Software Engineering in Industrial Automation: State-of-the-Art Review

Valeriy Vyatkin, *Senior Member, IEEE*

***Abstract*—This paper presents one perspective on recent developments related to software engineering in the industrial automation sector that spans from manufacturing factory automation to process control systems and energy automation systems. The survey's methodology is based on the classic SWEBOK reference document that comprehensively defines the taxonomy of software engineering domain. This is mixed with classic automation artefacts, such as the set of the most influential international standards and dominating industrial practices. The survey focuses mainly on research publications which are believed to be representative of advanced industrial practices as well.**

***Index Terms*—Industrial automation, software engineering.**

I. Introduction

THERE are numerous evidences on the growing complexity and importance of software in industrial automation systems. For example, the German Engineering Federation VDMA has presented the growing ratio of software development in the costs of machinery [1]. The ratio of software has doubled in one decade from 20% to 40%. If this trend continues, the main activity of automation systems suppliers and developers will be software engineering.

Software engineering is an established industry with its methods, traditions, and curriculum. Most of such methods and tools are applicable across different domains, from information processing applications, such as databases, trade, and Web-commerce, to real-time control of energy and transportation systems and manufacturing automation. The IEEE Computer Society defines software engineering in [2] as follows:

1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, that is, the application of engineering to software.
2) The study of approaches as in 1).

Reflecting on the trend of growing importance of software in automation, there is a great number of research projects and corresponding publications addressing various aspects of software development process in industrial automation domain. The main driving forces of these developments are lifecycle costs, dependability, and performance. The cost issues are addressed through the entire lifecycle of software, from requirements capturing to phasing the software out. The dependability-related challenges focus on the methods and activities ensuring functional safety of computer systems through guaranteeing certain safety properties of the software. The performance—related developments aim at ensuring sufficient execution speed and lower memory footprint of the code. These characteristics can be interdependent, for example, certain dependability guarantees may depend on sufficient performance, and higher performance of software (achieved on account of more efficient coding or compilation) can reduce the overall costs of automation system by using lower spec hardware. All of these characteristics of computer hardware and software certainly impact on the performance and reliability of systems being automated.

The goal of this paper is to review the software engineering approaches used in the automation domain and put the automation research into the context of mainstream software engineering. In the author's opinion, this will help the researchers to transfer the knowledge across the different domains of computer systems and identify appropriate development methods for particular tasks.

The remainder of this paper is structured as follows. Section II introduces basic terminology of the area. Section III presents the scope of the survey aligned with the areas of software engineering. Section IV gives a brief overview of the most important standards related to software development in automation. Section V is dedicated to requirements engineering. Section VI reviews developments related to software design, in particular on structures and architectures. Section VII specifically focuses on several design strategies and methods, and on software construction issues, such as programming languages. Section VIII discusses most notable developments related to software testing, maintenance, and evolution, as well as to software configuration management and software engineering management, software process, and software quality. Section IX provides discussion on current and future challenges. Section X concludes the paper.

II. Terminology

The ARC advisory group, a respected consultancy and trends observer in industrial automation, distinguishes several classes of automation products [3] as shown in Fig. 1 (left-hand side). Most of these products are software-intensive in the sense that their functionality is largely determined by the software executed on embedded computers. The automation software is classically (e.g., in [4]) divided into three categories presented in the

Manuscript received May 13, 2012; revised August 22, 2012; accepted April 01, 2013. Date of publication April 15, 2013; date of current version August 16, 2013. Paper no. TII-12-0355.

The author is with the Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Luleå 97187, Sweden, and also with the Department of Electrical and Computer Engineering, The University of Auckland, Auckland 1142, New Zealand (e-mail: valeriy.vyatkin@ltu.se; v.vyatkin@auckland.ac.nz).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2013.2258165

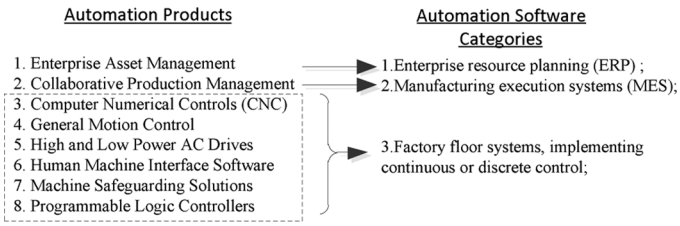

Fig. 1. Relation of automation products to software categories.

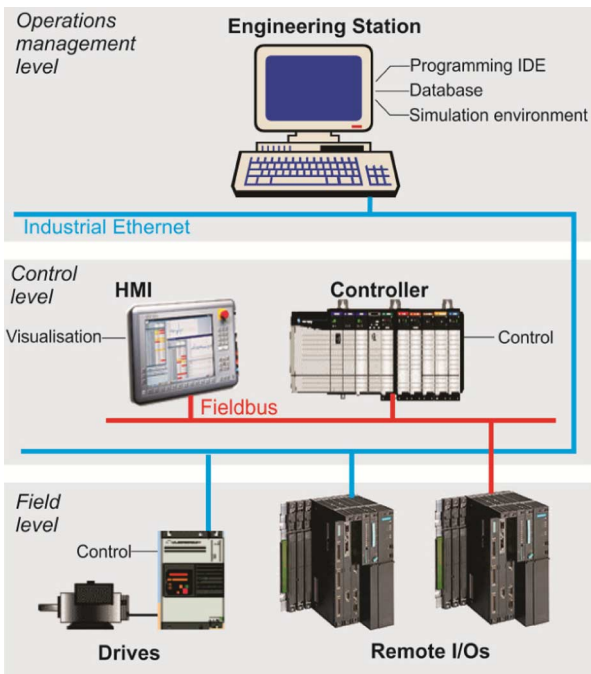

Fig. 2. Typical hardware architecture of factory floor automation systems and software allocations.

right-hand side of the figure. This review is focusing on the developments related to the factory floor systems and manufacturing execution systems. However, numeric control systems (CNCs), which also belong to the factory floor level, are excluded from the survey due to their very specific nature. The reader is referred to [5], [6] for more information on this subject.

One can get an idea of the factory floor automation software typical functionalities from the traditional centralized automation system structure presented in Fig. 2. As one can see, there are four types of data-processing devices communicating via networks: engineering station with simulation, database and programming software, HMI device (e.g., touchscreen computer) with visualization and human interface software, programmable logic controller (PLC) with control software, and one or several motor drives controlling speed of various moving parts.

The PLCs are often considered as the main workhorses of industrial automation systems. They are connected with the plant peripherals, such as sensors and actuators, via direct electric wiring to their input/output (I/O) ports, or via remote I/O modules which communicate with the PLC via field area networks (fieldbuses).

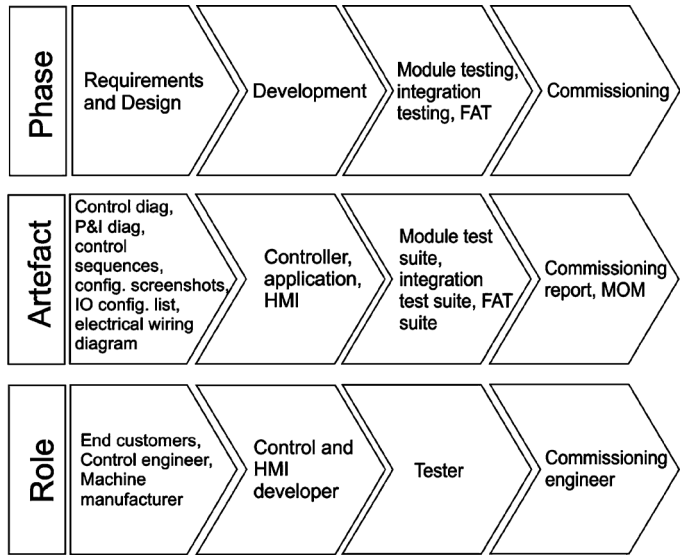

Fig. 3. Automation application development life cycle according to [8].

III. Areas of Software Engineering

This survey is structured according to the key areas of software engineering as suggested in the "Guide to the Software Engineering Body of Knowledge" [7] as follows: requirements, software design and construction, testing, maintenance, configuration management, software engineering management, software engineering process, tools and methods, and software quality.

Another classification line can be conditionally drawn between the so-called *traditional* versus *emergent* software engineering methodologies. Examples of the relatively traditional ones include component-based design architectures, object-oriented and aspect-oriented design methods, and agile development. These methods aim at more efficient construction of software. The emergent ones include the technologies aiming at automatic composition of software functionalities (online or offline) achieved on account of multi-agent collaboration, the use of web-services, or automatic reasoning. These technologies provide for flexibility and plug-and-play composition of software which is enabled by such mechanisms as multi-agent collaboration and service discoveries.

One of the few available surveys on the topic is done by Dubey [8] and provides evaluation of software engineering methods in the context of automation applications. This work presents the phases of the automation application development life cycle (AADLC) as shown in Fig. 3. The paper describes the AADLC with four major phases: 1) requirements and design; 2) development; 3) testing; and 4) deployment and commissioning. Fig. 3 shows phases, artifacts (involved or generated) in each phase, and roles responsible to drive each phase. There is also a long maintenance phase after the commissioning phase (which is not shown in the figure).

Other focused sources of reference are the special sessions on software engineering in automation that have been coorganized by the author at international conferences INDIN 2010 and 2011 and INCOM 2009, as well as a variety of IEEE and IFAC publications.

IV. Role of Standards and Norms

Standards and norms are very influential in the development of industrial automation software. Here, we briefly review the most relevant ones.

A. IEC 61131-3

The IEC 61131-3 standard [9] for PLC programming languages has been one of the most successful global standards for industrial control software. The standard has captured the most common ways of thinking of control engineers. The standard contributes to portability of automation software by suggesting four programming languages and sequential function charts (SFC). The standard helps to migrate software developed for one PLC type to PLCs of other vendors.

The standard is constantly developing to reflect the latest trends in the development practice. Its third edition includes extensions towards object-oriented programming, as preliminary discussed by Werner in [10].

The critique of IEC 61131-3 mentions its semantic ambiguities [11] and difficult integration into distributed, flexible automation systems [12]. The centralized programmable control model of PLCs may cause unpredictable results and overheads when two or more controllers communicate via network, and its centralized, cyclically scanned program execution model limits the reuse of the components in case of reconfiguration.

B. ISA 88/95

According to [13], the ISA-88 standard is intended to provide solutions for *application configuration* and *system integration* problems. It is designed to be used with batch process control systems, but it can also be applied to discrete, continuous, hybrid, and storage process control. It is internationally standardized as IEC 61512 [14].

The ISA-95 standard for enterprise-control system integration (standardized internationally as IEC 62264 [4]) defines the interface between control functions and other enterprise functions, effectively between the ERP and MES levels.

C. IEC 61499

The IEC 61499 reference architecture [15] has been conceived to facilitate the development of distributed automation systems with decentralized logic. The standard presents a reference architecture that exploits the familiarity among control engineers accustomed to a block-diagram way of thinking. The main design artefact, function block (FB), has been extended from the subroutine-like structure in IEC 61131-3, to the process-like abstraction used in the theory of distributed computing systems where it represents an independent computational activity with its own set of variables (context) and communication with other processes via messages. The event interface of FB is well suited to modelling of inter-process message-based communication. For a state-of-the art survey on IEC 61499, the reader is referred to [16]. The standard directly addresses the trend of the increasing importance of software in automation systems design by improving *portability*, *configurability*, and *interoperability* of automation systems. This standard is in early stages of its adoption, with three commercial Integrated Development Environments (IDE) released on the market. This standard has been criticized for a number of weaknesses, such as semantic ambiguities, high learning threshold, and insufficient harmonization with existing PLC technologies. However, most of these issues have been addressed in the second edition released in 2011.

D. IEC 61804

The IEC 61804 standard draft (technical report) [17] describes the specification and requirements of distributed process control systems based on function blocks. As noted by Diedrich *et al.* in [17], a part of the proposed standard is Electronic Device Description Language (EDDL)—a language that describes the properties of automation system components, such as vendor information, version of firmware/hardware, and data format. Through this language, all of the information will be carried between the devices (e.g., controllers, sensors, actuators, and engineering stations) by a fieldbus. This language fills in the gap between the FB specification and product implementation by allowing manufacturers to use the same description method for devices of different technologies and platforms.

E. IEC 61850

The IEC 61850 standard—Communication Networks and Systems in Substation [18]—addresses the interfacing issues and standardizes communication to avoid the use of vendor-specific protocols in power systems automation. Along with communication, it suggests an object-oriented model for this domain. According to [19], IEC 61850 decomposes power substation, including functions for monitoring, control, and protection and primary devices, down to objects, thus obtaining object-oriented representation of the power system. The smallest object is a "data attribute" which is encapsulated into a "common data" object. These are the data used by devices and functions when they operate. The data and data attribute are the information models for the automation functions and primary devices, which are "wrapped" into a set and represented as a logical node (LN). LNs can be described by the data that they consume and produce. A logical device can be populated by a number of LNs to perform a particular function. In turn, the logical device can perform more than one function.

Higgins *et al.* in [20] pointed out a deficiency of this standard related to encapsulation of programmable logic into LNs and suggested using FBs for this purpose. This approach has been successfully applied by Strasser [21] towards closed-loop simulation of power automation systems with distributed intelligence. It was used by Zhabelova in [19] to propose a multi-agent architecture for power distribution automation.

V. Requirements Engineering

The domain of software requirements includes the fundamentals such as product vs. process, functional vs. non-functional, as well as emergent properties of the software. As such, it addresses r*equirements process, elicitation, analysis, specification, and validation*. Proper requirements management is becoming also important as a part of dependability certification.

Functional requirements to automation software are often extracted from product specifications or recipes. However,

nowadays non-functional requirements play increasingly important role. One of the major current trends in nonfunctional requirements of automation systems concerns with their flexibility which is needed to react on the ever-changing market demands. According to [22], there are two major approaches to achieve flexibility, known as reconfigurable manufacturing systems (RMSs) and flexible manufacturing systems (FMSs). In RMS machine components, machines, cells, or material handling units can be added, removed, modified, or interchanged as needed to respond quickly to changing requirements, while FMS change its position or state to accommodate for changes in manufacturing requirements. The key characteristics of RMS are defined to be modularity, integrability, flexibility, scalability, convertibility, and diagnosability. The goal of software developers is to minimize the effort in redesigning RMS software at every reconfiguration attempt.

An approach proposed by Ryssel *et al.* [23] targets automatic reconfiguration of the existing software components from a library to achieve an automation application that satisfies the user's requirements. The proposed technique takes into account not only compatibility of components at the syntactical level but also at the semantic level, e.g., to achieve interoperability of the assembled application. The Semantic Web technologies [24] were used to formally specify the requirements and allow the generator to interpret their meanings.

One more very important nonfunctional requirement is time. As pointed out in [25], specifically in distributed systems, timing characteristics strongly depend on application relations and chosen communication means. The authors describe a method for supporting engineers in their choices to meet nonfunctional requirements when designing distributed control systems and report on the ongoing development work towards the corresponding tool.

Ljungkrantz *et al.* [26] propose reusable automation components containing not only the implementation details but also a formal specification defining the correct usage and behavior of the component. This intends to shorten modification times as well as reduce number of programming errors. This formal specification uses temporal logic to describe time-related properties and has a special structure developed to meet industrial control needs.

Runde *et al.* [27] address the requirement engineering issues in the domain of building automation systems (BASs) by proposing a novel knowledge-based process also supported by means of Semantic Web technologies. The paper states that today's approach to elicitation of requirements for BAS depends on customers' requests and on the planner knowledge. The paper describes a novel "intelligent" software tool that supports the planner at requirements elicitation.

Hirsch [28] addresses the problem of requirements engineering in automation by employing the popular SysML technology. The author presents an extended case study of a modular automated manufacturing system design and shows the pathway of refining the requirements into a fully working automation code that is compliant with the IEC 61499 standard. The work describes the entire process of specifying an automation system including textual descriptions of the requirements, graphical descriptions of the structure and its behavior. A subset of SysML design structures has been identified which is sufficient for supporting the proposed design process. It is outlined in the work that the use of SysML can substantially ease some of automation problem solutions and help to embed model-based design methodologies in a broader context of embedded control systems.

VI. Software Design

The software design domain encompasses such areas as: structure and architecture, design strategies and methods, design quality analysis, along with evaluation and software design notations.

The relevant publications in the industrial automation domain often deal with a combination of these areas, for example, proposing a certain custom architecture, accompanied with the corresponding design strategies or methods. Therefore, here, several architectures will be discussed along with related quality analysis and notations. Section VII will exemplify several design strategies and methods.

A. Model-Driven Engineering (MDE)

MDE is a software development methodology which exploits domain models rather than pure computing or algorithmic concepts. MDE has been promoted as a solution to handle the complexity of software development by raising the abstraction level and automating labor-intensive and error-prone tasks.

The Object Management Group (OMG) is an influential standardization organization in business information technology that defined a number of standards for MDE, in particular model-driven architecture (MDA), based on MetaObject Facility (MOF) and Unified Modelling Language (UML). UML provides a number of diagram types for requirements capturing and refinement to executable code. SysML is a UML derivative for engineering applications that is getting increasingly popular. In particular, SysML supports such design phases as requirements capturing and formalization of specifications.

Bonfé and Fantuzzi [29] have introduced the use of UML in automation providing methodology of PLC code design by refining UML specification. Thramboulidis proposed in [30] generation of IEC 61499 function blocks from UML diagrams. Dubinin *et al.* described the UML-FB architecture with both ways of generation of UML diagrams from function block designs and vice versa in [31]. In [32] Thramboulidis proposed the IEC 61499-based concept of model-integrated mechatronic architecture for automation systems design. The concept of intelligent mechatronic components (IMCs) [33] stems from the ideas of object-oriented automation systems design [12] further enriched with embedded simulation and other functionalities. The IMC further develops the concept of automation object (AO), following [34] and [35].

A UML automation profile [36] was introduced by Ritala and Kuikka by extending UML v2 with various automation domain-specific concepts for modelling automation applications in an efficient way. Another similar development by Katzke and Vogel-Heuser [37] delivered the UML for Process Automation (UML-PA) modelling language. It is claimed that UML-PA allowed developers to identify plant relationships and apply these to UML-PA elements much quicker compared to the traditional

UML. Further tests compare the application of UML and Idiomatic Control Language (IDL) which is another type of description language for modelling industrial systems.

However, as noted by Mohagheghi and Gehen [38], very few efforts have been made at collecting evidence to evaluate MDE benefits and limitations. From that perspective, such works in the automation domain are especially valuable. Thus, a test reported in [37] was done using some test subjects and showed that subjects applying UML-PA recognized plant characteristics and their application almost 50% faster than those attempting to use traditional UML. Vogel-Heuser *et al.* [39] present an interesting field experiment on teaching of 85 trainees to two different approaches to PLC programming, UML and the FB Diagrams of IEC 61131-3. This paper focuses on the educational aspects of the training using both approaches, discussing the correlations found between the modelling and/or programming performance and cognitive abilities, interest, workload, expertise, and school grades of the trainees. However, the paper does not provide comprehensive analysis of the design efficiency for these technologies.

Witsch *et al.* have proposed in [40] a tool-supported approach for a bidirectional mapping of hybrid function blocks defined in the object-oriented extension of IEC 61131-3 to UML class diagrams. While most of MDE applications are related to control at the shop-floor level, Witsch and Vogel-Heuser [41] apply UML to higher level of the automation pyramid: manufacturing execution system (MES) design, and report on the qualitative and quantitative benefits observed. They state the MDE as an important formalization step, first introducing formal syntax and then formal semantics of the MES to be designed.

Thramboulidis and Frey investigate MDE in the IEC 61131-3 context in [42] aiming at increase of productivity and reliability of the development process of industrial automation systems. Piping and instrumentation diagrams (P&ID) are considered a source of requirements for process control engineering (PCE). The paper proposes the corresponding SysML profile for PLC programming to allow the developer to work in higher layers of abstraction than the one supported by IEC 61131-3 and effectively move from requirement specifications into the implementation model of the system. This work is remarkable for using a domain-specific model of P&ID rather than models provided by UML/SysML for representing requirements. In particular, the authors refer to the IEC 62424 standard [43] that defines how PCE requests should be represented in a P&ID to facilitate the automatic transfer of data between P&ID and PCE tool and to avoid misinterpretation of graphical P&ID symbols for PCE.

An important foundation for holistic system engineering and automatic program generation is availability of unified data formats for the artifacts used in the engineering process. AutomationML [44] provides such a generic data exchange mechanism to support the interoperability among various manufacturing engineering tools. The foundation of AutomationML is the *Computer-Aided Engineering eXchange* (CAEX) file format defined in the IEC 62424 standard. CAEX establishes a flexible object-oriented meta-model for data storage, where static object information can be structured and correlated. AutomationML further specifies concrete usages of the CAEX concepts for manufacturing plant engineering. The plant topology is defined using basic CAEX constructs. The geometry, kinematics, and control logic of each plant object are stored in their specific file formats. For example, in the current version of AutomationML, control logic is specified in PLCopen XML [45]. These data are then integrated into AutomationML using the linking mechanism provided by CAEX. As a result, all of the required domain-specific views of the plant are unified in the AutomationML model.

A MDA-based design approach aiming at automatic generation of automation programs [46] has been explored by Estevez and Marcos who proposed an XML-based *Industrial Control System Description Language* in [47] to consolidate the modelling methodologies used to support the development phases. In their approach, three domain-specific views have been identified to describe an industrial control system, including control hardware, electric circuitry, and software. Software components are described in a form that is suitable for generation of IEC 61131-3 PLC-based applications. Hardware components consist of hardware from a variety of vendors. The mapping between software components is done using an XML style sheet (XSL) which does the transformation between domains and can help in portability of code between different PLC vendors.

The specific of industrial automation dictates special requirements to model driven development. UML and SysML models would have limited applicability during systems commissioning at the shop floor if round-trip engineering is not supported by tools (i.e., once PLC code is generated from UML models, it may undergo "manual" modifications, after which the models cannot be reconstructed). This is why these ideas are still rarely used in practice. 3S Software presented an attempt to address this issue in Professional Developer Edition of CoDeSys. This tool supports three types of UML diagrams, two of which, State Charts and Activity Diagrams, were made fully executable up to the level of PLC programming language.

However, design processes in different industry branches, such as automotive, chemical or energy, differ substantially, and the potential of MDE acceptance, in the author's opinion, can also be different in these sectors. Therefore, finding a proper set of models which would represent equilibrium between UML diagrams and PLC languages still remains an open problem.

B. Component-Based

Component-based software engineering (CBSE) in a broader sense is a reuse-based approach to defining, implementing, and composing loosely coupled independent components into systems. An individual software component is a software unit that can be reused without any modifications in different software systems similar to such hardware components as integrated circuits.

Two major aspects of component-based development for automation are being discussed in the literature and are often confused: implementation-level issues and application-level issues.

On the implementation side, the main goal is masking particular details of component's location, which can be a remote server, connected by a variety of networks and controlled by an operating system different from that of the client. Middleware is commonly used as an implementation mechanism supporting this transparency, examples of which include CORBA [48] and

DCOM [49]—common component implementation models used in the general ICT. There are other examples, such as Object Management Facility (OMF) reported in [50], and Data Distribution Service that provides time-related quality of service (QoS), investigated in [51] in the IEC 61499 context.

There are examples of industrial development toward integration of PLCs into systems communicating via networks by proposing integration component architectures, such as Modbus-IDA [52] and PROFInet-CBA [53] based on the DCOM architecture.

The main implementation challenge is achieving execution properties such as determinism and real-time guarantees when using middleware.

Another aspect of component-based design is related to modularity planning at the application level. Designing, developing, and maintaining components for reuse is a very complex process which places high requirements not only for the component functionality and flexibility, but also for the development organization.

Maga *et al.* [54] attempted to define appropriate levels of granularity for reusable models. Depending on modelled aspects and on the purpose of the model, a fine-, medium-, or coarse-grained model can be more appropriate. The decision of which granularity is appropriate for a reusable model depends on both the considered domain and the intended reuse of the concerned model. As a general recommendation, the authors suggest to use well-documented, hierarchical, nested models, which provide different levels of granularity depending on the required functionality.

Szer-Ming *et al.* [55] describe the implementation of a distributed network-based industrial control system for an assembly automation system. The authors stress limitations of the common component models, such as lack of real-time guarantees and high complexity. They propose own component model for distributed automation, where the component encapsulates a state machine. The control system is composed of autonomous, intelligent components which have the capability of participating in the automation control without the need for a master controller. It is envisaged by the authors that assembly automation system that adopt this control approach can have greater agility, reusability and reduction in development cost.

The development of component-based automation architecture of the IEC 61499 standard has stipulated the large number of research works. For example, Strasser *et al.* [56] study the use of graphical programming languages in the automation domain to handle increasingly complex applications with the goal to reduce engineering costs for their establishment and maintenance. They use the concept of subapplications to provide a cost-effective solution for component-based industrial automation applications. The paper introduces a simple but effective engineering approach to structure large-scale distributed control programs based on a hierarchical plant structure model with IEC 61499 subapplications. Proponents of the IEC 61499 approach see its model as a proper equilibrium balancing MDE and component-based features with the legacy of PLC programming. However, the critics point out the lack of implementation support, similar to that of component-based middleware.

The evolution of requirements for products generates new requirements for components, such as planning the component life cycle where the component first reaches its stability and later degenerates in an asset that is difficult to use, to adapt, and to maintain.

Crnkovic and Larsson discuss in [50] both implementation and application aspects of component-based design referring to industrial experience. Different levels of component reuse have been identified along with certain aspects of component development in the context of industrial process control, including questions related to use of standard and *de facto* standard components. As an illustration of reuse issues, a successful implementation of a component-based industrial process control system is presented. The authors conclude that the growing adoption of component architectures raises new questions such as: component generality and efficiency, compatibility problems, the demands on development environment, and maintenance. For example, as far as compatibility concerns, the authors outline several levels, such as compatibility between different versions of components, between their implementations for different platforms, one aspect of which is compatibility of graphical user interfaces. Despite the existence of several popular component implementation models, often they are not equally supported on all required hardware–software platforms, which raises the problem of components' migration.

C. Design Based on Formal Models

Formal models of software systems create a firm foundation of software engineering, and this is especially true for industrial automation. The purpose of such design is to create dependable software with properties that can be guaranteed by design. The specifics of automation systems is that in most cases they are control systems where the dynamics of the plant matters and must be taken into account when control software is designed. Moreover, it can be used as an advantage helping in automatic software design.

In [57], Hanisch introduced a comprehensive framework for applying formal methods in automation based on the closed-loop plant—controller interaction. That work reviewed some of the basic design patterns of control engineering and shows the differences and similarities of the control engineering approach and methodologies taken from computer science and technology. In that work, the author proclaimed the goal of synthesizing formally controllers of industrial systems with subsequent code generation from the formal model.

The PLC code generation from a formal model is a relatively easy part of this proposition as compared to the formal model synthesis, and has been explored by many researchers. For example, Cutts and Rattigan in [58] used Petri nets, Frey [59] used signal-interpreted Petri nets, an example of PLC code generation from a modular Petri-net dialect has been demonstrated by Thieme and Hanisch in [60].

The progress in formal synthesis of control algorithms based on specifications and constraints has been reviewed in [61]. The general difficulty of formal synthesis is high computational complexity of the process. Other hurdles include the need of

formal specifications and their possible conflicts. Winkler *et al.* [62] present a new methodology to enhance the synthesis process by application of new structural analysis methods. The used formal modelling language is safe Net Condition/Event Systems (sNCES). A new kind of model structure based on graphical meta-description of the model behavior is introduced. This graphical representation allows system behavior analysis without exploring the whole state space that reduces performance requirements and extends applicability of the method.

However, the practical use of formal models in industrial practice is not yet common. Knowledge of formal methods is not common among automation and control engineers, while their application is associated with high computational complexity.

D. Multi-Agent Architecture

According to Woodbridge [63], multi-agent system (MAS) approach constitutes a novel software-engineering paradigm that offers a new alternative to design decision-making systems based on the decentralization of functions over a set of distributed entities. Research on application of multi-agent systems in industrial automation presents a rich body of results and has been subject of several surveys, e.g., [64]–[66]. As pointed out in [67], distributed agents are increasingly adopted in automation control systems, where they are used for monitoring, data collection, fault diagnosis, and control. Distributed multi-agent systems are an appropriate concept to meet the emerging requirements to software development: they enable a modular, scalable, and flexible software design and allow for distributed data collection and local preprocessing. They also provide on-site reactivity and intelligence for remote control scenarios, where the network channel is not capable of transporting each and every control command. Finally, they offer an abstraction level, when accessing proprietary devices for monitoring and control.

However, Theiss *et al.* [67] stress that existing agent platforms do not always fulfil the requirements of practical automation applications in respect of real-time properties and resource usage, leading to significant overhead in respect of design effort and runtime resources. To meet the specific requirements of the automation domain, a resource-efficient agent platform AMES was developed by the authors, which relies on established concepts of agent platforms, but modifies and supplements them accordingly. This platform is implemented in Java and in several C++ variants.

Herrera *et al.* [68] present the integration of the intelligent agent concept along with the service-oriented architecture (SOA) concept for industrial control software. A particular focus is on reconfigurable manufacturing systems and their ability to add/remove components on the fly. It is said that there is a strong synergy between MAS and SOA due to their goal of providing a platform neutral software component implementations. A standard protocol for MAS communication is developed by the Foundation for Intelligent Physical Agents (FIPA) and is said to be the current *de facto* standard. A language provided by the FIPA group is the Agent Communication Language (ACL) and is essentially a protocol that agent-based solutions should implement if they want to advertise that they are FIPA compliant. JADE was selected as a framework for MAS development. OWL-S which is an upper ontology for service descriptions is used in that work to provide semantic description of services. It defines their inputs, outputs and preconditions. Next, using ACL, it is described how some MAS components and features can be mapped to their corresponding abstractions in the SOA domain.

E. SOA

Semantic Web services are seen by many researchers as a way to overcome challenge of rapid reconfigurability in order to evolve and adapt manufacturing systems to mass customization. This includes the use of ontologies that enables performing logical reasoning to infer sufficient knowledge on the classification of processes that machines offer and on how to execute and compose those processes to carry out manufacturing orchestration autonomously. Lastra and Delamer in [69] provide a series of motivating utilization scenarios and present the corresponding research roadmap.

According to Erl [70], the SOA establishes an architectural model that aims at enhancement of the efficiency, agility, and productivity of an enterprise by positioning services as the primary means through which solution logic is represented in support of the realization of strategic goals associated with service-oriented computing. SOAs are getting increasingly important in general purpose computing and influence the corresponding developments in the automation world.

Jammes and Smit [71] outline opportunities and challenges in the development of next-generation embedded devices, applications, and services, resulting from their increasing intelligence. The work plots future directions for intelligent device networking based on service-oriented high-level protocols and outlines the approach adopted by the SIRENA research project.

The work by Cannata *et al.* [72] presents results of the follow-up SOCRADES project. Particularly the link between manufacturing systems and their corresponding business management components is focused upon. The authors propose the use of SOA at the device level to complement its capabilities at the enterprise level. This proposal adds vertical integration level required for device level all the way up to enterprise systems. Some features of SOA outlined in this paper are as follows:

- loose coupling, since software modules provide services to other modules they are designed in a relatively generic format; communication between components is asynchronous and only done when required;
- modularization of software components: ontrol is not programmed for the entire system, rather only for individual components resulting in natural control distribution;
- common communication protocol, which is particularly important since service providers are abstracted from the low level all the way to the high level, so that implementation makes no differentiation of hardware devices or enterprise systems.

Potential qualitative measurements for using SOA are claimed as cost reduction, potential to hire less skilled labor, interoperability (cross-platform and cross-company) and implementation speed.

Candido *et al.* in [73] present a service-oriented infrastructure to support the deployment of evolvable production systems (EPS). This work exploits the association of EPS and SOA paradigms in the pursuit of a common architectural solution to support the different phases of the device lifecycle. The result is a modular, adaptive, and open infrastructure forming a complete SOA ecosystem that will make use of the embedded capabilities supported by the proposed device model.

Mendes *et al.* [74] discuss another approach to SOA in automation in which components in the system (that are referred to as "bots") can be service requesters and providers, either pure software or providing hardware functions such as sensors and actuators. A particular requirement of this work was to maintain compatibility with automation standards IEC 61131-3 and IEC 61499. The composition of a typical bot is defined to have functionality encapsulated into a software module which may be composed of multiple submodules. A framework called the Continuum Bot Framework provides the class description required to realise the abstraction of services into a module. Petri-nets were used as the formal language to develop the bot functionality.

F. Design Patterns and Generative Programming

A design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. A design pattern is not a finished design that can be transformed directly into code but rather a description or template for how to solve a problem that can be used in many different situations.

Dibowski *et al.* [75] propose a novel automatic software design approach for large building automation systems (BAS). The design of large BASs with thousands of devices is a laborious task with a lot of recurrent works for identical automated rooms. The usage of prefabricated off-the-shelf devices and design patterns can simplify this task but creates new interoperability problems. The proposed method covers the device selection, interoperability evaluation, and composition of BASs. It follows a continuous top-down design with different levels of abstraction starting at requirement engineering and ending at a fully developed and industry-spanning BAS design.

Faldella *et al.* [76] introduce a set of new components that abstractly model the behavior of manifold field devices commonly used within automated manufacturing systems, regardless of their nature, intrinsic features, and specific functional purposes. These components, called generalized devices, are basic logic controllers/diagnosers, which play the role of keeping cleanly distinct higher level control policies from low-level mechanisms dealing with actuators and sensors. This development intends to contribute to a reference framework comprising a comprehensive set of highly reusable logic control components that may help the designers in the process of modelling and structuring their applications according to the specific needs.

Serna *et al.* [77] present a way to ease the development of IEC 61499-based applications identifying and characterizing "extended function blocks," adding semantic artefacts to basic function blocks. Specific design patterns are built from those extended function blocks, which match elements of the problem domain. Two specific design patterns are presented which allow dealing with failure management, a common topic in control applications.

The model-view-control (MVC) design pattern [78], was adapted by Christensen in [79] to the domain of industrial automation and integrated with the IEC 61499 standard architecture. According to [33], a software component composed following the MVC pattern is organized from two core subcomponents connected in closed-loop:

- controller, implementing a set of operations, published as services;
- model of the object's behavior;

The combination of these two functions enables simulation of the system in closed loop with the actual, ready-for-deployment control code. Moreover, the simulation model is obtained with a high degree of components' reuse. Additionally, the View component supports interactive simulation by rendering the system's status based on the parameters provided by the Model. Other functions, such as Diagnostics and Database Logger, are also fed by the data from the Object or the Model, and the Human–Machine Interface (HMI) component can be connected in the closed loop with the controller.

The successes in development of component architectures, agents and design patterns gave rise to the development of generative programming applications in automation where the automation software is automatically generated rather than manually written.

For example, Shutzt *et al.* [80] propose a pathway to semiautomatic generation of PLC-based multi-agent control of a flexible manufacturing system. The approach includes first extraction of the machine properties and generation of hierarchical SysML model which is used to instantiate and connect agents into a system. The approach is evaluated on a flexible fixture device.

The earlier mentioned work [23] by Ryssel *et al.* also contributes to the generative approach to automatically create the controls of domain-specific automation systems based on a generic knowledge base.

Dubinin *et al.* [81], address the problem of semantic incompatibilities between different implementations of programming languages, on a particular example of IEC 61499 standard for which several execution models exists. To achieve a degree of execution model independence, design patterns are suggested that make applications robust to changes of execution semantics. A semantic-robust pattern is defined for a particular source execution model. The patterns themselves are implemented by means of the same function block language apparatus and therefore are universal. The patterns can be formally defined and implemented using the function block transformations expressed in terms of attributed graph grammars. An obvious downside of this approach is performance overheads in the transformed applications.

G. Comparisons and Critical Remarks

The area of software design clearly stands out by the number and variety of investigations conducted and results achieved. The downside of this variety is the difficulty to extract clear suggestions and recommendations to practitioners. Table I presents

TABLE I
COMPARISON OF SOFTWARE DESIGN APPROACHES BY THEIR TARGET CHARACTERISTICS

	Lifecycle characteristics					Operation performance	Dependability
	Design effort	Scalability	Interoperability	Flexibility	Distribution		
Model-based engineering	**P**			C			C
Formal models	C						**P**
Multi-agent architectures	C	**P**		C	**P**	C	
Service-oriented architecture	C	C	**P**	C	**P**	C	
Component-based design		C	C	C	**P**		
Design patterns and generative programming	**P**	C					C

Legend: P – primary concern, C – contributes to.

an attempt to collate target characteristics of different design approaches. Arguably, almost all of the characteristics chosen for comparison are interdependent in some cases, nevertheless they seem to represent a possible comparison basis.

When research and piloting of certain concepts reaches some maturity, it is a good idea to proceed with standardization which will add necessary unification and would help practitioners to benefit from the research results. However, the process does not always go in this direction. For example, there was an attempt to standardize the Automation Object concept in IEC standardization activity, but the process got stalled due to insufficient critical mass of development.

VII. SOFTWARE DESIGN STRATEGIES, METHODS, AND SOFTWARE CONSTRUCTION

Examples of software design strategies and methods include function-oriented design, object-oriented design (Section VII-A) and aspect-oriented design (Section VII-B). The former one is quite traditional in PLC programming with functions and function blocks being main reusable program artefacts. However, the limitations of those structures raised interest in more modern design methods discussed as follows.

The term software construction, according to [2], refers to the detailed creation of working, meaningful software through a combination of coding, verification, unit testing, integration testing, and debugging. In particular, the related topics include programming languages (Section VII-C) and the concept of software product line (Section VII-D).

A. Object-Oriented Programming

Object-oriented programming (OOP) is a popular programming paradigm using "objects" to design applications and computer programs. An object stands for a data structure consisting of data fields and data-processing methods. Distinct features of OOP are data abstraction, encapsulation, messaging, modularity, polymorphism, and inheritance. The benefits of using OOP are in a more efficient code reuse and increased safety and stability of software.

Many general-purpose programming languages, such as Java, C++, and Python, support some or all OOP features. As noted above, the third edition of IEC 61131-3 also introduces OOP features. For the time being, CoDeSys is the only known programming environment to support these, therefore it is early to judge whether this paradigm will find broad application in automation. In particular, the OOP concepts of polymorphism and inheritance raise serious concerns in safety and dependability of applications. On the other hand, IEC 61499 provides lightweight OOP features in the new function block concept, which strongly encapsulates data and where events can be regarded as methods.

B. Aspect-Oriented Programming (AOP)

In computing, AOP is a programming paradigm which aims to increase modularity by allowing the separation of cross-cutting concerns. AOP forms a basis for aspect-oriented software development.

Tangermann *et al.* [82] present an aspect-oriented approach for weaving the communication related code into the distributed control application code by means of AspectJ, an extension for AOP with Java. The paper gives a comparison to other approaches.

Wehrmeister [83] applies concepts of the aspect-oriented paradigm in order to improve the treatment of nonfunctional requirements in the design of distributed embedded real-time systems. A tool named GenERTiCA, which generates source code from UML diagrams and weaves aspect adaptations, has been developed to support such an approach. This paper presents results regarding the use of this tool to generate code and implement aspects (from a high-level framework of aspect) for distributed embedded real-time systems.

Binotto *et al.* [84] introduce the use of software aspect-oriented paradigms to perform machines' monitoring and a self-rescheduling strategy of tasks to address nonfunctional timing constraints. As a case study, tasks for a production line of aluminium ingots are designed.

C. Programming Languages

The usual set of programming languages for PLCs is standardized in IEC 61131-3 and includes two textual ones: structured text (ST) and instruction list (IL), and two diagrammatic ones: function block diagrams (FBD), ladder logic diagrams (LLD), and sequential function charts (SFC) for overall configuration. In addition to these, tool vendors and OEMs provide proprietary languages, such as flowcharts and continuous function charts.

The general-purpose programming languages are also used in programming of automation and control applications. Some programming environments, such as ISaGRAF, allow programming in C along with the IEC 61131-3 languages. Lüder *et al.* [85] state that OOP concepts and languages like Java become more and more interesting for all levels of automation, but mainly for nonreal-time applications. Java is a high-level OOP language with a strong type system. Java enables platform-independent application design. Java programs are executable on every platform that can run Java Virtual Machine which opens many possibilities for reusability of code and provides a high stability of applications realized by extensive checks during compile-, load-, and run time. To overcome real-time limitations, there is Real Time Java Specification (RTSJ).

Thramboulidis and Doukas [86] describe an approach for the transparent use of real-time Java in control and automation. The proposed approach, which is in the context of the model integrated mechatronics paradigm, exploits the function block construct for the design model and the real-time Java specification for the implementation model.

Development of many domain-specific languages (DSLs) targeting particular sectors of automation is reported in literature. A typical DSL example is Habitation language for home automation discussed in [87]. DSLs are envisaged in model-driven engineering, aiming at more intuitive descriptions of the system using graphic models, and, as such, facilitating software development by domain specialists rather than software engineers. However, practical impact of DSL in automation does not seem to be significant.

D. Software Product Line (SPL)

The concept of SPLs relying on UML technology have been a breakthrough in software reuse in the IT domain. In [88], SPL is introduced in the industrial automation domain. The object-oriented extensions of IEC 61131-3 are used in SPL architecture and product specifications are expressed as UML class diagrams, which serve as straightforward specifications for configuring PLC control application with OO extensions. A product configurator tool has been developed to support the generation of an executable PLC application according to chosen product options. The approach is demonstrated using a mobile elevating working platform as a case study.

An important challenge is cost-effective migration of legacy systems towards product lines. Breivold and Larsson [89] present a number of specific recommendations for the transition process covering four perspectives: business, organization, product development processes and technology.

VIII. Testing, Management, Evolution and Quality

A. Software Testing

Software testing is the process of revealing software defects and evaluating software quality by executing the software.

Wang and Tan describe fundamentals of software testing for safety critical automation systems in [90]. The software testing comprises of both functional and performance testing. The former includes conventional black-box and white-box testing, while the latter is made up of testing for software availability, reliability, survivability, flexibility, durability, security, reusability, and maintainability. In [90], a case study of automated testing of control rotating-turbine machinery is presented.

As software components become more complex to construct and test, test-driven development (TDD) of software systems has been successfully used for agile development of business software systems. Test cases guide the system implementation and can be executed automatically after software changes (continuous integration and build strategy). Hametner *et al.* [91] investigate application of TDD processes in control automation systems engineering. They introduce an adapted TDD process identifying a set of UML models that enable the systematic derivation of test cases and evaluate the adapted TDD process in an industrial use case. Results of the study show that UML models enabled effective test case derivation.

B. Software Maintenance and Evolution

Software engineering considers the entire life cycle of software which includes its maintenance and evolution. Software maintenance is defined by the IEEE 1219 standard [92] as "the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment."

Bennett and Rajlich [93] define the goal of software evolution as to adapt the application to the ever changing user requirements and operating environment.

PLC software, for example, is commonly maintained by users rather than its developers. This explains the use of such relict programing languages as ladder logic diagrams in many automation projects: programs in this language can be maintained by the factory floor electricians.

Software migration is the process of moving from the use of one operating environment to another operating environment. Migration is an important part of software maintenance. Standards facilitate migration between compliant platforms, but as it is noted e.g., by Bauer [11], there are many compatibility issues between different implementations of IEC 61131-3. Hussain and Frey [94], Dai *et al.* [95], and Wenger *et al.* [96], [97] study the problem of automatic migration from PLCs to IEC 61499.

Code refactoring [98] is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior. Refactoring is commonly used in software maintenance to improve usability of the code. Advantages include improved code readability and reduced complexity to improve the maintainability of the source code, as well as a more expressive internal architecture or object model to improve extensibility. Dubinin *et al.* [99] propose extended refactoring of execution control charts in basic function blocks of the IEC 61499 standard that also aims at fixing semantic problems.

Froschauer *et al.* [100] address the life-cycle management of large-scale, component-based automation systems. The paper presents an approach based on the product line variability models to manage the lifecycle of industrial automation systems and to automate the maintenance and reconfiguration process. The paper suggests complementing the IEC 61499

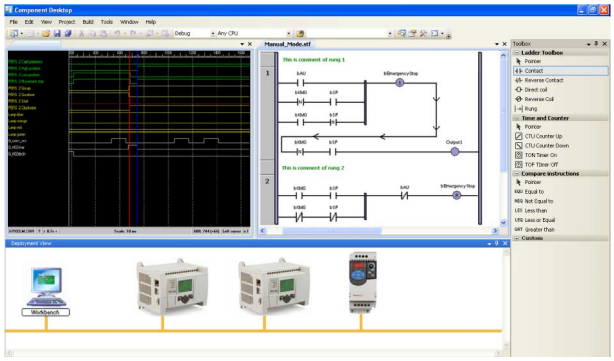

Fig. 4. ISaGRAF workbench screenshot.

standard with a variability modelling approach to support both initial deployment and runtime reconfiguration.

C. Software Configuration and Engineering Management

According to Pressman [101], software configuration management (SCM) is a set of activities designed to control change by identifying the work products that are likely to change, establishing relationships among them, defining mechanisms for managing different versions of these work products, controlling the changes imposed, and auditing and reporting on the changes made.

Driven by the manufacturing agility and reconfigurability challenges [22], Suender *et al.* have developed an original SCM approach called downtimeless evolution in [102]. These works address the often need to change the automation software during the operation of production facility without stopping the automated production process (i.e., without downtime). Implementation of this requirement asks for several fundamental changes on the run-time level and on the tool side, including the need to respect real-time properties of the devices during this process.

Moser *et al.* in [103] present a novel idea of supporting automation systems development with the help of Engineering Cockpit, a social-network-style collaboration platform for automation system engineering project managers and engineers, which provides a role-specific single entry point for project monitoring, collaboration, and management. A prototype implementation of the Engineering Cockpit is presented and discussed. Major results are that the Engineering Cockpit increases the team awareness of engineers and provides project-specific information across engineering discipline boundaries.

D. Software Process and Tools

Design of automation software is supported by integrated development environments (IDEs), provided by the vendors of PLC hardware. Large vendors, such as SIEMENS, Rockwell Automation, Schneider Electric, Omron, and others, develop such tools in house. Many smaller original equipment manufacturers (OEM) license the tools from independent software tools vendors, such as ISaGRAF, 3S Software or KW Software and others, and provide to their customers with or without proprietary camouflage. Fig. 4 presents a snapshot of ISaGRAF IDE v.6.0, where one can see a ladder logic program, data log in form of waveforms, and network of connected devices. The tools of independent vendors are compliant with international standards IEC 61131-3 and IEC 61499. For the latter, ISaGRAF supports both standards which implied some engineering tradeoffs and limitations in IEC 61499 part, while NxtControl has developed a tool NxtStudio that is purely IEC 61499-compliant.

Such standards as IEC 61499 have motivated and provided the opportunity for many researchers and developers to come up with own tools. For example, Thramboulidis and Tranoris [104] presents an engineering tool compliant with the IEC 61499 standard based on a four-layer architecture that successfully unifies the FB concept with UML. Another example is 4DIAC-IDE, an IEC 61499 compliant open-source tool and run-time environment.

E. Software Quality

Functional safety is an important quality characteristic, defined in [105] as a part of the overall system safety relating to the control system which depends on the correct functioning of the programmable electronic safety-related systems, other technology safety-related systems and external risk reduction facilities.

Von Krosigk [106] presents several practices to address functional safety of automation systems software, both proprietary and based on international standards.

The generic standard IEC 61508 [105] and the automation industry-specific standard IEC 62061 "Safety of machinery" address the vast variety of safety related processes, in particular, related to the design pathway from hazard and risk analysis up to the specification of safety requirements of the control system. The IEC 61508 defines safety integrity levels (SIL) that are specified by requirements for the probability of failures of the safety related system. The probability of failure of a system can be calculated with mathematical methods.

Another, even more rigorous quality assurance method is called formal verification. An early survey by Frey [107] overviews formal method use in PLC programming for their verification and validation. Johnson provides [108] an interesting overview of key milestones and prospects for the use of formal verification methods in achieving enhanced dependability of future manufacturing software. Hanisch *et al.* have written a focused survey on formal verification of IEC 61499 [109]. Yoong *et al.* [110] propose implementation of IEC 61499 with the synchronous semantics similar to that of synchronous programming languages such as Esterel. This method allows for combination of benefits from IEC 61499 model-driven design and dependable deterministic execution. However, industrial adoption of such methods is hindered by their high computational complexity, as well as lack of user-friendly tools oriented on automation domain.

IX. Discussion

A. Lessons Learned

The conducted study fully confirms the "hypothesis" on the growing importance of software engineering in automation. Research works in virtually every domain specified by the SWEBOK reference document have been identified with clear dominance of works related to design concepts.

Cost reduction remains the main driving force in automation. As one can conclude from Table I, the development of software engineering areas in automation has been mostly concerned with improvement of software lifecycle efficiency and dependability rather than with performance issues. The same applies to the downstream methods, languages, and processes: very few of them were developed in response to insufficient performance of automation systems. Moreover, as noted by some researchers, many advanced software engineering methods can bring substantial performance overhead.

On the other hand, the application of distributed computing architectures is often driven by performance issues among other motives. The same applies to such architectures, as the multi-agent one, which often aim and achieve performance improvement on account of local data processing.

An important research method in industrial automation is to adopt developments from the general computing area. This is the case for virtually any software-related technology, e.g., component-orientation, service-orientation or model-based engineering, to mention a few. This allows the developers to take advantage of the huge investments into such technologies and rely on proven solutions rather than reinventing the wheel. One of the latest examples of this sort is "reuse" of Semantic Web technologies reported in many automation-related works. These technologies are aiming at knowledge representation and automatic manipulation and can be useful for enhancing the performance of software development processes.

A potential downside of this research method is the slow take-up, or low applicability, of the results in industrial practice. Often, researchers contributing to the progress of industrial automation in this way may have little industrial experience. Examples used in such projects are often from university laboratories. Such research may be the enabler by which new and innovative solutions can emerge, particularly in the long-term, but it may also sometimes suffer from limited industrial applicability and impact. The situation is similar to the gap at times observed between academic computer science and software development practice.

One of the reasons for using Semantic Web technologies in automation was related to the development of SOAs that required mechanisms to declare, discover, and orchestrate services. However, the potential of these technologies in software engineering is much greater. As identified by Breslin *et al.* [24], the value proposition of such technologies is multifold, related to software development phase, infrastructure, information exchange and interoperability, and software behavior. As further exemplified by Gašević *et al.* [111] and Happel *et al.* [112], the mechanism of ontologies, widely used for semantic knowledge representation and management, can be used for generation of software similarly to model-based engineering approach. Initial application of semantics enhanced engineering and model reasoning for control application development has been demonstrated by Hästbacka and Kuikka in [113].

There are many promises in making software solutions more intelligent and easier integrated into systems by using Semantic Web technologies. For example, wider application of intelligent agents with reasoning capabilities will raise the issue of proper knowledge representation for reasoning and semantic interoperability in such multi-agent systems. These can be achieved by using common domain ontologies. Semantic Web technologies provide not only such mechanisms, but are going even further by providing languages such as SWRL to describe reasoning over the knowledge concepts.

B. Industrial Adoption of Advanced Software Concepts

Along with industrial practices, this review discussed several advanced engineering concepts being investigated by researchers. For many of them prospects of their wide industrial adoption are still unclear. Many of such concepts represent disruptive design paradigm shifts therefore their adoption in the industry is quite slow. Such technologies include distributed automation architectures, such as multi-agent, service-oriented, as well as IEC 61499.

In the author's opinion, one of the reasons slowing their adoption is the lack of integral view on the matter leading to confusion among practitioners. Often in the literature these approaches are unnecessarily separated and antagonized, although they rather represent complementary features and their synergies need to be explored. There are corresponding examples. For instance, the function block architecture of IEC 61499 has been used to implement multi-agent control concepts in works by Black [114], Lepuschitz [115] and Hegny [116]. The added value of such a combination is in system level representation of the entire distributed system for verification and validation, and in simplified maintenance and reconfiguration. Similar arguments would apply in the context of service-oriented architectures. This is especially true in view of the number of engineering concepts and tools discussed in this paper as related to maintenance and evolution. The presence of standardized distributed software architecture would drastically simplify industrial application of those advanced software concepts.

Another roadblock is higher complexity and education threshold requiring training of engineers in new unfamiliar software topics. The situation is changing but slowly: the new generation of graduates are often aware of many emergent software technologies and ready to apply them in practice.

C. Open Issues for Future Research

The list of open issues could be long, only the most compelling ones will be discussed.

There are encouraging examples of model-driven software engineering in automation which explore not only software "mainstream" models of UML/SysML, but also "genuine" automation models, such as P&ID. These works surely need to be continued and extended towards formalisation of such models and their "morphing" into software models. Standardization of such diagrammatic specifications on the semantic level is becoming an urgent task.

Also, there are plenty unresolved challenges related to combining legacy PLC programming (e.g., in ladder and function block diagrams) with autonomous behavior of agents, synchronization of distributed processes or knowledge-based reasoning within one programming framework. The challenges concern both expressive power of programming languages along with performance, determinism, and guaranteed reactivity of applications.

The increasing demands of dependability, stemming from various safety requirements and certification procedures, will increase the demand for formal methods application in automation software engineering. This will require a new generation of software tools to make such methods more approachable by control and software engineers.

In the testing domain, usual testing and code debugging techniques need to be integrated with simulation and emulation environments, as well as with formal verification tools. Again, defining common standardized models to bridge control and simulation worlds would be very helpful for enabling such integrated solutions.

Convincing proof of benefits for many presented novel design approaches can be done only in real industrial pilot projects and in comparison with each other and existing technologies. Such developments and comparisons are rarely done yet. It would be the task of major research funding bodies to include the corresponding priorities of comparison and evaluation for several research-mature technologies in the future calls in order to enable such comparative pilot studies, rather than funding development of new fancy ideas. The candidate technologies can include multi-agent control, SOAs, and IEC 61499 to mention a few. The industrial and economic impact of such studies can be very substantial.

On the other hand, practices in many automation companies include the most advanced software engineering methods, but they are rarely reported in academic publications. This situation is being overcome by new instruments such as "tools special sessions" at major international conferences, but more can be done.

X. Conclusion

Despite the colloquial opinion of software design in automation mainly consisting of PLC programming in ladder logic done in front of the machine, the presented survey proves that it is a large and growing area with a rich body of knowledge. One should note that the paper only scratched the surface and (due to size limitations) could not address and reflect all of the related developments in the industrial automation software engineering. Many interesting and important developments were unfortunately left behind. These include, for example, agile development and generative programming. The main purpose of this work was to introduce a reference basis bridging the gap between automation and software engineering worlds. Hopefully, it will be useful for more focused studies in the future.

Software engineering is an integral part of systems engineering, and this is especially true in automation. Proposing holistic system design processes in which software engineering is tightly intertwined is a compelling challenge, but the progress observed and reflected in this review provides convincing evidences in the feasibility of this goal.

References

[1] R. Stetter, "Software Im Maschinenbau -Lästiges Anhängsel Oder Chance Zur Marktführerschaft? VDMA, ITQ," 2011 [Online]. Available: http://www.software-kompetenz.de/servlet/is/21700/Stetter-SW_im_Maschinenbau.pdf?command=downloadContent&filename=Stetter-SW_im_Maschinenbau.pdf

[2] *IEEE Standard Glossary of Software Engineering Terminology*. New York, NY, USA: IEEE, 1990.

[3] "Automation expenditures for discrete industries," ARC, 2012 [Online]. Available: http://www.arcweb.com/market-studies/pages/automation-systems-for-discrete-industries.aspx

[4] *International Standard IEC 62264-1: Enterprise—Control System Integration Part 1: Models and Terminology*. Geneva, Switzerland: Int. Electrotech. Commission, 2003.

[5] X. Xu *et al.*, "STEP-compliant NC research: The search for intelligent CAD/CAPP/CAM/CNC integration," *Int. J. Prod. Res.*, vol. 43, pp. 3703–3743, 2005.

[6] J. K. Chaar, D. Teichroew, and R. A. Volz, "Developing manufacturing control software: A survey and critique," *Int. J. Flexible Manuf. Syst.*, vol. 5, pp. 53–88, 1993.

[7] , A. Abran, J. W. Moore, P. Bourque, R. Dupuis, and L. L. Tripp, Eds., *SWEBOK: Guide to the Software Engineering Body of Knowledge*. Washington, DC, USA: IEEE Comput. Soc., 2004.

[8] A. Dubey, "Evaluating software engineering methods in the context of automation applications," in *Proc. 9th Int. Conf. Ind. Inf.*, 2011, pp. 585–590.

[9] *International Standard IEC 61131-3: Programmable Controller—Part 3: Programming Languages*. Geneva, Switzerland: Int. Electrotech. Commission, 1993, pp. 230–230.

[10] B. Werner, "Object-oriented extensions for IEC 61131-3," *IEEE Ind. Electron. Mag.*, vol. 3, no. 4, pp. 36–39, Dec. 2009.

[11] N. Bauer, R. Huuck, B. Lukoschus, and S. Engell, *A Unifying Semantics for Sequential Function Charts Integration of Software Specification Techniques for Applications in Engineering*, H. Ehrig, W. Damm, J. Desel, M. Große-Rhode, W. Reif, and E. Schnieder, Eds. *et al.* Berlin, Germany: Springer, 2004, vol. 3147, pp. 400–418.

[12] V. Vyatkin, J. H. Christensen, and J. L. M. Lastra, "OOONEIDA: An open, object-oriented knowledge economy for intelligent industrial automation," *IEEE Trans. Ind. Inf.*, vol. 1, no. 1, pp. 4–17, Feb. 2005.

[13] J. Virta, I. Seilonen, A. Tuomi, and K. Koskinen, "SOA-Based integration for batch process management with OPC UA and ISA-88/95," in *Proc. IEEE Conf. Emerging Technol. Factory Autom.*, Bilbao, Spain, 2010, pp. 1–8.

[14] *International Standard IEC 61512–1: Batch Control—Part 1: Models and Terminology*. Geneva, Switzerland: Int. Electrotech. Commission, 1997.

[15] *International Standard IEC614999-1: Function Blocks—Part 1 Architecture*, First ed. Geneva, Switzerland: Int. Electrotech. Commission, 2005.

[16] V. Vyatkin, "IEC 61499 as enabler of distributed and intelligent automation: State of the art review," *IEEE Trans. Ind. Inf.*, vol. 7, no. 4, pp. 768–781, Nov. 2011.

[17] C. Diedrich, F. Russo, L. Winkel, and T. Blevins, "Function block applications in control systems based on IEC 61804," *ISA Trans.*, vol. 43, pp. 123–131, 2004.

[18] *International Standard IEC 61850: Communication Networks and Systems in Substations—Part 7, Basic Information and Communication Structure*. Geneva, Switzerland: Int. Electrotech. Commission, 2003.

[19] G. Zhabelova and V. Vyatkin, "Multi-agent smart grid automation architecture based on IEC 61850/61499 intelligent logical nodes," *IEEE Trans. Ind. Electron.*, vol. 59, no. 5, pp. 2351–2362, May 2011.

[20] N. Higgins, V. Vyatkin, N. K. C. Nair, and K. Schwarz, "Distributed power system automation with IEC 61850, IEC 61499, and intelligent control," *IEEE Trans. Syst., Man Cybern. C, Applic. Rev.*, vol. 41, no. 1, pp. 81–92, Jan. 2011.

[21] T. Strasser, M. Stifter, F. Andren, D. B. de Castro, and W. Hribernik, "Applying open standards and open source software for smart grid applications: Simulation of distributed intelligent control of power systems," in *Proc. IEEE Power Energy Soc. General Meeting*, 2011, pp. 1–8.

[22] H. ElMaraghy, "Flexible and reconfigurable manufacturing systems paradigms," *Int. J. Flexible Manuf. Syst.*, vol. 17, pp. 261–276, 2005.

[23] U. Ryssel, H. Dibowski, and K. Kabitzsch, "Generation of function block based designs using Semantic Web technologies," in *IEEE Conference on Emerging Technologies and Factory Automation (ETFA'09)*, 2009.

[24] J. G. Breslin, D. O'Sullivan, A. Passant, and L. Vasiliu, "Semantic Web computing in industry," *Comput. Industry*, vol. 61, pp. 729–741, 2010.

[25] T. Hadlich *et al.*, "Time as non-functional requirement in distributed control systems," in *Proc. IEEE Conf. Emerging Technol. Factory Autom.*, 2012.
[26] O. Ljungkrantz, K. Åkesson, M. Fabian, and C. Yuan, "Formal specification and verification of industrial control logic components," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 3, pp. 538–548, Jul. 2010.
[27] S. Runde, A. Fay, and W. O. Wutzke, "Knowledge-based Requirement-Engineering of building automation systems by means of Semantic Web technologies," in *Proc. 7th IEEE Int. Conf. Ind. Inf.*, Cardiff, Wales, 2009, pp. 267–272.
[28] M. Hirsch*, Systematic Design of Distributed Industrial Manufacturing Control Systems*. Berlin, Germany: Logos Verlag, 2010.
[29] C. Secchi, M. Bonfe, C. Fantuzzi, R. Borsari, and D. Borghi, "Object-oriented modeling of complex mechatronic components for the manufacturing industry," *IEEE/ASME Trans. Mechatron.*, vol. 12, no. 6, pp. 696–702, Dec. 2007.
[30] K. C. Thramboulidis, "Using UML in control and automation: A model driven approach," in *Proc. 2nd IEEE Int. Conf. Ind. Inf.*, 2004, pp. 587–593.
[31] V. Dubinin, V. Vyatkin, and T. Pfeiffer, "Engineering of validatable automation systems based on an extension of UML combined with function blocks of IEC 61499," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2005, pp. 3996–4001.
[32] K. Thramboulidis, "Model-integrated mechatronics: Toward a new paradigm in the development of manufacturing systems," *IEEE Trans. Ind. Inf.*, vol. 1, no. 1, pp. 54–61, Feb. 2005.
[33] V. Vyatkin, H. Hanisch, C. Pang, and C. Yang, "Closed-loop modeling in future automation system engineering and validation," *IEEE Trans. Syst., Man Cybern. C, Applic. Rev.*, vol. 39, no. 1, pp. 17–28, 2009.
[34] *IEC SB3/TC 65 Working Draft: Automation Objects for Industrial-Process Measurement and Control Systems*, , 2002.
[35] R. Brennan, L. Ferrarini, J. L. M. Lastra, and V. Vyatkin, "Automation objects: Enabling embedded intelligence in real-time mechatronic systems," *Int. J. Manuf. Res.*, vol. 1, pp. 379–381, 2006.
[36] T. Ritala and S. Kuikka, "UML automation profile: Enhancing the efficiency of software development in the automation industry," in *Proc. 5th IEEE Int. Conf. Ind. Inf.*, 2007, pp. 885–890.
[37] U. Katzke and B. Vogel-Heuser, "Combining UML with IEC 61131-3 languages to preserve the usability of graphical notations in the software development of complex automation systems," *Anal., Design, Evaluation of Human-Mach. Syst.*, pp. 90–94, 2007.
[38] P. Mohagheghi and V. Dehlen, "Where is the proof?-A review of experiences from applying MDE in industry," *Model Driven Architecture-Foundations and Applications*, pp. 432–443, 2008.
[39] B. Vogel-Heuser, M. Obermeier, S. Braun, K. Sommer, F. Jobst, and K. Schweizer, "Evaluation of a UML-based versus an IEC 61131-3-Based software engineering approach for teaching PLC programming," *IEEE Trans. Education*, vol. PP, no. 99, 2012.
[40] D. Witsch and B. Vogel-Heuser, "Close integration between UML and IEC 61131-3: New possibilities through object-oriented extensions," in *Proc. IEEE Conf. Emerging Technol. Factory Autom.*, 2009, pp. 1–6.
[41] M. Witsch and B. Vogel-Heuser, "Towards a formal specification framework for manufacturing execution systems," *IEEE Trans. Ind. Inf.*, vol. 8, no. 2, pp. 311–320, May 2012.
[42] K. Thramboulidis and G. Frey, "An MDD process for IEC 61131-based industrial automation systems," in *Proc. IEEE 16th Conf. Emerging Technol. Factory Autom.*, Toulouse, France, 2011, pp. 1–8.
[43] *IEC International Standard IEC 62424: Representation of Process Control Engineering—Requests in P&I Diagrams and Data Exchange Between P&ID Tools and PCE-CAE Tools*. Geneva, Switzerland: Int. Electrotech. Commission, 2008.
[44] L. Hundt, R. Drath, A. Lüder, and J. Peschke, "Seamless automation engineering with automationML," in *14th International Conference on Concurrent Enterprising (ICE 2008)*, Lisboa, Portugal, 2008, pp. 685–692.
[45] "XML Formats for IEC 61131-3," PLCopen Technical Commitee 6, 2009 [Online]. Available: http://www.plcopen.org/pages/tc6_xml/downloads/tc6_xml_v201_technical_doc.pdf
[46] E. Estévez, M. Marcos, and D. Orive, "Automatic generation of PLC automation projects from component-based models," *Int. J. Adv. Manuf. Technol.*, vol. 35, pp. 527–540, 2007.
[47] E. Estevez and M. Marcos, "Model based validation of industrial control systems," *IEEE Trans. Ind. Inf.*, vol. 8, no. 2, pp. 302–310, May 2011.
[48] S. Vinoski, "CORBA: Integrating diverse applications within distributed heterogeneous environments," *IEEE Commun. Mag.*, vol. 35, pp. 46–55, 1997.
[49] R. Sessions*, COM and DCOM: Microsoft's Vision for Distributed Objects*. New York, NY, USA: Wiley, 1997.
[50] I. Crnkovic and M. Larsson, "A case study: Demands on component-based development," in *Proc. Int. Conf. Software Eng.*, 2000, pp. 23–31.
[51] I. Calvo, F. Perez, I. Etxeberria, and G. Moran, "Control communications with DDS using IEC61499 service interface function blocks," in *Proc. IEEE Conf. Emerging Technol. Factory Autom.*, 2010, pp. 1–4.
[52] J. Camerini, A. Chauvet, and M. Brill, "Interface for distributed automation: IDA," in *Proc. 8th IEEE Int. Conf. Emerging Technol. Factory Autom.*, 2001, vol. 2, pp. 515–518.
[53] K. Trkaj, "Users introduce component based automation solutions," *Computing Control Eng. J.*, vol. 15, pp. 32–37, 2004.
[54] C. R. Maga, N. Jazdi, and P. Göhner, "Reusable models in industrial automation: Experiences in defining appropriate levels of granularity," in *Proc. 18th World Congress IFAC*, 2011, pp. 9145–9150.
[55] L. Szer-Ming, R. Harrison, and A. A. West, "A component-based distributed control system for assembly automation," in *Proc. 2nd IEEE Int. Conf. Ind. Inf.*, 2004, pp. 33–38.
[56] T. Strasser *et al.*, "Structuring of large scale distributed control programs with IEC 61499 subapplications and a hierarchical plant structure model," in *Proc. IEEE Int. Conf. Emerging Technol. Factory Autom.*, Sep. 15–18, 2008, pp. 934–941.
[57] H.-M. Hanisch, "Closed-loop modeling and related problems of embedded control systems in engineering," in *Proc. ASM Abstract State Machines: Advances in Theory and Practice*, Lutherstadt Wittenberg, 2004, pp. 24–28.
[58] G. Cutts and S. Rattigan, "Using Petri Nets to develop programs for PLC systems," in *Application and Theory of Petri Nets*, K. Jensen, Ed. Berlin, Germany: Springer, 1992, vol. 616, pp. 368–372.
[59] G. Frey, "Automatic implementation of Petri net based control algorithms on PLC," in *Proc. Amer. Control Conf.*, Chicago, 2000, pp. 2819–2823.
[60] J. Thieme and H. M. Hanisch, "Model-based generation of modular PLC code using IEC61131 function blocks," in *Proc. IEEE Int. Symp. Ind. Electron.*, 2002, vol. 1, pp. 199–204.
[61] L. E. Pinzon, H. M. Hanisch, M. A. Jafari, and T. Boucher, "A comparative study of synthesis methods for discrete event controllers," *Formal Methods in System Design*, vol. 15, pp. 123–167, 1999.
[62] T. Winkler, H. C. Lapp, and H. M. Hanisch, "A new model structure based synthesis approach for distributed discrete process control," in *Proc. 9th IEEE Int. Conf. Ind. Inf.*, Lisbon, Portugal, 2011, pp. 527–532.
[63] M. Wooldridge*, An Introduction to Multi-Agent Systems*. New York, NY, USA: Wiley, 2002.
[64] P. Leitão, "Agent-based distributed manufacturing control: A state-of-the-art survey," *Eng. Applic. Artif. Intell.*, vol. 22, pp. 979–991, 2009.
[65] M. Metzger and G. Polaków, "A survey on applications of agent technology in industrial process control," *IEEE Trans. Ind. Inf.*, vol. 7, no. 3, pp. 570–581, Aug. 2011.
[66] M. Pechoucek and V. Marík, "Industrial deployment of multi-agent technologies: Review and selected case studies," *Autonomous Agents and Multi-Agent Syst.*, vol. 17, pp. 397–431, 2008.
[67] S. Theiss, V. Vasyutynskyy, and K. Kabitzsch, "Software agents in industry: A customized framework in theory and praxis," *IEEE Trans. Ind. Inf.*, vol. 5, no. 1, pp. 147–156, Feb. 2009.
[68] V. V. Herrera, A. Bepperling, A. Lobov, H. Smit, A. W. Colombo, and J. Lastra, "Integration of multi-agent systems and service-oriented architecture for industrial automation," in *Proc. 6th IEEE Int. Conf. Ind. Inf.*, 2008, pp. 768–773.
[69] J. L. M. Lastra and M. Delamer, "Semantic web services in factory automation: Fundamental insights and research roadmap," *IEEE Trans. Ind. Inf.*, vol. 2, no. 1, pp. 1–11, Feb. 2006.
[70] T. Erl*, Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice-Hall PTR, 2005.
[71] F. Jammes and H. Smit, "Service-oriented paradigms in industrial automation," *IEEE Trans. Ind. Inf.*, vol. 1, no. 1, pp. 62–70, Feb. 2005.
[72] A. Cannata, M. Gerosa, and M. Taisch, "SOCRADES: A framework for developing intelligent systems in manufacturing," in *Proc. IEEE Int. Conf. Ind. Eng. Eng. Manag.*, 2008, pp. 1904–1908.

[73] G. Candido, A. W. Colombo, J. Barata, and F. Jammes, "Service-Oriented infrastructure to support the deployment of evolvable production systems," *IEEE Trans. Ind. Inf.*, vol. 7, no. 4, pp. 759–767, Nov. 2011.

[74] J. M. Mendes, A. Bepperling, J. Pinto, P. Leitao, F. Restivo, and A. W. Colombo, "Software methodologies for the engineering of service-oriented industrial automation: The continuum project," in *Proc. 33rd Annu. IEEE Int. Comput. Software Applic. Conf.*, 2009, pp. 452–459, .

[75] H. Dibowski, J. Ploennigs, and K. Kabitzsch, "Automated design of building automation systems," *IEEE Trans. Ind. Electron.*, vol. 57, no. 11, pp. 3606–3613, Nov. 2010.

[76] E. Faldella, A. Paoli, A. Tilli, M. Sartini, and D. Guidi, "Architectural design patterns for logic control of manufacturing systems: The generalized device," in *Proc. XXII Int. Symp. Inf., Commun. Autom. Technol.*, 2009, pp. 1–7.

[77] F. Serna, C. Catalan, A. Blesa, and J. M. Rams, "Design patterns for failure management in IEC 61499 function blocks," in *Proc. IEEE Conf. Emerging Technol. Factory Autom.*, 2010, pp. 1–7.

[78] T. M. H. Reenskaug, "Model-View-Controller design pattern," Apr. 2013 [Online]. Available: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

[79] J. H. Christensen, "Design patterns for systems engineering with IEC 61499," in *Proc. Verteilte Automatisierung—Modelle und Methoden für Entwurf, Verifikation, Eng Instrum.*, Magdeburg, Germany, Mar. 22–23, 2000, pp. 63–71.

[80] D. Schutz, M. Schraufstetter, J. Folmer, B. Vogel-Heuser, T. Gmeiner, and K. Shea, "Highly reconfigurable production systems controlled by real-time agents," in *Proc. 16th IEEE Conf. Emerging Technol. Factory Autom.*, 2011, pp. 1–8.

[81] V. Dubinin and V. Vyatkin, "Semantics-robust design patterns for IEC 61499," *IEEE Trans. Ind. Inf.*, vol. 8, no. 2, pp. 279–290, May 2012.

[82] M. Tangermann, C. Schwab, A. Kalogeras, K. Lorentz, and A. Prayati, R. Meersman, Z. Tari, and ed, Eds., in *Proc. OTM Workshops Aspect-Orientation of Control Application Code for Distributed Automation Systems: The TORERO Approach on the Move to Meaningful Internet Syst.*, Berlin, Germany, 2003, vol. 2889, pp. 335–345, Springer.

[83] M. A. Wehrmeister, E. P. Freitas, C. E. Pereira, and F. Rammig, "GenERTiCA: A tool for code generation and aspects weaving," in *Proc. 11th IEEE Int. Symp. Object Oriented Real-Time Distrib.*, 2008, pp. 234–238.

[84] A. Binotto, E. P. Freitas, C. E. Pereira, and T. Larsson, "Towards dynamic task scheduling and reconfiguration using an aspect oriented approach applied on real-time concerns of industrial systems," in *Proc. Inf. Control Problems Manuf.*, Moscow, Russia, 2009, pp. 1423–1428.

[85] A. Luder, J. Peschke, and M. Heinze, "Control programming using Java," *IEEE Ind. Electron. Mag.*, vol. 2, pp. 19–27, 2008.

[86] K. Thramboulidis and A. Zoupas, "Real-time Java in control and automation: A model driven development approach," in *Proc. 10th IEEE Conf. Emerging Technol. Factory Autom.*, Catania, Italy, 2005, pp. 8–46.

[87] M. Jimenez, F. Rosique, P. Sanchez, B. Alvarez, and A. Iborra, "Habitation: A domain-specific language for home automation," *IEEE Software*, vol. 26, pp. 30–38, 2009.

[88] N. Papakonstantinou, S. Sierla, and K. Koskinen, "Object oriented extensions of IEC 61131-3 as an enabling technology of software product lines," in *Proc. 16th IEEE Conf. Emerging Technol. Factory Autom.*, Toulouse, France, 2011, pp. 1–8.

[89] H. P. Breivold, S. Larsson, and R. Land, "Migrating industrial systems towards software product lines: Experiences and observations through case studies," in *Proc. 34th Euromicro Conf. Software Eng. Adv. Applic.*, 2008, pp. 232–239.

[90] L. F. Wang, K. C. Tan, X. D. Jiang, and Y. B. Chen, "A flexible automatic test system for rotating-turbine machinery," *IEEE Trans. Autom. Sci. Eng.*, vol. 2, no. 1, pp. 1–18, Jan. 2005.

[91] R. Hametner, D. Winkler, T. Ostreicher, S. Biffl, and A. Zoitl, "The adaptation of test-driven software processes to industrial automation engineering," in *Proc. 8th IEEE Int. Conf. Ind. Inf.*, 2010, pp. 921–927.

[92] *IEEE Standard for Software Maintenance*, IEEE Std 1219-1998, 1998, .

[93] V. T. Rajlich and K. H. Bennett, "A staged model for the software life cycle," *Computer*, vol. 33, pp. 66–71, 2000.

[94] T. Hussain and G. Frey, "Migration of a PLC controller to an IEC 61499 compliant distributed control system: Hands-on experiences," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2005, pp. 3984–3989.

[95] W. Dai and V. Vyatkin, "Redesign distributed PLC control systems using IEC 61499 function blocks," *IEEE Trans. Autom. Science Eng.*, vol. 9, no. 2, pp. 390–401, Apr. 2012.

[96] C. Suender, M. Wenger, C. Hanni, I. Gosetti, H. Steininger, and J. Fritsche, "Transformation of existing IEC 61131-3 automation projects into control logic according to IEC 61499," in *Proc. IEEE Int. Conf. Emerging Technol. Factory Autom.*, 2008, pp. 369–376.

[97] M. Wenger, A. Zoitl, C. Sunder, and H. Steininger, "Transformation of IEC 61131-3 to IEC 61499 based on a model driven development approach," in *Proc. 7th IEEE Int. Conf. Ind. Inf.*, 2009, vol. 1,2, pp. 715–720.

[98] M. Fowler and K. Beck, *Refactoring: Improving the Design of Existing Code*. Reading, MA, USA: Addison-Wesley, 1999.

[99] V. Vyatkin and V. Dubinin, "Refactoring of execution control charts in basic function blocks of the IEC 61499 standard," *IEEE Trans, Ind. Inf.*, vol. 6, no. 2, pp. 155–165, May 2010.

[100] R. Froschauer, D. Dhungana, and P. Grunbacher, "Managing the life-cycle of industrial automation systems with product line variability models," in *Proc. 34th Euromicro Conf. Software Eng. Adv. Applic.*, 2008, pp. 35–42.

[101] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed. New York, NY, USA: McGraw-Hill, 2001.

[102] O. Hummer, C. Sunder, A. Zoitl, T. Strasser, M. N. Rooker, and G. Ebenhofer, "Towards zero-downtime evolution of distributed control applications via evolution control based on IEC 61499," in *Proc. IEEE Conf. Emerging Technol. Factory Autom.*, 2006, pp. 1285–1292.

[103] T. Moser, R. Mordinyi, D. Winkler, and S. Biffl, "Engineering project management using the engineering cockpit: A collaboration platform for project managers and engineers," in *Proc. 9th IEEE Int. Conf Ind. Inf.*, Lisbon, Portugal, 2011, pp. 579–584.

[104] K. C. Thramboulidis and C. S. Tranoris, "Developing a CASE tool for distributed control applications," *Int. J. Adv. Manuf. Technol.*, vol. 24, pp. 24–31, 2004.

[105] "International standard: IEC 61508 functional safety of electrical electronic programmable electronic safety related systems," in *Part1-Part7*. Geneva, Switzerland: Int. Electrotech. Commission, 1999–2000.

[106] H. Von Krosigk, "Functional safety in the field of industrial automation. The influence of IEC 61508 on the improvement of safety-related control systems," *Computing Control Eng. J.*, vol. 11, pp. 13–18, 2000.

[107] G. Frey and L. Litz, "Formal methods in PLC programming," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 2000, vol. 4, pp. 2431–2436.

[108] T. L. Johnson, "Improving automation software dependability: A role for formal methods?," *Control Eng. Practice*, vol. 15, pp. 1403–1415, 2007.

[109] H. M. Hanisch, M. Hirsch, D. Missal, S. Preuße, and C. Gerber, "One decade of IEC 61499 modeling and verification-results and open issues," in *Proc. Inf. Control Problems Manuf. Conf.*, Moscow, Russia, 2009, pp. 211–216.

[110] L. H. Yoong, P. S. Roop, V. Vyatkin, and Z. Salcic, "A synchronous approach for IEC 61499 function block implementation," *IEEE Trans. Comput.*, vol. 58, no. 12, pp. 1599–1614, Dec. 2009.

[111] D. Gašević, N. Kaviani, and M. Milanović, *Ontologies and Software Engineering Handbook on Ontologies*, S. Staab, D. R. Studer, and ed, Eds. Berlin, Germany: Springer, 2009, pp. 593–615.

[112] H.-J. Happel and S. Seedorf, "Applications of ontologies in software engineering," in *Proc. 2nd Int. Workshop Semantic Web Enabled Software Eng/5th Int. Semantic Web Conf.*, Athens, GA, USA, 2006, pp. 5–9.

[113] D. Hästbacka and S. Kuikka, "Semantics enhanced engineering and model reasoning for control application development," *Multimedia Tools Applic.*, vol. 65, no. 1, pp. 47–62, Jul. 2013.

[114] G. Black and V. Vyatkin, "Intelligent component-based automation of baggage handling systems with IEC 61499," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 2, pp. 337–351, Apr. 2010.

[115] W. Lepuschitz, A. Zoitl, M. Valle, and M. Merdan, "Toward self-reconfiguration of manufacturing systems using automation agents," *IEEE Trans. Syst., Man, Cybern. C, Applic. Rev.*, vol. 41, no. 1, pp. 52–69, Jan. 2011.

[116] I. Hegny, O. Hummer, A. Zoitl, G. Koppensteiner, and M. Merdan, "Integrating software agents and IEC 61499 realtime control for reconfigurable distributed manufacturing systems," in *Proc. Int. Symp. Ind. Embedded Syst.*, 2008, pp. 249–252.

Valeriy Vyatkin (SM'04) received the Engineer degree in applied mathematics, Ph.D., and Dr. Sci. degrees from Taganrog State University of Radio Engineering, Taganrog, Russia, in 1988, 1992, and 1998, respectively. and the Dr.Eng. degree from Nagoya Institute of Technology, Nagoya, Japan, in 1999.

He is the Chaired Professor of Dependable Computation and Communication Systems at Luleå University of Technology, Luleå, Sweden, and a Visiting Scholar with Cambridge University, Cambridge, U.K., on leave from The University of Auckland, Auckland, New Zealand, where he has been an Associate Professor and Director of the InfoMechatronics and Industrial Automation lab (MITRA) with the Department of Electrical and Computer Engineering. His previous faculty positions were with Martin Luther University of Halle-Wittenberg in Germany (Senior Researcher and Lecturer, 1999–2004), and with Taganrog State University of Radio Engineering, Taganrog, Russia (Associate Professor and Professor, 1991–2002). His research interests are in the area of dependable distributed automation and industrial informatics, including software engineering for industrial automation systems, distributed architectures, and multi-agent systems applied in various industry sectors: smart grid, material handling, building management systems, and reconfigurable manufacturing. He is also active in research on dependability provisions for industrial automation systems, such as methods of formal verification and validation, and theoretical algorithms for improving their performance.

Prof Vyatkin was the recipient of the Andrew P. Sage Award for Best IEEE Transactions Paper in 2012.