* which leads to historically grown, error-prone legacy software in the long run [3]. Automation software in CPPS has to cope with boundary conditions that strongly differ from classical high-level language programming in computer science. To name only some of them, automation software in CPPS is often maintained and commissioned on the customer's site by technicians without software background, it has to fulfill hard real-time requirements to ensure safety and process stability, and it is usually written in specific languages defined by standards such as the IEC 61131-3, which also includes graphical languages. These differences hamper the applicability of approaches well-established in computer science without major adaptions.

Industry 4.0 (I4.0) deploys the tools provided by the advancements in operational, communication, and information technology to increase the levels of automation and digitization of production within manufacturing and industrial processes [4]. The complexity of CPPS and the corresponding automation software increases, integration with internet and wireless services becomes essential, and end-users demand functionalities of learning, computer vision, and speech recognition. Therefore, different opportunities and research areas arise from software and hardware perspectives. Efficient software development, including appropriate reuse and modularization strategies, is essential for companies to implement emerging technologies that arise in the context of I4.0 and, thus, stay competitive in a globalized market. However, cleanly modularized software architectures are hard to achieve due to the heterogeneity of software and the lack of generally applicable best practices and guidelines. The initial prerequisite for systematically analyzing and subsequently improving the structure of existing software is first a clear understanding of what defines software architecture in the field of CPPS and which additional factors beyond classic definitions from high-level programming must be considered. Up to now, there is a largely uniform consensus on the meaning

of software architecture in computer science [5] while architecture for automation software in CPPS has been little explored.

The main contribution of this paper is an overview of influencing factors on automation software architecture to demonstrate current challenges and the potentials for the application of established approaches from computer science and embedded systems. A definition of automation software architecture is formulated based on the preceding work and experience of renowned researchers from different software engineering domains, substantiated by industrial practitioners' experience. Finally, an outlook on future research directions to master the implementation of emerging technologies in the context of I4.0 and cyber-physical production systems is provided.

The remaining part of this paper is structured as follows: First, state of the art in analyzing software architecture is outlined in Section II. Subsequently, Section III introduces influencing factors on automation software architecture in CPPS, followed by a discussion of future challenges and potentials for architecture analysis in CPPS in Section V. Finally, the paper closes with a summary and an outlook in Section VI.

II. State of the Art

In the following, boundary conditions of CPPS and the state of the art in defining and analyzing software architecture are outlined.

A. Boundary conditions of CPPS

CPPS represent a special type of mechatronic systems, characterized by high *variability* [3] in hardware and software, lifetimes of up to several decades, and strong coupling of the involved disciplines, i.e., predominantly mechanical, electrical, and software engineering [2]. In machine and plant manufacturing, requirements often change even in the late development phases, and sometimes even during commissioning. Since automation software can be changed more easily at short notice than the automation hardware, an increasing amount of functionality and, thus, complexity is shifted to software.

CPPS are mainly controlled using *Programmable Logic Controllers (PLC)*, which are usually programmed according to the IEC 61131-3 standard. This standard defines five programming languages (three graphical and two textual ones) as well as three types of so-called *Program Organization Units (POU)* to structure the software into reusable elements [6]. While many PLC manufacturing companies support IEC 61131-3, it does not support highly distributed systems and dynamic reconfiguration [7]. Therefore, the standard *IEC 61499* defines a domain-specific modeling language that allows the separation of the application model from the hardware configuration. Thus, an application can be mapped to any number of devices, enabling the convenient design of distributed control applications. Fadhlillah et al. [8] outline a research agenda for variability management in IEC 61499 and propose an approach to manage variability in IEC 61499-based systems using delta-oriented variability modeling. Sonnleithner et al. [9] propose a catalog of bad smells for IEC 61499-based automation software, i.e., suboptimal structures or patterns in the software and discuss design patterns [10], which foster reuse and (re-)configurability in IEC 61499-based automation software. So far, however, IEC 61499 has barely found its way into industrial practice, but it is in the phase of early adopters [11].

PLCs show some particularities compared to classical embedded systems, e.g., they provide a cyclic execution model with monitored cycle times instead of a general-purpose operating system [12]. Additionally, programming PLCs often requires domain knowledge about the controlled processes and their interaction, which means that application engineers and technicians often program the software without fundamental software background [13]. In large-scale questionnaire studies in Germany [14–16] and in systematic case studies [17, 18], Vogel-Heuser et al. confirmed that these particularities lead to challenges in automation software that strongly differ from classical embedded systems applications. Thus, established software analysis techniques and quality definitions from computer science cannot be directly applied but must be customized and enlarged [19] to be applicable for automation software in CPPS.

Rabiser and Zoitl [20] discuss open research issues and goals and propose a research agenda towards mastering variability in software-intensive CPPS such as metallurgical or manufacturing plants. The authors also conclude that promising software engineering methods and tools, e.g., from software product lines, need to be adapted for the particular challenges in this domain. An exploratory case study [21] involving engineers and technical project managers of an industrial automation system for metallurgical plants provides empirical evidence on how CPPS are tested, commissioned, and operated in practice concluding that processes and tools need to support multi-disciplinary engineering across system boundaries. Furthermore, managing variability and evolution is crucial. Variability and complexity are still core research challenges [22]. Intentional and unintentional variability in functionality or quality attributes (e.g., performance) of software significantly increases the complexity of the problem and design space of systems. Variability becomes increasingly difficult to handle [23] due to the increasing size of software systems, new and emerging application domains, dynamic operating conditions, fast-moving and highly competitive markets, and more powerful and versatile hardware.

Model-based engineering represents an established means from computer science to enable efficient engineering and manage variability. Tools with advanced editing support can simplify working with large models and thus increase the benefits of the applied modeling language [24]. However, model-based software engineering is so far barely used in industrial automation software development, although Meixner et al. [25] present a reusable set of four case studies with varying complexity from three different domains, which can be used in research and practice to investigate variability modeling approaches.

The boundary conditions of automation software in CPPS lead to various challenges that are usually not considered by software analysis techniques established in high-level language programming. For example, although the object-oriented extension of IEC 61131-3 has been incorporated to the standard in 2013, it is up to now barely used in industry. Automation software architecture often follows different programming paradigms and design principles than embedded systems software. Furthermore, certain constellations of object-oriented language elements may lead to runtime issues in PLC software [26], thus

compromising compliance with hard real-time requirements, which are essential for CPPS. Currently available techniques and tools do not cover the boundary conditions of automation software. Existing version management tools, e.g., often do not support a detailed comparison of graphical programs or differentiate different change criticalities, which is particularly crucial if a software change may affect hardware components [27].

Hardware behavior is another predominant factor determining automation software: In CPPS, a significant number of faults are induced by hardware failures. Automation software can be utilized to mitigate this problem by detecting and managing the different failure events that may occur in the system. However, automation software design methodologies have mainly focused on the system's nominal behavior, marginally considering software generation due to hardware failures [28].

B. Defining and Analyzing Software Architecture

Software architecture [29] is one of the main subfields of software engineering research and intensively researched since the early 1990s. In computer science, several definitions for software architecture can be found and almost all include a common essence, i.e., software architecture description based on components and connectors [29–33]. Reussner et al., e.g., define software architecture as "a high-level abstraction of a software system, its components and their connections. Thus, architecture complements component definition which focuses on the individual components and their interfaces" [33]. Software architecture is formed by design decisions [34] that can be made reusable in the form of design patterns [35], i.e., reusable best practices for designing software architectures. Design decisions directly affect the system quality, e.g., usability [36] or maintainability [37]. The IEEE Standard 1471 identifies practices to establish a framework and vocabulary for software architecture concepts [31]. Available definitions focus on the description of the system structure from different viewpoints. Interactions of the parts a system is composed of are often at the center.

Recent advances and development practices such as continuous software engineering [38], cloud computing [39], microservices [40], and the Internet of Things (IoT) [41] are the main drivers of current software architecture research. Diverse approaches to design and analyze software architectures have been proposed in the last 30 years, including, e.g., many architecture description languages. In his recent work, Hasselbring [5] discusses software architecture's past, present, and future. In the past, the focus was on architecture description and reuse. Currently, domain-specific architectures are established, and the focus shifts towards quality attributes. Future work, according to [5], needs to resolve the tension between the agile software development and software architecture communities, especially in the context of DevOps and runtime adaptability.

In the context of the standards, IEC 61131 and IEC 61499, (micro-)service-oriented architectures and dynamically (re-) configurable architectures are recently heavily investigated as part of satisfying I4.0 requirements [42]. Software design patterns play a vital role in reusability and (re-)configurability of distributed control systems [43]. Sonnleithner et al. [10] provide an overview of existing work on designing software architectures in the automation software domain, including hierarchical design pattern-based approaches, service-oriented approaches, skill-based engineering approaches, microservices, and enterprise service bus. All these works aim to make automation software systems more modular and flexible, thereby improving reusability and (re-)configurability.

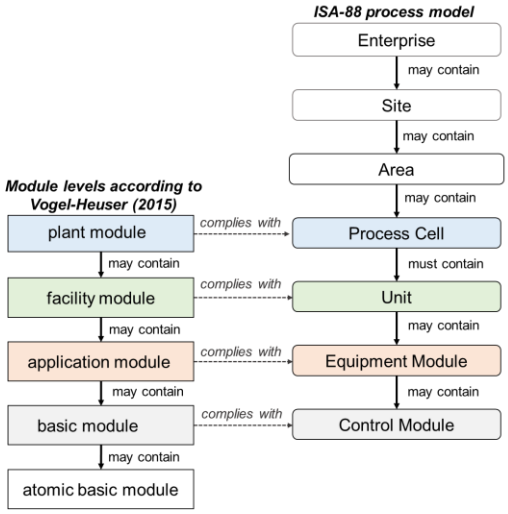

Fig. 1. Five level architectural model of Voegel-Heuser et al. [17] compared to ISA-88 process model [44]

For CPPS, IEC 61512 [44] defines three hierarchical models, including a physical model specifying a production system's architectural layers including *Control Modules* representing technical elements on the lowest level of the plant (e.g., sensors or servo drives) and *Equipment Modules* for controlling individual process sequences of the *Unit*. Vogel-Heuser et al. [17] observed a five-level module hierarchy in industrial automation software in CPPS compliant with the ISA-88 physical model [44]. The levels [17] range from plant modules controlling whole production plants to (atomic) basic modules reading sensors or controlling actuators (cf. Fig. 1). Although many companies are already implementing their software architecture following ISA-88, the standard focuses primarily on the process industry and packaging machine manufacturing with OMAC PackML [45].

III. Categorization of Influencing Factors on Automation Software Architecture

Based on the authors' previous work in software and system analysis and optimization, it can be identified that contrary to computer science, the architecture of automation software in CPPS depends on various influencing factors and boundary conditions (cf. Fig. 2). The following categories have been derived in a joint workshop in an international working group with leading international researchers in different fields of automation software. The categories serve as the basis to derive a joint definition of software architecture in Section IV and to discuss challenges and future research potentials in Section V.

A. Functional and Non-Functional Software Characteristics

Automation software in CPPS is characterized by functional and non-functional attributes that strongly differ from classical applications in computer science, and, thus, require particular consideration when formulating an architecture definition.

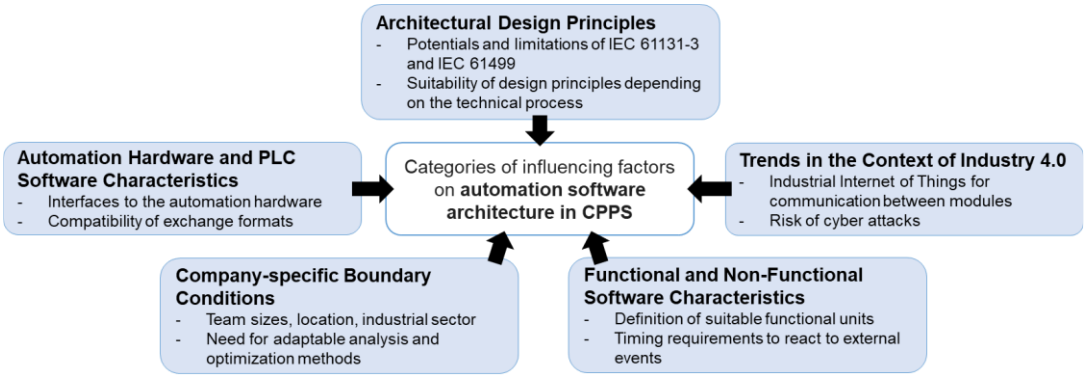

Fig. 2. Categories of influencing factors on automation software architecture in CPPS

1) Functional Characteristics

To meet the requirements during regular production, CPPS need to fulfill specific functional characteristics. Thereby, design decisions need to be taken on choosing an appropriate granularity of tasks from a software point of view. Automation software in CPPS is often divided into reusable software modules (standardized in libraries) and application- or customer-specific software. Previous studies [14, 17] show that the degree of reuse of software modules is often dependent on the architectural level [44]. Modules on lower levels are often characterized by a high degree of standardization and reusability since they often control standardized electro-mechanical components that occur multiple times in a machine or plant (e.g., sensors for measuring the filling level of tanks). On the other hand, modules on higher levels controlling the behavior of a machine part or even the whole machine tend to be more customized to specific boundary conditions (e.g., the mechanical plant layout or particular customer needs regarding implemented functionalities).

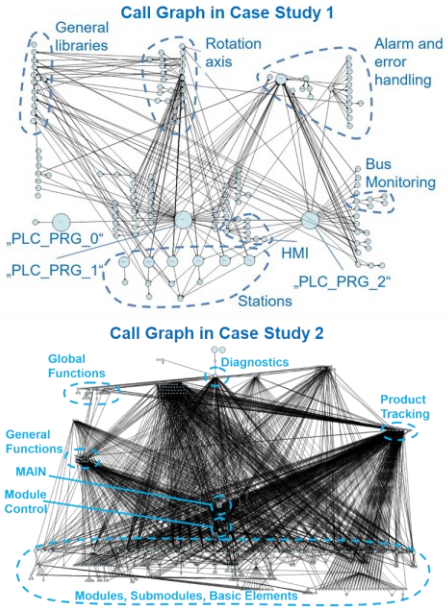

Fig. 3. Call graphs and functional distribution of companies with highest maturity results in a previous case study [14]

Companies follow different strategies to customize these application-specific modules to different boundary conditions. Customizable modules, e.g., can be adapted by using parameters in the source code to select and call the required software part and, thus, to create a customized variant. However, this may cause duplicated code in case the implementations of both variants are similar to each other. Another option are universal modules, which contain program sections for both variants executed if the respective parameter is set in the source code. The object-oriented extension of the IEC 61131-3-standard offers the possibility of transferring established reuse strategies that have already been in use in high-level language programming for decades to automation software in CPPS, e.g., reuse through inheritance. Different strategies and combinations can be found in industrial practice depending on the company-specific boundary conditions and the functionalities that the CPPS needs to provide (mainly determined by the type of technical process implemented). This heterogeneity of design principles and process-dependent influences makes it hard to develop a unique definition of software architecture that can be applied to all types of companies and their functional requirements. Vogel-Heuser et al. [14], e.g., compared the automation software maturity in different companies concluding that mature software can look significantly different depending on the company (cf. call graphs of case studies 1 and 2 in Fig. 3).

2) Non-functional Characteristics

Non-functional requirements (NFR) of automation systems play an essential role in designing software architecture. Common NFRs such as maintainability, scalability, portability, or reusability [45] are generally targeted in any software-intensive system. Automation systems, however, must meet specific NFRs derived from their embedded aspects as they control the external environment, sometimes in challenging conditions and considering humans in the surroundings.

Typically, CPPS must meet timing requirements dealing with the maximum cycle time to adapt to the dynamics of the controlled entities. On the other hand, they are also required to meet maximum response times (within a deadline) to external events related to alarms, which demand immediate action. Thus, to a certain extent, they must be predictable. Service availability is also essential, as their primary function is to control manufacturing assets. Fault tolerance mechanisms are usually implemented to assure availability by detecting and recovering from different types of faults. When operating in hazardous environments, they must also meet safety requirements. Safety integrity levels are used to classify safety functions corresponding to their

safety integrity. Compliance requirements related to standards, rules, and regulations are also important. The IEC 61131-3 offers mechanisms to meet some of these requirements, while open standardization efforts promoted by *PLCopen* – an independent, international association for industrial automation – try to cover others, such as motion control, safety, or data exchange.

New NFR appears to be met by automation systems. , as one of the components of the smart factory following the I4.0 concept. They are required to be interoperable and communicate to other components in the factory or inter-factories. Interoperability is met by integrating the assets they control, including legacy systems, mainly when those systems lack communication capabilities. They must offer communication capabilities using industry standards, e.g., OPC UA, to offer information services about their state and the state of assets they control to other components in the factory. Also, other types of services, such as manufacturing services, must be offered to achieve the flexibility needed to react not only to the ever-changing market demands but also to recover from manufacturing resources faults.

B. Automation Hardware and PLC Software Characteristics

The scope of CPPS ranges from individual machines up to entire production plants, which are controlled by reading sensor signals and controlling actuators, such as servo-drives to move workpieces [13]. This often leads to up to 100,000 input and output signals for a whole production plant [46]. Thus, besides software-internal characteristics, software architecture in CPPS is highly dependent on external interfaces and restrictions resulting from the controlled automation hardware. An automation project, usually running on a PLC, automates the behavior of the plant, but it also can contain control algorithms. The complete definition of an automation project's architecture requires:

1. Specification of the industrial control system hardware (centralized or distributed solution). The first implies that all information coming from and going to the plant are wired to the I/O modules of the PLC. The second requires industrial communication interfaces connected to I/O devices via Industrial Ethernet protocols (e.g., Profinet).
2. Software architecture of the PLC in terms of (global or local) variables and software modules. Despite providing guidance for defining the software, such as IEC 61131-3, there are several manufacturer-dependent software modules. According to PLCOpen, proposed standardization approaches (e.g., for creating library modules [47] or information exchange across projects [48]) are well received and are implemented by many platform providers.

The PLCOpen members Siemens, Beckhoff, Rockwell, B&R/ABB, Mitsubishi, or Schneider Electric, are the best-known control and automation system manufacturers, and each of them offers a proprietary tool normally non-compliant with others. Some of them follow IEC 61131-3 software modules and other proprietary standards such as S7. These standards do not specify an import or export format but the elements of the software model and the mechanisms to be offered to the user to define an application graphically. PLCopen is composed of different Technical Committees (TC), whereby TC6 proposes a Markup Language containing the elements of the IEC 61131-3 standard, their relationship, and other related issues, such as specific supplier information (*PLCOpen XML,* [48]).

As commented above, the automation project also contains the hardware architecture. Again, exporting and importing facilities offered by manufacturer vendors condition the content and file storage format. EPLAN Electric is an ECAD program used for configuring, documentation, and managing electro-technical automation projects in industrial environments. It offers the required facilities to create circuit diagrams, models, and cabinet configurations. EPLAN Electric defined an interface for sharing PLC data with PLC programming tool suites provided by PLC manufacturers. As far as the authors know, only two vendors allow importing models from EPLAN: Mitsubishi Electric and Siemens. The defined interface uses a vendor-neutral Markup Language notation, following AutomationML Standard format. Thus, PLCopen XML [49] and AutomationML [50] could be considered the common representation formats for software and hardware architecture of automation projects, respectively. Therefore, the challenge will be to define and perform the corresponding mapping rules that will be applied to the exported automation projects, which depend on manufacturing vendors' tool export and import capabilities.

C. Architectural Design Principles

As discussed in Section II, in computer science, extensive resources on software architecture, design patterns, and design principles are well proven and established. While object-oriented programming is standard in information technology, this programming method has not yet been widely used in automation software, e.g., due to heterogeneous qualifications of the programmers or influences on the runtime behavior. However, the object-oriented extension has been officially part of the IEC 61131-3 standards since 2013 and is supported by numerous platform providers. In the IEC 61131-3 area, there is thus a heterogeneous spectrum of programming styles and resulting architectures - from basic, low-level hierarchies programmed in graphical languages up to sophisticated, object-oriented module structures that can keep up with high-level language software. These differences in the utilization of available language resources are often far more significant in automation software than in high-level language software, making it difficult to derive a unified architecture definition.

Distributed control of CPPS is considered a prerequisite to increase efficiency and safety by providing redundancy of mechatronic components and their software [51]. However, up to now, CPPS are mainly controlled by central PLCs programmed according to IEC 61131-3, which supports distributed control only to a limited extent. IEC 61499 promises the convenient design of distributed control applications and brings features such as interoperability and portability to the domain of CPPS. Part 1 of the standard defines a general architecture in terms of several reference models (i.e., system model, device model, resource model, and application model). However, it does not provide guidelines on designing software parts and applications. In the past, IEC 61499 applications were mostly designed in a hierarchical way [52]. Recent research also shows promising results regarding distributed design patterns [53]. A main challenge for IEC 61499 is industry adoption [54, 55]. In

the past, it was only applied to small demonstrators and academic examples, therefore, the scalability was an open question. Meanwhile, IEC 61499 is in the phase of early adopters [11].

Besides the applied programming standard, further architectural principles can be observed in industrial practice leading to different types of software architecture. Common to all observed software structures is that the technical process to be implemented has a significant influence on which architectural principles are suitable. The core prerequisite for recommending suitable architectural principles is thus to learn from industry users and optimally support them in improving their software design. This can be done, e.g., within the scope of large-scale questionnaire studies in industry, which have already proven to be a valuable means of obtaining representative data on the current state of technical practice [14, 15, 46].

D. Company-specific Boundary Conditions

Previous analyses of industrial automation software, combined with in-depth interview studies, show that software architecture strongly depends on company-specific boundary conditions. These factors include, e.g., the locations the company is operating in, the size of software development teams, the industrial sector, or the amount and type of company-specific programming guidelines [56]. Due to these different factors, it is impossible to define a representative standard for an ideal software architecture for all different types of companies. Instead, it is required to define software architecture based on adaptable models and standards customized to an individual company's needs. One major factor that must be considered includes, e.g., which programming platforms a company works on. Depending on the platform vendor, certain restrictions and boundary conditions arise that influence the programming style and must therefore be considered when defining the architecture.

In contrast to computer science, where there is a uniform, clearly defined understanding of software architecture and quality, automation software in CPPS must be thought differently due to the high level of heterogeneity. As a prerequisite for evaluating the architecture and identifying optimization potential, an understanding must first be developed of what exactly is to be implemented and which functional and non-functional requirements must be met for the respective stakeholders in the company and on the customer side. These stakeholders include not only programmers in-house. Especially in plant manufacturing, e.g., the software is often adapted during commissioning by technicians with little software background, which poses a particular challenge to the architecture design.

E. Trends in the Context of Industry 4.0

In recent years, the rapid technological developments pose major challenges for companies to remain successful in global competition. Implementing I4.0 places entirely new demands on software architecture and communication between mechatronic components, which is why these influences are examined in more detail below.

PLC capabilities are continuously increasing according to technological advances in electronics and communications. For instance, modern units can perform bit operations in nanoseconds and are manufactured with an inbuilt Ethernet port, facilitating their integration into Ethernet-based networks. In addition, approaches for virtual commissioning are increasingly finding their way into industrial practice. However, the *programming model* motivated by IEC 61131-3 has not changed substantially across PLC realizations by different vendors. Therefore, there is the need for deterministic and distributed programming models that embrace explicit timing, event-triggered computation, and improved security [12]. Even if PLCs continue to be required to a considerable extent for the production of tomorrow, the introduction of the *service paradigm* may likely support these controllers in fulfilling the additional requirements resulting from the new production conditions. Therefore, different levels of services must be explored, spanning from the implementation of non-critical and overarching functionalities to the full development of all the control functions as services [57].

Within the I4.0 revolution, manufacturing companies are transforming to intelligent enterprises constituted by smart systems capable of connecting and communicating through Industrial Internet of Things (I2oT) technologies. However, *interoperability* remains a challenge due to a lack of standard approaches and vocabularies for the software of CPPS [58]. PLCs are starting to exploit I2oT technologies so that they will face an increasingly large set of threat vectors. For instance, classical approaches from computer science to prevent cyber-attacks cannot be applied to PLCs due to their extreme differences from main priorities to resource constraints. Therefore, innovative approaches and equipment must be developed [59].

IV. Defining Automation Software Architecture in Automated Production Systems

Using the workshop results and the derived categories, established software architecture definitions from computer science can be enlarged and adapted for automation software in CPPS as follows:

Automation software architecture in CPPS is composed by the basic structure of the software determined by **design decisions** forming the skeleton of the software. These design decisions primarily determine modules' size, interfaces, and interaction, i.e., groups of POUs jointly performing a specific functionality. Automation software architecture in CPPS is required to consider the **hardware topology**, including the number, distribution, and connection of PLCs. Thus, architecture is dependent on the interface to and the number and type of **hardware I/Os**, i.e., the sensors and actuators for controlling the mechatronic system's behavior.

This definition is intended to encourage software developers in practice to systematically design their architecture to fit the respective boundary conditions and technical processes in the company by taking all relevant factors into account.

V. Challenges and Research Potentials for Automation Software Architecture

The paper succeeded in formulating an architecture definition for automation software by combining the learnings from previous work and industrial case studies in a joint workshop. Nevertheless, there is still a long way to go for industrial users to capture, describe and optimize their software architecture sys-

tematically. Contrary to high-level language software development, automation software in CPPS has to deal with complex challenges. Many factors of these had already been identified in previous work [2, 13] and were reconfirmed in the conducted workshop. The most essential hurdles include:

1. Integration of extra-functional tasks (e.g., exception handling) into the module structure of the functional software
2. Lack of methodological support for the design of software architecture, e.g., design patterns and quality metrics
3. Complex hardware topologies with partially more than 100,000 hardware I/Os
4. High heterogeneity of stakeholders and company-specific boundary conditions leading to a lack of shared understanding of software quality
5. Discrepancy between technological requirements in the context of Industry 4.0 and historically grown legacy systems in practice

Therefore, in future work, it is crucial to develop methods and concepts based on the developed definition to better address the prevailing problems in practice and provide users with concrete means to improve the software. Questionnaire studies can be used, e.g., to obtain an even more comprehensive picture of current challenges in the machine and plant manufacturing sector, especially in an international comparison [46]. Furthermore, in-depth insights into industrial software architectures are required, which can be achieved by using targeted static code analysis to compare the strengths and weaknesses of design decisions regarding the overall quality. To make good solutions reusable, suitable design patterns that address the needs of industrial practitioners need to be formulated and evaluated in collaboration with experienced automation software developers.

VI. Conclusion and Outlook

While many established definitions of software architecture exist in computer science, and there is a generally accepted picture of software architecture, these definitions are not sufficient for the application to automation software. Therefore, this paper systematically examines different influencing factors that need to be considered for the definition of automation software architecture. An architecture definition is formulated that includes the above factors and is thus applicable to automation software in CPPS. A suitable architecture design is one of the fundamental prerequisites for many current technology innovations and is, therefore, an essential factor for companies to remain competitive in the long term.

However, there is still a long way to go to enable industrial practitioners to design and implement automation software architectures that fulfill the requirements of all stakeholders and fit the company-specific boundary conditions. Therefore, further questionnaire studies and industry comparisons are planned to receive feedback on current pain points and needs in industrial practice. On this basis, the developed definition is to be extended and specified in continuous exchange with industry partners to enhance the applicability to a broad field of use cases. The numerous influencing factors show that a pure analysis of software without considering the hardware and the different requirements cannot successfully improve automation software in a sustainable and long-lasting way. Instead, new analysis methods must be developed to provide appropriate support.

Acknowledgments

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology, and Development, and the Christian Doppler Research Association is gratefully acknowledged. We explicitly want to thank our industry partners for their continuous support.

References

[1] S. Biffl, A. Lüder, and D. Gerhard, Eds., *Multi-Disciplinary Engineering for Cyber-Physical Production Systems: Data Models and Software Solutions for Handling Complex Engineering Projects*. Cham: Springer, 2017.

[2] 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.

[3] J. Fischer, S. Bougouffa, A. Schlie, I. Schaefer, and B. Vogel-Heuser, "A Qualitative Study of Variability Management of Control Software for Industrial Automation Systems," *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 615–624, 2018.

[4] A. Gilchrist, "Introducing Industry 4.0," in *Industry 4.0*, A. Gilchrist, Ed., Berkeley, CA: Apress, 2016, pp. 195–215.

[5] W. Hasselbring, "Software Architecture: Past, Present, Future," in *The Essence of Software Engineering*, V. Gruhn and R. Striemer, Eds., Cham: Springer International Publishing, 2018, pp. 169–184.

[6] *IEC 61131-3: Programmable controllers - Part 3: Programming languages*, 3.0, IEC 61131-3, 2013.

[7] C. Sunder, A. Zoitll, J. H. Christensen, H. Steininger, and J. Rritsche, "Considering IEC 61131-3 and IEC 61499 in the context of component frameworks," in *2008 6th IEEE International Conference on Industrial Informatics*, Daejeon, Korea (South), 2008, pp. 277–282.

[8] Hafiyyan Sayyid Fadhlillah, Bianca Wiesmayr, Michael Oberlehner, Rick Rabiser, and Alois Zoitl, "Towards Delta-Oriented Variability Modeling for IEC 61499," in *Proceedings of the 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Västeras, Sweden, 2021.

[9] Lisa Sonnleithner, Sándor Bácsi, Michael Oberlehner, Elene Kutsia, and Alois Zoitl, "Do you smell it too? Towards IEC 61499 Bad Smells," in *Proceedings of the 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Västeras, Sweden, 2021.

[10] L. Sonnleithner, B. Wiesmayr, V. Ashiwal, and A. Zoitl, "IEC 61499 distributed design patterns," in *Proceedings of the 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Västeras, Sweden, 2021.

[11] G. Lyu and R. W. Brennan, "Towards IEC 61499-Based Distributed Intelligent Automation: A Literature Review," *IEEE Trans. Ind. Inf.*, vol. 17, no. 4, pp. 2295–2306, 2021.

[12] M. A. Sehr *et al.,* "Programmable Logic Controllers in the Context of Industry 4.0," *IEEE Trans. Ind. Inf.*, vol. 17, no. 5, pp. 3523–3533, 2021.

[13] B. Vogel-Heuser, E.-M. Neumann, and J. Fischer, "MICOSE4aPS: Industrially Applicable Maturity Metric to Improve Systematic Reuse of Control Software," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 1, pp. 1–24, 2022.

[14] B. Vogel-Heuser, J. Fischer, S. Feldmann, S. Ulewicz, and S. Rösch, "Modularity and architecture of PLC-based software for automated production Systems: An analysis in industrial companies," *JSS*, vol. 131, pp. 35–62, 2017.

[15] B. Vogel-Heuser and F. Ocker, "Maintainability and evolvability of control software in machine and plant manufacturing — An industrial survey," *CEP*, vol. 80, pp. 157–173, 2018.

[16] B. Vogel-Heuser, F. Ocker, and E.-M. Neumann, "Maturity variations of PLC-based control software within a company and among companies from the same industrial sector," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, St. Petersburg, 2018, pp. 283–290.

[17] 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 *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 362–371.

[18] B. Vogel-Heuser, S. Rösch, J. Fischer, T. Simon, S. Ulewicz, and J. Folmer, "Fault Handling in PLC-Based Industry 4.0 Automated Production Systems as a Basis for Restart and Self-Configuration and Its Evaluation," *JSEA*, vol. 09, no. 01, pp. 1–43, 2016.

[19] R. Reussner, M. Goedicke, W. Hasselbring, B. Vogel-Heuser, J. Keim, and L. Märtin, *Managed Software Evolution*. Cham: Springer International Publishing, 2019.

[20] R. Rabiser and A. Zoitl, "Towards Mastering Variability in Software-Intensive Cyber-Physical Production Systems," *Procedia Computer Science*, vol. 180, pp. 50–59, 2021.

[21] M. Vierhauser, R. Rabiser, and P. Grünbacher, "A case study on testing, commissioning, and operation of very-large-scale software systems," in *Companion Proceedings of the 36th International Conference on Software Engineering*, Hyderabad India, 2014, pp. 125–134.

[22] M. Galster *et al.,* "Variability and Complexity in Software Design," *SIGSOFT Softw. Eng. Notes*, vol. 41, no. 6, pp. 27–30, 2017.

[23] J. Bosch, R. Capilla, and R. Hilliard, "Trends in Systems and Software Variability [Guest editors' introduction]," *IEEE Softw.*, vol. 32, no. 3, pp. 44–51, 2015.

[24] B. Wiesmayr, A. Zoitl, and R. Rabiser, "Assessing the usefulness of a visual programming IDE for large-scale automation software," *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2021.

[25] K. Meixner, K. Feichtinger, R. Rabiser, and S. Biffl, "A reusable set of real-world product line case studies for comparing variability models in research and practice," in *Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume B*, Leicester United Kindom, 2021, pp. 105–112.

[26] E.-M. Neumann *et al.,* "Identifying Runtime Issues in Object-Oriented IEC 61131-3-Compliant Control Software using Metrics," in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, Singapore, Singapore, 2020, pp. 259–266.

[27] R. Jetley, A. Nair, P. Chandrasekaran, and A. Dubey, "Applying software engineering practices for development of industrial automation applications," in *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, Bochum, Germany, 2013, pp. 558–563.

[28] H. D. Tafur Muñoz, G. Barbieri, and C. E. Pereira, "An FMEA-based Methodology for the Development of Control Software Reliable to Hardware Failures," *IFACPapersOnline*, no. 1, pp. 420–425, 2021.

[29] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice,* 2nd ed. Boston, Mass.: Addison-Wesley, 2010.

[30] P. Kruchten, H. Obbink, and J. Stafford, "The Past, Present, and Future for Software Architecture," *IEEE Softw.*, vol. 23, no. 2, pp. 22–30, 2006.

[31] M. W. Maier, D. Emery, and R. Hilliard, "Software architecture: introducing IEEE Standard 1471," *Computer*, vol. 34, no. 4, pp. 107–109, 2001.

[32] E. Folmer, J. van Gurp, and J. Bosch, "Software Architecture Analysis of Usability," in *Lecture Notes in Computer Science, Engineering Human Computer Interaction and Interactive Systems*, D. Hutchison et al., Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 38–58.

[33] R. H. Reussner, H. W. Schmidt, and I. H. Poernomo, "Reliability prediction for component-based software architectures," *Journal of Systems and Software*, vol. 66, no. 3, pp. 241–252, 2003.

[34] J. Bosch, "Software Architecture: The Next Step," in *Lecture Notes in Computer Science, Software Architecture*, T. Kanade et al., Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 194–199.

[35] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: Abstraction and Reuse of Object-Oriented Design," in *Lecture Notes in Computer Science, ECOOP' 93 — Object-Oriented Programming*, G. Goos, J. Hartmanis, and O. M. Nierstrasz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 406–431.

[36] D. Hutchison *et al., Eds., Engineering Human Computer Interaction and Interactive Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[37] L. C. Briand, S. Morasca, and V. R. Basili, "Measuring and assessing maintainability at the end of high level design," in *1993 Conference on Software Maintenance*, Montreal, Que., Canada, 1993, 88-87.

[38] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017.

[39] M. Harris and R. Z. Khan, "A Systematic Review on Cloud Computing," *International Journal of Computer Sciences and Engineering (JCSE)*, Vol. 6, Issue 11, pp. 632–639, 2018.

[40] E. Wolff, *Microservices: Flexible software architecture*. Boston: Addison-Wesley, 2017. [Online]. Available: http://proquest.tech.safaribooksonline.de/9780134650449

[41] H. Muccini and M. T. Moghaddam, "IoT Architectural Styles," in *Lecture Notes in Computer Science, Software Architecture*, C. E. Cuesta, D. Garlan, and J. Pérez, Eds., Cham: Springer International Publishing, 2018, pp. 68–85.

[42] T. Terzimehic *et al.,* "Towards an industry 4.0 compliant control software architecture using IEC 61499 & OPC UA," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Limassol, 92017, pp. 1–4.

[43] 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)*, Porto, 72018, pp. 725–732.

[44] *Batch control - Part 1: Models and terminology*, IEC 61512-1:1997, International Electrotechnical Commission (IEC), 2000.

[45] *ISO/IEC 25010 System and software quality models,* ISO/IEC, 2010.

[46] B. Vogel-Heuser, Eva-Maria Neumann, A. Zoitl, D. A. Gutierrez, R. Rabiser, and H. S. Fadhlillah, "An International Case Study on Control Software Development in Large-Scale Plant Manufacturing Companies of One Industrial Sector at Different Locations," in *47th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, IEEE, Ed., 2021.

[47] PLCOpen, "PLCopen Software Creation Guidelines:: Creating PLCopen Compliant Libraries," 2017.

[48] PLCOpen, "XML Formats for IEC 61131-3," 2009.

[49] M. Marcos, E. Estevez, F. Perez, and E. Der Wal, "XML exchange of control programs," *IEEE Ind. Electron. Mag. (IEEE Industrial Electronics Magazine)*, vol. 3, no. 4, pp. 32–35, 2009.

[50] R. Drath, *AutomationML: A Practical Guide*: De Gruyter, 2021.

[51] W. Dai and V. Vyatkin, "Redesign distributed IEC 61131-3 PLC system in IEC 61499 function blocks," in *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)*, Bilbao, 2010, pp. 1–8.

[52] A. Zoitl and H. Prähofer, "Guidelines and Patterns for Building Hierarchical Automation Solutions in the IEC 61499 Modeling Language," *IEEE Trans. Ind. Inf.*, vol. 9, no. 4, pp. 2387–2396, 2013.

[53] B. Wiesmayr, L. Sonnleithner, and A. Zoitl, "Structuring Distributed Control Applications for Adaptability," in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, Tampere, Finland, 2020, pp. 236–241.

[54] K. H. Hall, R. J. Staron, and A. Zoitl, "Challenges to Industry Adoption of the IEC 61499 Standard on Event-based Function Blocks," in *2007 5th IEEE International Conference on Industrial Informatics*, Vienna, Austria, 2007, pp. 823–828.

[55] A. Zoitl, T. Strasser, K. Hall, R. Staron, C. Sünder, and B. Favre-Bulle, "The Past, Present, and Future of IEC 61499," in *Lecture Notes in Computer Science, Holonic and Multi-Agent Systems for Manufacturing*, V. Mařík, V. Vyatkin, and A. W. Colombo, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 1–14.

[56] E.-M. Neumann, B. Vogel-Heuser, J. Fischer, F. Ocker, S. Diehm, and M. Schwarz, "Formalization of Design Patterns and Their Automatic Identification in PLC Software for Architecture Assessment," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 7819–7826, 2020.

[57] Langmann and Stiller, "The PLC as a Smart Service in Industry 4.0 Production Systems," *Applied Sciences*, vol. 9, no. 18, p. 3815, 2019.

[58] G. Barbieri and D. A. Gutierrez, "A GEMMA-GRAFCET Methodology to enable Digital Twin based on Real-Time Coupling," *Procedia Computer Science*, vol. 180, pp. 13–23, 2021.

[59] G. Corbò, C. Foglietta, C. Palazzo, and S. Panzieri, "Smart Behavioural Filter for Industrial Internet of Things," *Mobile Netw Appl*, vol. 23, no. 4, pp. 809–816, 2018.