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

Dai, Wenbin; Riliskis, Laurynas; Wang, Peng; Vyatkin, Valeriy; Guan, Xinping

A Cloud-Based Decision Support System for Self-Healing in Distributed Automation Systems Using Fault Tree Analysis

Published in:
IEEE Transactions on Industrial Informatics

DOI:
[10.1109/TII.2018.2791503](https://doi.org/10.1109/TII.2018.2791503)

Published: 01/03/2018

Document Version
Peer reviewed version

Please cite the original version:
Dai, W., Riliskis, L., Wang, P., Vyatkin, V., & Guan, X. (2018). A Cloud-Based Decision Support System for Self-Healing in Distributed Automation Systems Using Fault Tree Analysis. *IEEE Transactions on Industrial Informatics*, 14(3), 989-1000. <https://doi.org/10.1109/TII.2018.2791503>

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

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

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

A Cloud-based Decision Support System for Self-Healing in Distributed Automation Systems using Fault Tree Analysis

Wenbin Dai, *Senior Member, IEEE*, Laurynas Riliskis, Peng Wang,
Valeriy Vyatkin, *Senior Member, IEEE*, Xinping Guan, *Fellow, IEEE*

Abstract – Downtime is a key performance index for industrial automation systems. An industrial automation system achieves maximum productivity when its downtime is reduced to the minimum. One approach to minimize downtime is to predict system faults and recover from them automatically. A cloud-based decision support system is proposed for rapid problem identifications and to assist the self-management processes. By running multiple parallel simulations of control software with real-time inputs ahead of system time, faults could be detected and corrected automatically using autonomous industrial software agents. Fault trees, as well as control algorithms, are modeled using IEC 61499 function blocks that can be directly executed on both physical controllers and cloud services. A case study of water heating process is used to demonstrate the self-healing process supported by the cloud-based decision support system.

Index Terms — Cloud-based Decision Support Systems; Distributed Automation Systems; Programmable Logic Controllers; IEC 61499 Function Blocks; Fault Tree Analysis; Faster-than-real-time Simulation; Supervisory Control; Self-Healing.

I. INTRODUCTION

Industrial automation systems have been seriously influenced in the last decade by new information and communication technologies (ICT) such as multi-agent systems [1], service-oriented architecture [2], cloud computing [3] and Internet-of-Things [4]. The introduction of new ICT into automated manufacturing processes increases their abilities to adapt more intelligently to the changing conditions and requirements. On the other hand, the complexity of control software is continuously increasing not only due to the growing physical dimensions of manufacturing plants but also due to their increasing functionalities. For example, at large manufacturing plants, control software is often distributed across some controllers to meet the real-time requirements.

Copyright (c) 2009 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org

W. Dai and X. Guan are with Shanghai Jiao Tong University, China, 200240. (e-mail: w.dai@ieee.org, xpguan@sjtu.edu.cn)

L. Riliskis is with the Harmony.ai, USA, 94104. (e-mail: laurynas.riliskis@gmail.com)

P. Wang is with Shenyang Institute of Automation, Chinese Academy of Science, 110016. (e-mail: wangpeng@sia.cn)

V. Vyatkin is with Lulea University of Technology, Sweden, 971 87 and Aalto University, Finland, 02150. (e-mail: vyatkin@ieee.org)

Safety and reliability are the key factors to ensure productivity of industrial automation systems. A single point of failure in large manufacturing processes can result in massive economic losses. For example, during a continuous chemical process, if a failure happens at a middle step, all raw materials being processed are wasted due to strict process control guidelines and requirements. Also, system failure may cause site hazards and threat safety of site workers, for example, chemical materials may get exploded. Furthermore, when a system failure happens, site engineers need to identify problems (like hardware failure or software bugs) rapidly to avoid long downtime. With inexperienced engineers, the control software issues could take days to be identified. To avoid such situations, industrial control systems shall be able to predict failures, identify reasons and prevent from happening by reconfiguring control software automatically. This research is focusing on minimizing system downtime by quickly identifying reasons before they actually have happened. In addition, if failures could be prevented, online reconfigurations of control software must be taken automatically.

To achieve real-time problem identification and resolution, the control software of industrial automation systems must be self-manageable and adapt to external environmental changes. Upon detection of a potential failure, the system would reconfigure its control logic dynamically to perform self-healing. A decision support system is required to provide the faster-than-real-time prediction based on operating control algorithms and real-time inputs from the shop floor. The faster-than-real-time simulation refers to a model of a physical system that can execute faster than actual clock time. When a failure is detected, the decision support must identify its source through the analysis process which shall be customizable for various industrial applications and could be inferred using the existing control software code. Finally, to ensure the decision support systems are synchronized with the physical plant state, the decisions must be made within the same scan cycle update of inputs from the shop floor.

In this paper, a cloud-based decision support system is proposed for performing faster-than-real-time hardware-in-the-loop emulation. The cloud can provide sufficient computing power to ensure the real-time constraints for self-healing could be met. The fault tree analysis (FTA) technique is adopted for the decision making processes. A single modeling language is used to create both fault trees and control algorithms. Both models could be easily extended or modified

without affecting the emulation operations.

The rest of the paper is organized as follows: the related literature is reviewed in section II. In section III, a cloud-based decision support system architecture is proposed for self-manageable distributed automation systems. We describe the FTA techniques and the fault tree modeling methodologies in section IV. Followed by that, self-healing using FTA and cloud-based decision support system is proposed in Section V. Section VI describes a parallel simulation method for improving real-time performance of the cloud-based decision support system. A case study of the water processing system is used to demonstrate the self-healing mechanism by the cloud-based decision support system in section VII. Finally, conclusions and future work directions are provided.

II. RELATED WORKS

Application of emerging ICT in factory automation has been a popular trend in the industrial informatics research. Cloud computing is considered as the key for enabling collaborations between smart devices in future distributed automation systems [5].

II. A INDUSTRIAL CLOUD

Cloud technologies are already experimented by several researchers in the industrial automation domain. A service cloud named IMC-AESOP is proposed by Karnouskos et al. in [6] to compose services from the sensors and actuators level up to the enterprise resource planning (ERP) level. Multi-system interactions and collaborations rely on the service-oriented architecture paradigm. Cross-layer interoperability in the service cloud is achieved between all layers in automation systems including sensors/actuators, controllers, SCADA systems, manufacturing execution systems (MES) and ERP systems.

The integration of Internet-of-Things with cloud computing is proposed by Wang et al. [7] to assist computer-aided design (CAD) software generate assembly plan for complex products. The proposed automatic assembly modeling system is based on modularized system components using object-oriented architecture. However, the cloud is only used for large storage.

Decision support systems are widely used in industrial applications especially in the energy domain. A cloud-based decision support system is proposed for adaptive energy information systems by Nikolopoulos et al. [8]. The authors suggested a web-based knowledge system based on the distributed cloud infrastructure. The authors also investigated a middleware, agent-based clustering and data mining for smart metering. Lee et al. [9] adopted decision support systems in similar manners for diabetes diagnosis. A five-layer fuzzy ontology and a semantic decision support agent were used to construct new knowledge. Diabetes symptoms were defined using the ontology web language (OWL) and a construction mechanism.

Xu et al. proposed a universal data access method for Internet-of-Things (IoT) and cloud-based healthcare information system [10]. A semantic data model is used to store and interpret sensor data; meta-models are defined in

ontology and XML. The cloud-based information system has three layers: business layer, resource layer, and tenant database layer. The resource model is designed specifically for emergency medical services. Authors use a proprietary model format that cannot be easily ported to other domains.

II. B SELF-MANAGEABLE AGENTS

Self-manageable and adaptive automation systems are supported by industrial software agents [12]. Leitao *et al.* [11] propose a multi-agent system solution for self-adaptive manufacturing systems. The preliminary results show improvement of production efficiency and costs reduction. Mubarak et al. [13] proposed an agent-oriented approach for achieving self-management in industrial automation systems. A generic agent-based concept that integrates with self-management functions is introduced, and the self-healing function is demonstrated with a passenger lift example. However, that work did not cover the issue of integrating agents with industrial controllers.

Mendes et al. [14] proposed a service-oriented multi-agent method for self-manageable automation systems. Resources are shared as services between agents in automation systems. Modularity, flexibility, and interoperability are enhanced by using SOA for software agents. Self-reconfigurations are achieved in the proposed SoMAS framework.

Kaindl et al. [15] proposed another agent-based method for providing self-configuration and autonomous monitoring. The agent is defined as a combination of several hardware and software components. In this implementation, low-level control functions are implemented using IEC61499 service interface function blocks [35]. The self-reconfiguration is achieved by creating new agents or modifying existing agents that are both defined using IEC 61499 function blocks. However, dynamic reconfiguration of IEC61499 Function Blocks(FB) using the management commands was not covered.

Zidan et. al. [16] proposed a cooperative framework for self-healing in smart grids. By using agent technologies, this framework is capable of identifying and isolating faults in the grid and recovering required loads by switching operations. A two-way communication protocol is used to coordinate multiple agents. Eriksson et. al. [17] proposed a similar idea of achieving self-healing in smart grids by using distributed multi-agent that provides fault location, isolation, and service restoration. In both approaches, agents are deployed locally in distributed control systems.

Self-healing as an essential characteristic of software agents and it is already experimented by many researchers as shown in [11-17]. Industrial software agents have been demonstrated their capability of providing the self-healing majorly in smart grids and energy systems. Current implementations are often not linked with the automation software platforms, and the interface between agents and controllers may cause extra communication latency. Deployment of software agents on existing programmable controllers is challenging from the performance perspective.

II. C FAULT-TREE ANALYSIS

Finally, FTA is a well-developed methodology for understanding how a system can fail [18] [19]. It has been applied in safety and reliability engineering such as energy systems and nuclear plants [20] [21] [22]. It is also used for reliability analysis in industrial automation by many researchers. Hussain et al. proposed a methodology for automatic generation of fault trees for networked control systems [23]. The generated fault tree models can be formally verified to improve safety and availability. The limitation of this approach is that all verification must be done offline. For self-healing, the verification process must be able to run online along with controllers.

FTA shows its impacts on many other application domains as well. For example, Fiontorou *et al.* provided a knowledge-based approach for real-time diagnosis of insulator semiconductor field effect transistor-based biosensor systems [24]. Fault trees are built using component-based models and reasoning through a fuzzy-rules-based inference engine. Mo et. al. proposed a multiple-valued decision-diagram-based method for constructing dynamic fault trees [25]. By using multiple-valued decision diagram, the efficiency of computing reliability on large dynamic trees is improved. Moreover, the multiple-valued decision tree can also be integrated with industrial control systems.

Overall, existing cloud-based decision support systems provide comprehensive support for enterprise, management and supervision levels in distributed automation systems. Agent-based solutions enable self-management by communicating with low-level controllers. To extend existing works, low-level control software is deployed to the cloud service along with agent-based decision support system. Faster-than-real-time simulation of multiple instances of low-level control software on the cloud service are setup for self-healing with supervision of industrial software agents. Finally, low-level control software and failure detection will be defined using the same modeling language to assist agent-based decision support system.

III. SELF-MANAGEABLE DISTRIBUTED AUTOMATION SYSTEMS WITH CLOUD SUPPORT

Reliability is one of the key factors of industrial automation systems as a single component failure could cost a fortune and threaten lives of site workers. One big challenge of achieving reliable industrial automation systems is to predict and prevent failures from happening. Multi-agent systems (MAS) are experimented with industrial automation systems by many researchers for self-management purposes [26]. However, integrating agents on device level is limited by available resources on controllers. Heavy data processing tasks such as querying knowledge base require massive computing power and memory that is beyond capabilities of existing PLCs. To achieve self-manageable automation systems, external resources are expected to support real-time software agents on the device level.

In the previous work [27], a flexible and interoperable execution environment for distributed automation system is introduced by adopting the service-oriented architecture (SOA). Each function block is considered as a software service

that is accessible from all other FBs as services by exchanging messages. Following that work, an autonomic service manager (ASM) is proposed at the PLC runtime level for enabling self-management features [28]. Self-properties including self-configuration, self-healing and self-optimization are demonstrated on the device level with coordinations of the ASM. For time-critical features such as self-protection, the ASM must response to symptoms generated from PLC programs within one Input/Output (I/O) update cycle. A typical I/O update cycle contains computation time and communication time. For process control and building automation systems, this cycle time could be in the order of several hundred milliseconds (ms). For material handling systems and manufacturing systems, 20 to 30 milliseconds update rate is required. For motion control systems, the typical update rate is between 4 and 5 ms [29]. Slow response time could lead the ASM out of sync with control software and physical plants. With increasing size of knowledge bases and complexities of the inference algorithms, the ASM will not be able to make decisions prior to next input update.

To assist time-critical tasks in the ASM, the cloud support is introduced. A decision support system is usually running as an independent computer or server that provides limited computing power. With complex decision-making processes, the traditional DSS cannot meet the real-time constraints of industrial automation systems. The cloud provides massive computing power for computation of intelligence. Parallel simulations could be adopted to speed up decision-making processes to the milliseconds level. There are a couple of assumptions made in this case. Firstly, the cloud service must always be available, and Internet connection from control systems to the cloud has to be guaranteed. Secondly, the connection speed between control systems and the cloud service must use broadband or other faster connections. Besides, the requirement of QoS gave rise to research and developed on scaled cloud platforms and fog computing which can be brought as close to the level of field control devices as necessary to guarantee the required QoS. The approach proposed in this paper is entirely compatible with such architectures.

The Software-as-a-Service (SaaS) model is selected for the cloud DSS. Each component in the ASM, as well as PLC runtimes, are running as software services in the cloud. The entire self-management framework is shifted from the SOA-based execution environment on the device level to the cloud-based decision support system (DSS) as illustrated in Fig. 1. The ASM acts as a supervisor that is constantly monitoring and collecting operational data from PLCs. The ASM reads inputs simultaneously as PLCs from industrial fieldbuses and is capable of overriding PLC decisions by forcing the PLC outputs and modifying parameters in the control software. By deploying ASM on the device level, some self-management features are feasible. However, time critical features such as self-healing still cannot be achieved due to the limited resources of existing PLCs. To predict industrial automation systems failures in real-time, using cloud computing for real-time (or faster than real-time) simulation to assist self-management decision making is a feasible solution.

Fig. 1: Supervisory Control with Cloud-based DSS for PLCs

The self-healing is defined as “automatic discovery and correction of faults” in autonomic computing [30]. In industrial automation domain, self-healing refers to the discovery of potential errors that may cause system malfunctions and prevent it from happening by dynamically reconfiguring control applications. Such errors could be triggered by hardware failures including sensors and actuators, network failures such as disconnections or software bugs caused by both syntactic and semantic errors. The fault detection and prediction process using the cloud-based decision support system is shown in Fig. 2.

In a PLC scan (as illustrated on the left-hand side), PLC reads inputs from industrial fieldbuses, executes deployed algorithms and updates outputs at the end of each scan. Two extra steps are introduced: outputs are overridden by the cloud-based decision support system if needed, and a copy of current output image is sent to the cloud service at the end of each scan for verification purposes. The cloud-based DSS is shown on the right side in Fig. 2.

Fig. 2: Fault Detection and Prediction Supervisory Control with Cloud-Based Decision Support Systems.

To keep synchronization with PLC operations, a parallel PLC execution environment is running on the cloud side. Inputs are read by both physical and simulated runtime simultaneously. Simulated outputs are fed into fault tree analyzer in the decision support system to search for potential faults. Fault tree models are pre-defined in the knowledge base. Details of fault modeling technique will be described in the next section. If any pre-defined fault is found by FTA, a corresponding change plan will be fetched from the knowledge base and current PLC output values will be overridden by the cloud supervisor. Besides, a sequence of IEC 61499 management commands are sent to the resource service for reconfiguration of control software programs if the change of software is required for resolving the issue. Finally, the cloud-based DSS will compare simulated and actual outputs to identify any mismatch. Alarms will be composed to inform operators that program running in PLC is inconsistent to its simulated version. The DSS will re-sync application between cloud-based simulated execution environment and physical PLCs. The details will be provided in section V.

IV. FAULT MODELLING

To utilize FTA techniques for self-healing, gaps between control algorithms and fault trees need to be filled. In existing PLC implementations, faults are raised from supervisory control and data acquisition (SCADA) systems by polling alarm signals from PLC variables. Those alarm signals are manually programmed by engineers for identifying sources of faults. Site operators and maintenance engineers usually recover systems by manually clearing all faults notified by SCADA systems. To prevent the faults occurring, diagnostics must be performed by DSS automatically which means fault models shall be executable on the cloud directly and models could be easily understood and reconfigured.

Decision trees are widely adopted in the operations research. A decision tree is a model of decisions that is structured in a flow chart [31]. One application of decision trees in ICT is the FTA. The fault tree defines three types of symbols: event, gate, and transfer [18]. Event symbols consist of basic events (that refer to input failure events from external sources), intermediate events (as a combination of basic and intermediate faults) and top events (that normally used for describing a system failure). Gate symbols contain Boolean algebra such as *AND*, *OR*, *Combination*, *Exclusive OR*, *Inhibit* and *Priority AND* gates. Transfer symbols are used for indicating corresponding symbols on other locations.

To apply FTA in industrial automation domain, event symbols are redefined as: *Basic* events representing individual device fault signals from original control programs; *Intermediate* faults referring to faults caused by multiple individual faults, for example, a zone failure. Finally, system-level or isolated sub-system failures are indicated by top events.

There is a gap in application of fault tree models in industrial automation systems as these models cannot be executed directly on industrial controllers. Modeling fault trees using the existing PLC programming languages is a promising way to bridge this gap. Nowadays, the majority of industrial

automation systems are running IEC 61131-3 based PLC systems [32]. However, using IEC 61131-3 standard for modeling system faults is not sufficient due to the limitation of its software model: the top-level entity is limited to a single controller. For modeling large distributed automation systems, it is not possible to use the IEC 61131-3 standard alone. Another option is to model fault trees using computer programming languages and connects to PLCs using OPC connections [33]. Data mappings between PLCs and PCs must be manually created where errors are often introduced. The last option is to apply the IEC 61499 standard that is commonly used as a modelling language for distributed systems [34]. All logic must be encapsulated in function block (FB) models for distribution in the IEC 61499 standard. Function blocks are transferred to target devices according to the deployment configurations automatically. Compared to IEC 61131-3 based systems, the IEC 61499 standard provides better support for distribution of PLC programs in the cloud environment [35].

The basic mapping rules between elements of fault trees and IEC61499 function blocks are defined as follows.

(1) Events are represented by Boolean variables

As shown in Fig. 3 below, *Basic*, *Condition* and *External* events refer to the function block input variables while *undeveloped* events are mapped to the function block output variables. *Intermediate* events are used as both input variables (From *Basic* events) and output variables (To other *Intermediate* or *Undeveloped* events).

Event Type	Symbol
Basic Event
Condition Event
External Event
Undeveloped Event
Intermediate Event

Fig. 3: Event Symbols Definitions.

Another possibility is to model all fault events as IEC 61499 events. As clarified in the 2nd edition of the IEC 61499 standard, a function block instance can only be activated by one event input simultaneously. When two or more fault events raise simultaneously, non-deterministic behaviours may occur due to various execution semantics. When input events are queued, an individual internal flag needs to be set for remembering the status of input events. If input events are not queued, only one event input will appear, and all other events will be lost. As a result, using Boolean data variables is a better modeling approach for faults from both platform-independence and efficiency perspectives.

(2) Gate symbols are mapped to function blocks

AND, *OR* and *Exclusive OR* gate are mapped to predefined service interface function block (SIFB) types *AND*, *OR* and *XOR* directly. These bitwise operation SIFBs take two or more

inputs (*Basic* or *Intermediate* events) and generate an *Intermediate* or *Undeveloped* event by applying *AND*, *OR* and *XOR* Boolean algebra. *Priority AND*, *Inhibit* gates conjoint with *Condition* events are treated slightly different. As conditions vary case by case, a generic function block type cannot cover all conditions. A specific basic function block (BFB) is created for every *Priority AND* gate and *Inhibit* gate.

Logic Gate Type	Symbol
AND Gate
OR Gate
Exclusive OR Gate
Priority AND Gate
Inhibit Gate

Fig. 4: Logic Gate Symbols Definitions.

For *Priority AND* gate, inputs must appear in a certain order to trigger output fault that the sequence of inputs is defined by a *Condition* event. The template of *Priority AND* basic function block is given in the Fig. 5. The *Condition* event is converted to an execution control chart (ECC) in the new BFB as illustrated in Fig. 5 (B).

Fig. 5: *Priority AND* with *Inhibit* Function Block Template.

The equivalent number of FB data inputs are created for EC states. Fault tree events are used as EC transition conditions. EC states are ordered according to *Condition* events. For the *Inhibit* gate, similar BFB is used here. However, only two EC states are defined: *WAIT* and *EXECUTE*. *Inhibit* BFBs remain in the *WAIT* state until input event *REQ* is triggered. The condition is then checked and a fault signal is emitted when the condition is matched. Once the output event *CNF* is emitted, the *Inhibit* BFB will jump back to the *WAIT* state. Both BFBs

have an initial state that will reset fault event.

(3) Transfer symbols are replaced by *PUBLISH* and *SUBSCRIBE* service interface function blocks

If events are sent from other resources ($\blacktriangle$ transfer in, $\blacktriangle$ transfer out), *SUBSCRIBE* SIFBs are used to receive external data. On the other hand, if fault events are about to be sent to external resources, *PUBLISH* SIFBs are applied. The *PUBLISH* and the *SUBSCRIBE* SIFB design is given in Fig. 6 where fault event is sent via data variables *FB1.RD_1* (Publish) and *FB0.SD_1* (Subscribe).

Fig. 6: Publish and Subscribe SIFBs for Transfer Gates.

V. SELF-HEALING BASED ON SUPERVISORY CONTROL USING CLOUD-BASED DECISION SUPPORT SYSTEM

In this section, self-healing of distributed automation systems will be illustrated based on supervisory control using the proposed fault tree models and the cloud-based DDS. In legacy SCADA systems, supervisory control is achieved by overwriting parameters stored in PLC internal variables via OPC connections or proprietary communication protocols. Under current implementations, control code and POU's cannot be modified using the legacy SCADA systems. Regarding performance, OPC uses client/server communication model which OPC server needs to read values of variables from each OPC client (embedded in PLCs) in sequential order (as cycles). In each scan cycle, there are thousands of variables need to be updated from multiple PLCs. The processing time may account to several seconds. This is not sufficient for real-time supervisory control tasks such as self-healing that requires millisecond level response time. To meet real-time requirements of self-management properties, supervisory control process with the cloud-based DSS and IEC 61499 is proposed as shown in the Fig. 7.

During the system design stage, fault trees are generally created by software engineers. An example of a fault tree is shown in Fig. 7 (A) where: $EG1 = EB1 + EB2$; $EG2 = EB3$; $ES1 = EG1 \times EG2$. The fault tree is mapped to IEC 61499 function blocks according to the rules provided in the previous section. When the system configuration is deployed, all control FBs and fault tree FBs (one "OR" FB "*FB_EQG1*" and one "AND" FB "*FB_SYSF1*") are deployed to the simulated PLC runtime running on the cloud service. When the system starts operation, simulated PLCs read inputs from Ethernet-based fieldbuses, execute FB network and generate simulated outputs and fault events. Those fault events are logged into the knowledge base in the cloud-based DSS automatically.

As proposed in the previous work [28], the ASM on the decision support system keeps monitoring those events and

generates symptoms by applying semantic rules defined in the Semantic Web Rule Language (SWRL) based on the simulated outputs. Faults will be analyzed by searching through the predefined knowledge base in the ASM, and a corresponding change plan will be created.

Fig. 7: Supervisory Control Process using Cloud-based DSS and IEC 61499.

According to the change plan, actions are taken by sending IEC 61499 management commands to the physical device to override outputs or reconfigure function block networks. The resource service receives requests from the ASM and reconfigures FBs and parameters of FBs to provide supervisory control.

The current work demonstrates that the cloud-based decision support system subscribes to Ethernet-based fieldbuses such as ProfiNet [36], EtherCAT [37] and Ethernet/IP [38]. I/O modules are distributed and connected to simulated PLCs via Ethernet-based fieldbuses. For other non-Ethernet based communication technologies, a wrapper of an IEC 61499 service interface function block is required to bridge between the runtime and I/O modules.

The connection of the cloud-based DSS into the same fieldbus ensures inputs are synchronized between PLCs and cloud-based DSS. Most fieldbuses' outputs can only be updated by one PLC to avoid concurrent accesses from multiple PLCs for security purposes. Instead of writing outputs back to fieldbuses directly, cloud-based DSS provides supervisory control by override decisions made from PLCs. Existing control systems will not be aware that the cloud-based DSS exists.

An IEC 61499 execution environment (Function Block Service Runtime – FBSRT) is developed based on the concept of service-oriented architecture in the previous work [27]. The FBSRT can be run on physical controllers as well as on the cloud. Each function block instance is setup as a separated software service in FBSRT. The ASM and other functions on the cloud-based DSS are also implemented as services so they can access function block instances directly on the cloud.

Fig. 8: Decision Support System Architecture.

The architecture of the cloud-based DSS is illustrated in Fig. 8. It contains three major components: the communication handler, the simulated PLC runtime stack, and the decision support manager. The communication handlers are responsible for polling inputs from Ethernet-based fieldbuses. The polling interval shall be identical to the refresh rate in physical PLCs.

The ASM creates a separate service for each type of fieldbus. Apart from I/O handling, multiple instances of PLC runtime are operating in parallel inside the simulated PLC runtime stack. Whenever I/O handlers read inputs, updated values will be passed to a simulated PLC runtime and simulated outputs are generated after logic execution is completed. Finally, updated outputs will be written back to I/O modules by PLCs. Except communicating with industrial fieldbuses, all other services are communicated using web service protocols.

VI. RUNNING PARALLEL SIMULATION SYSTEMS ON CLOUD-BASED DECISION SUPPORT SYSTEM

Multiple PLC runtime instances could be placed on the stack in the cloud-based DSS as illustrated in Fig. 8. Using scalability of the cloud, numerical instances of the simulated PLC's can be run in parallel. The supervisory control provided by the cloud-based DSS is synchronous with the update rate of the industrial fieldbus connected:

$$T_{SC} < T_{RPI}$$

Where T_{SC} is the execution time of supervisory control process and T_{RPI} is the update rate of industrial fieldbus.

The execution time of supervisory control is defined as:

$$T_{SC} = T_{COMI} + T_{SIM} + T_{ASM} + T_{COMO}$$

Where T_{COMI} is the communication overhead for reading all inputs; T_{SIM} is the execution time of simulated PLC logic; T_{ASM} is the processing time of the autonomic service manager including knowledge base searching time and T_{COMO} is the communication overhead for sending commands back to physical PLC.

In a PLC scan, the total execution time consists of three parts:

$$T_{RPI} = T_{COMI}' + T_{ALG} + T_{COMO}'$$

Where T_{COMI}' is the communication overhead for reading all inputs; T_{ALG} is the execution time of PLC logic and T_{COMO}' is the communication overhead for update all outputs.

Assuming communication overheads are equivalent on both PLCs and the cloud service:

$$\text{Assume } T_{COMI}' = T_{COMI} \text{ and } T_{COMO}' = T_{COMO}$$

$$T_{COMI} + T_{SIM} + T_{ASM} + T_{COMO} < T_{COMI}' + T_{ALG} + T_{COMO}'$$

$$T_{SIM} + T_{ASM} < T_{ALG}$$

Sum of simulation time T_{SIM} and self-healing process time T_{ASM} must be less than logic execution time T_{ALG} on physical PLC to guarantee supervisory control is in sync with a plant. The execution time of the autonomic service management cannot be reduced as monitor, analyze, plan and execute function must operate in sequence. However, simulation time could be saved significantly by running parallel simulations of PLC runtime on the cloud service.

The parallel simulation is illustrated in Fig. 9. A separate runtime instance is created for every possible combination of inputs. Assume there is number N of digital inputs and M of analog inputs scheduled on fieldbus, total possible input

combinations can occur can be calculated as:

$$N^2 \times M^X \text{ and } X = \frac{|V_{max}-V_{min}|}{Step}$$

Where V_{max} is the maximum range of the analog input value; V_{min} is the minimum range of the analog input value; $Step$ is the minimum scale of the analog input and X is the resolution of the analog input.

Fig. 9: Parallel Simulation of PLC runtime on the cloud service.

Executing parallel runtime instances with input combinations generates all potential output combinations in the next scan – that is called a step. Inputs are encoded into a single array of 4 bytes (32-bit integer). When inputs from next scan arrive, the ASM compares real inputs with the list of input arrays and rapidly pick up the corresponding set of simulated outputs as shown in Fig. 9. All other runtime instances with other combinations in the same step are eliminated and one further step will be simulated. In this case, simulation time T_{SM} could be reduced dramatically. The minimum simulation step required to be retained on the cloud DSS is two: one current step and one next step. For example, for a medium size of an industrial automation system with 500 digital inputs and 50 analog inputs with a resolution of 10, there are 4.8×10^{24} instances required for two steps. By removing unused branches of runtime instance in each simulation step, only 2.4×10^{24} instances are required that could save 50% of the cloud resources. To further reduce the required computing power on the cloud, the range of analog input data could be reduced. For instance, the value of a temperature sensor will only vary within several degrees in a day. Therefore, it is not necessary to keep all possible values of an analog input from negative maximum to positive maximum. This could save up to 90% of resources on the cloud as analog inputs consume much more resources compared to digital inputs. At the current stage, the synchronization between multiple PLC runtimes is not achieved and will be investigated in the future.

VII. CASE STUDY

To test the proposed cloud-based DSS, a water heating test bed is set up. The test bed configuration is provided in Fig. 10. A FESTO water processing station is wired to 3 Advantech I/O modules. A Beckhoff CX-2020 PLC [39] (1.4 GHz Intel Celeron CPU with 2GB RAM) is connected to those I/O

modules via EtherCAT fieldbus that has an update rate of 30ms [37]. Firstly, a local cloud with Intel Core i7-3770k (3.5GHz Quad-core), 16GB RAM and 1TB hard disk is setup. The local cloud server has two Ethernet adapters, one is connected to EtherCAT fieldbus and the other is connected to the Beckhoff PLC. The server is using Ubuntu Server 14.04 LTS (Long Term Support) with MAAS (Metal as a Service) [40] and OpenStack [41] to provide cloud computing platform. On the Beckhoff PLC, the IEC 61499 runtime function block service runtime (FBSRT) [27] is deployed. The decision support system is deployed to the cloud server as many services including communication handling service, PLC simulation service, autonomic service manager service and knowledge base service as shown in Fig. 8 previously.

Fig. 10: Test Bed Hardware Configuration.

The piping and instrumentation diagram (P&ID) of the modified FESTO water processing station is illustrated in Fig. 11. Cold water is fed into the makeup water tank (B300) and supplied to the preheater tank (B100) via valve Y101. The water in the preheater tank is preheated to a temperature close to the boiling point and then feed into the boiler water tank (B200) by using a control valve Y102 and water pump M100. A flow measurement indicator F100 is continuously measuring flow rate to avoid excessive water flow. Two high (Analog Lx00, Digital Lx01) and a low water level indicator (Lx02) are installed on all tanks to monitor the water level. A temperature indicator (Tx00) is also installed for both the preheater tank and the boiler to detect overheating hazard. As a pressure indicator P200 is mounted with the boiler to ensure the pressure inside the boiler stays in the safe range. Finally, boiled water will be supplied to the customers via the valve Y401. Wastewater could be disposed via the valve Y103.

In the hot water supply sequence, water heated in the B100 tank is pumped into the boiler tank B200, then go through valve Y202 and Y401 (simulated water supply) and finally refill back into the preheater tank. The interlock control is implemented for safety purposes. For example, when the water level of the tank is too high, the related in-feed valve will be closed automatically and alternatively, when the water level is too low, the outfeed valve will be closed. Although basic interlock is achieved, not all scenarios could be considered during the design stage, especially faults caused by a consequence of steps.

Fig. 11: Water Heating Test Bed P&ID.

One critical fault identified during commissioning stage is that when the water levels of the B100 and the B200 water tank are both high, the safety valve Y201 and Y202 are closed and the water pump M100 is turned off simultaneously. This leads to a gridlock situation that water flow of the entire primary line will be stopped. To solve this fault, a new symptom defined in a SWRL rule is inserted into the knowledge base of the autonomic service manager. The fault tree definition is given in Fig. 12 where:

$Fault_PrimaryLineFull =$
 $B100TankControl.WaterLevelSensorHHAAlarm \cap$
 $B200TankControl.WaterLevelSensorHHAAlarm$

When a gridlock is detected during parallel simulation in the cloud, the ASM generates a change plan and sends the following management command to open the valve of the dispose water line to the *Resource* service on the physical PLC runtime:

```
<Request ID="1" Action="SET">
  <FB Name="Y103Control" Type="Service_DOControl">
    <Parameter Reference="OM" Value="TRUE" />
    <Parameter Reference="OM_ON" Value="TRUE" />
  </FB>
</Request>
```

This command will override decision from sequence control function block and force the valve Y103 open to dispose of water in the pre-heater tank. Once water level in the B100 tank is lowered and high water level alarm disappears, the gridlock will be resolved by interlocking control function block automatically. The ASM will remove overrides by sending the following management command to the *Resource* service on the physical PLC:

```
<Request ID="2" Action="RESET">
  <FB Name="Y103Control" Type="Service_DOControl">
    <Parameter Reference="OM" Value="FALSE" />
    <Parameter Reference="OM_ON" Value="FALSE" />
  </FB>
```

</Request>

Fig. 12: Self-Healing Example of Water Heating Test Bed.

After the local cloud is tested, the public cloud of Microsoft Azure is used to repeat the test. Preliminary evaluations for self-healing capabilities are analyzed to verify the behaviour of proposed the cloud-based DSS using the water processing station on both local and public cloud services.

Firstly, real-time constraints of self-healing mechanism are verified by measuring the response time of the proposed cloud-based DSS.

RESPO NSE TIME	LOCAL CLOUD COMPUT ATIONAL TIME (ms)	LOCAL CLOUD RESPONSE TIME (ms)	PUBLIC CLOUD COMPUT ATIONAL TIME (MS)	PUBLIC CLOUD RESPON SE TIME (MS)
MIN	21	1	5	40
AVG	26	1.6	6.2	56.7
MAX	31	3	8	84
STD. DEV	3.16	0.7	1.01	11.28

Fig. 13: Self-Healing Response Time Analysis.

As shown in the Fig. 13, two tests are performed: one with a computer as the cloud (local) and the other with Microsoft Azure cloud service (public). With 100 samples of self-healing experiments, the average computation time is approximately 26ms with the Core i7-3770k desktop computer with a maximum time of 31ms, a minimum time of 21ms and standard deviation of 3.16. The communication time varies

between 1ms and 3ms with an average of 1.6ms. From the results, as target fieldbus update rate is less than 35ms, the cloud-based decision support system is capable of handling self-healing in real-time constraints. Although this number is applicable for a large number of industrial automation applications, some systems required high processing speed, such as motion control systems (typically 4-5ms update rate), cannot be handled yet.

From the public cloud tests, the response is also divided into two parts: computation time and response time. From the computation time results, the response time is relatively stable with minimum 5ms, maximum 8ms and standard deviation of 1.01. When the network latency is introduced, the standard deviation increases to 11.28 caused by 40ms to 84ms communication delays between the public cloud and the local PLCs. As the target of this work is to prove self-management based on real-time simulation by cloud support, improving response time is not covered in this paper. Assuming public cloud has low network latency, it will also able to handle self-management before target cycle time.

To test scalability, the self-healing example is created 10, 100, 1000 and 10000 copies for both local cloud and public cloud and response time is measured as in Fig. 14. With the local cloud, the computing power will reach a limit with 100 or more copies that cause computational time to rise dramatically. However, this is not affected by the public cloud as there is enough computing power available.

NUM OF COPIES	LOCAL CLOUD COMPUTATIONAL TIME (ms)	PUBLIC CLOUD COMPUTATIONAL TIME (MS)
10	25	6
100	120	75
1000	30120	110
10000	667854	200

Fig. 14: Scalability Response Time Analysis.

Secondly, downtime analysis is conducted. Downtime is a key performance index for manufacturing plants as less downtime will result in higher production efficiency. To measure downtime of the water processing station with/without the proposed cloud-based decision support system, two tests are performed: One recover by the operator intervention, and the other recovers automatically. Downtime report is collected from SCADA system (as indicated in Fig. 15) after 20 tests, and the self-manageable cloud-based decision support system reduces the downtime of water processing system by approximately 89%. The human intervention is mainly eliminated so that downtime caused by system failures is largely reduced from minute level to millisecond level. However, since the SCADA update rate is limited to 1s, the cloud-based downtime is shown as 1s although it is significantly shorter.

DOWNTIME HUMAN OPERA	DOWNTIME ME CLOUD SUPPORT	THROUGHPUT FROM PLCs TO CLOUD	THROUGHPUT UT FROM CLOUD DSS
----------------------	---------------------------	-------------------------------	------------------------------

	TOR (MANUAL)	(AUTO)	DSS	TO PLCs
MIN	4s	1s	3.2K/s	0k/s
MAX	15s	1s	3.2K/S	32K/S
AVG	8.5s	1s	3.2K/S	0.5K/S

Fig. 15: Self-Healing Downtime and Throughput Analysis.

From throughput between the cloud DSS and the PLCs, two measurements are taken: from PLCs to the Cloud DSS and vice versa. From PLCs to the cloud DSS, the throughput is constant as sampled input size remains the same during the monitoring process. On the other hand, the minimum throughput is 0k/s as there is no action required for self-healing. The maximum throughput varies based on the decision made in the cloud.

Compared to existing embedded MAS approaches, cloud-based DSS provides excessive computing power and storage to assist self-manageable and adaptive industrial automation systems. The proposed architecture enables rapid and flexible fault detection by integrating control software with fault trees seamlessly. With the support of knowledge bases and software agents, diagnosis of new faults could be extended dynamically without affecting normal operation. However, real-time constraints are affected by network latency between the cloud and controllers: Public cloud provides unlimited computational resources but longer communication delay and lower stability; Local cloud provides limited computing power but higher stability and minimal network latency.

From two tests presented, one key feature of self-management – self-healing is achieved by adopting cloud-based DSS. Assuming the low network latency, one can conclude that, agent-based approach with cloud support could provide real-time supervisory control based on parallel simulation of control software running on the cloud. The proposed ASM with knowledge support provides a flexible infrastructure to extend self-management features.

VIII. CONCLUSIONS AND FUTURE WORK

A cloud-based DSS is proposed to provide real-time supervisory control for industrial automation systems. The ASM and the knowledge base are shifted from PLCs to the cloud for real-time decision making. One of the self-management features, self-healing, is accomplished by introducing FTA techniques into the cloud-based DSS. To seamlessly integrate the fault tree with the PLC control logic, the IEC 61499 standard is used as the fault modeling language. The FTA combined with the ASM perform necessary configuration changes to maintain system operation. Parallel faster-than-real-time simulations are applied to improve the performance of the cloud-based DSS to meet real-time requirements for time-critical industrial applications. The cloud-based DSS is deployed to a local server as well as the public cloud with a water processing station. From the self-healing tests, one can conclude that the proposed DSS using the local cloud is capable of providing real-time supervisory control for both manufacturing and process control systems. With the public cloud, the response time is primarily affected

by the network latency but it is still capable of meeting most of the requirements to process control systems.

There are several limitations of the cloud-based DSS that shall be addressed in the future research. First, fault tree models shall be imported directly from the design documents using model-driven engineering approaches. In the existing approach, engineers need to develop program models of fault trees manually. By adopting model-driven engineering methods, fault trees could be generated automatically. Secondly, the number of parallel simulation runtimes could be reduced significantly by defining proper maximum and minimum values for every analog input. The range of analog input values could also be fetched from the system design model. Finally, more self-management features such as self-configuration, self-protection, and self-optimization shall be addressed by integrating the cloud-based DSS with industrial software agents.

IX. ACKNOWLEDGEMENTS

This work was supported, in part, by National Science Foundation, China (No. 61503247), Shanghai Pu Jiang Scholarship 2015 (No. 15PJ1403400) and the DIACPA project of Swedish research council (Vetenskapsrådet).

REFERENCES

- [1] P. Vrba, V. Marik, P. Siano, P. Leitao, G. Zhabelova, V. Vyatkin, T. Strasser, "A Review of Agent and Service-Oriented Concepts Applied to Intelligent Energy Systems", *IEEE Transactions on Industrial Informatics*, Vol. 10, Issue 3, page 1890 – 1903, 2014
- [2] F. Jammes and H. Smit, "Service-oriented paradigms in industrial automation", *IEEE Transactions on Industrial Informatics*, Vol. 1, Issue 1, page 62 – 70, 2005
- [3] H. Schaffer, "X as a Service, Cloud Computing, and the Need for Good Judgment", *IT Professional*, Vol. 11, Issue 5, Page 4 -5, 2009
- [4] M. Feki, F. Kawsar, M. Boussard and L. Trappeniers, "The Internet of Things: The Next Technological Revolution", *Computer*, Vol. 46, Issue 2, page 24 – 25, 2013
- [5] O. Givehchi, H. Trsek and J. Jaspemeite, "Cloud computing for industrial automation systems – A comprehensive overview", *18th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, page 1 -4, 2013
- [6] S. Kamouskos, A. Colombo, T. Bangemann, K. Manninen, R. Camp, M. Tilly, P. Stluka, F. Jammes, J. Delsing and J. Eliasson, "A SOA-based architecture for empowering future collaborative cloud-based industrial automation", *38th Annual Conference on IEEE Industrial Electronics Society (IECON)*, page 5766 – 5772, 2012
- [7] C. Wang, Z. Bi and D. Li, "IoT and Cloud Computing in Automation of Assembly Modeling Systems", *IEEE Transactions on Industrial Informatics*, Vol. 10, Issue. 2, page 1426 – 1434, 2014
- [8] V. Nikolopoulos, G. Mpardis, I. Giannoukos and I. Lykourantzou, "Web-based decision-support system methodology for smart provision of adaptive digital energy services over cloud technologies", *IET Software*, Vol. 5, Issue 5, page 454 – 465, 2011
- [9] C. Lee and M. Wang, "A Fuzzy Expert System for Diabetes Decision Support Application", *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, Vol. 41, No. 1, page 139 - 153, 2011
- [10] B. Xu, Li. Xu, H. Cai, C. Xie, J. Hu and F. Bu, "Ubiquitous Data Accessing Method in IoT-Based Information System for Emergency Medical Services", *IEEE Transactions on Industrial Informatics*, Vol. 10, No. 2, page 1578 – 1586, 2014
- [11] P. Leitao, N. Rodrigues, C. Turrin and A. Pagani, "Multiagent System Integrating Process and Quality Control in a Factory Producing Laundry Washing Machines", *IEEE Transactions on Industrial Informatics*, Vol.11, No. 4, page 879 – 886, 2015
- [12] S. Pech, "Software Agents in Industrial Automation Systems", *IEEE Software*, Vol. 30, Issue 3, page 20 – 24, 2013
- [13] H. Mubarak and P. Gohner, "An Agent-Oriented Approach for Self-Management of Industrial Automation Systems", *8th IEEE International Conference on Industrial Informatics*, page 721 – 726, 2010
- [14] J. Mendes, P. Leitao, F. Restivo and A. Colombo, "Service-Oriented Agents for Collaborative Industrial Automation and Production Systems", *Holonic and Multi-Agent Systems for Manufacturing*, Springer, page 13 – 24, 2009
- [15] H. Kaindl, M. Vallee and E. Arnautovic, "Self-Representation for Self-Configuration and Monitoring in Agent-Based Flexible Automation Systems", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Vol. 43, No. 1, page 164 – 175, 2013
- [16] A. Zidan and E. F. El-Saadany, "A Cooperative Multiagent Framework for Self-Healing Mechanisms in Distribution Systems", *IEEE Transactions on Smart Grid*, Vol. 3, No. 3, page 1525 – 1539, 2012.
- [17] M. Eriksson, M. Armendariz, O. Vasilenko, A. Saleem and L. Nordstrom, "Multiagent-Based Distribution Automation Solution for Self-Healing Grids", *IEEE Transactions on Industrial Electronics*, Vol. 62, No. 4, page 2620 – 2628, 2015.
- [18] W. Vesely, F. Goldberg, N. Roberts and D. Hassl, "Fault Tree Handbook", *U.S. Nuclear Regulatory Commission*, 209pp, 1981
- [19] W. S. Lee, D. Grosh, F. Tillman and C. Lie, "Fault Tree Analysis, Methods, and Applications: A Review", *IEEE Transactions on Reliability*, Vol. 34, No. 3, page 194 – 203, 1985.
- [20] S. Cheng, B. Lin, B. Hsu and M. Shu, "Fault-tree analysis for liquefied natural gas terminal emergency shutdown system", *Elsevier Expert Systems with Applications*, Vol. 36, No. 9, page 11918 – 11924, 2009.
- [21] A. Volkanovski, M. Cepin and B. Mavko, "Application of the fault tree analysis for assessment of power system reliability", *Elsevier Reliability Engineering and System Safety*, Vol. 94, No. 6, page 1116 – 1127, 2009.
- [22] Y. Lin, Y. Li and E. Zio, "A Reliability Assessment Framework for Systems With Degradation Dependency by Combining Binary Decision Diagrams and Monte Carlo Simulation", *IEEE Transactions on Systems, Man and Cybernetics: Systems*, Vol. 46, No. 11, page 1556 – 1564, 2016.
- [23] T. Hussain and R. Eschbach, "Automated fault tree generation and risk-based testing of networked automation systems", *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, page 1 – 8, 2010
- [24] C. Siontorou, F. Batzias and V. Tsakiri, "A Knowledge-Based Approach to Online Fault Diagnosis of FET Biosensors", *IEEE Transactions on Instrumentation and Measurement*, Vol. 59, Issue 9, page 2345 – 2364, 2010
- [25] Y. Mo, "A Multiple-Valued Decision-Diagram-Based Approach to Solve Dynamic Fault Trees", *IEEE Transactions on Reliability*, Vol. 63, Issue 1, page 81 – 93, 2014
- [26] V. Marik and D. McFarlane, "Industrial adoption of agent-based technologies", *IEEE Intelligent Systems*, Vol. 20, Issue 1, page 27 – 35, 2005
- [27] W. Dai, V. Vyatkin, J. Christensen, V. Dubinin, "Bridging Service-Oriented Architecture and IEC 61499 for Flexibility and Dynamic Reconfigurability", *IEEE Transactions on Industrial Informatics*, Vol. 11, Issue 3, page 771 – 781, 2015
- [28] W. Dai, V. Dubinin, J. Christensen, V. Vyatkin, X. Guan "Towards Self-Manageable and Adaptive Industrial Cyber-Physical Systems with Knowledge-Driven Autonomic Service Management", *IEEE Transactions on Industrial Informatics*, Vol. 13, No. 2, page 725 - 736, 2017
- [29] A. Mader, "A classification of PLC models and applications.", *Discrete Event Systems, Springer US*, page 239-246, 2000.
- [30] M. Parashar and S. Hariri, "Autonomic Computing: An Overview", *Unconventional Programming Paradigms*, ISSN: 0302-9743, page 257 – 269, 2005

- [31] M. Kantardzic, "Decision Trees and Decision Rules", *Data Mining: Concepts, Models, Methods and Algorithms*, page 139 – 164, 2003
- [32] IEC 61131-3, Programmable controllers - Part 3: Programming languages, *International Standard*, Second Edition, 2003
- [33] "OPC Foundation and MTConnect Institute Announce a Memorandum of Understanding", *OPC Foundation*, 2010
- [34] A. Zoitl and H. Prahof, "Guidelines and Patterns for Building Hierarchical Automation Solutions in the IEC 61499 Modelling Language", *IEEE Transactions on Industrial Informatics*, Vol. 9, Issue 4, page 2387 – 2396, 2013
- [35] IEC 61499, Function Blocks, *International Standard*, International Electrotechnical Commission, Geneva, Switzerland, Second Edition, 2012
- [36] PROFINET, May 1996. [Online]. Available: <http://www.profinet.com>
- [37] EtherCAT, 2003. [Online]. Available: <http://www.ethercat.org>
- [38] Ethernet/IP, Jun. 2006. [Online]. Available: <http://www.ethernet-ip.org>
- [39] Beckhoff Automation GmbH, [Online]. Available: <http://www.beckhoff.com>
- [40] Ubuntu MAAS, [Online]. Available: <https://maas.ubuntu.com>
- [41] OpenStack – Open source software for building private and public clouds, [Online]. Available: <http://www.openstack.org>

Wenbin Dai (GM' 09, M' 13, SM' 16) is an Associate Professor at Shanghai Jiao Tong University, China. He received a Bachelor of Engineering (with honours) degree in Computer Systems Engineering from the University of Auckland, New Zealand in 2006. He completed PhD in Electrical and Electronic Engineering at the Department of Electrical and Computer Engineering, The University of Auckland, New Zealand in 2012. He was a Postdoc Fellow at Lulea University of Technology, Sweden from 2013 to 2014. He was also a software engineer from Glidepath Limited – a New Zealand based airport baggage handling system provider from 2007 to 2013. His research interests are IEC 61131-3 PLC, IEC 61499 function blocks, industrial cyber-physical systems, semantic web technologies in industrial automation and industrial software agents.

Laurynas Riliskis is the founder and CEO of harmony.ai developing software for automation of data security and privacy. His work is based on his past three years of research at Stanford University in collaboration with MIT, Berkeley UC, and UPenn, as well as leading corporations including Intel, VMware, and Panasonic. His research has been published more than 40 times in most scientific conferences such as MobiSys, SenSys, ACM SIGSAC, USENIX Security Symposium, USENIX NSDI. He has eight prestigious awards for scientific excellence. He is an expert advisor in Critical Infrastructure Protection for European Commission and has been United Nation ITU mission. He received his Ph.D. in Computer Science from Lulea University of technology in 2014.

Peng Wang received Bachelor degree in Aerospace Engineering and Master degree in Electrical Engineering from Harbin Institute of Technology, China, in 2011 and 2013, respectively. He is currently working towards the Ph.D. degree in Control theory and engineering at Shenyang Institute of Automation, Chinese Academy of Sciences. His research interests are in the area of reconfigurable manufacturing system with special focus on model-driven engineering of the reconfigurable software.

Valeriy Vyatkin (M' 03, SM' 04) received Ph.D. degree in computer science from the State University of Radio Engineering, Taganrog, Russia, in 1992. He is on joint appointment as Chaired Professor of Dependable Computation and Communication Systems, Lulea University of Technology, Lulea, Sweden, and Professor of Information and Computer Engineering in Automation at Aalto University, Helsinki, Finland. Previously, he was a Visiting Scholar at Cambridge University, U.K., and had permanent academic appointments with the University of Auckland, Auckland, New Zealand; Martin Luther University of Halle-Wittenberg, Halle, Germany, as well as in Japan and Russia. His research interests include dependable distributed automation and industrial informatics; software engineering for industrial automation systems; and distributed architectures and multi-agent systems applied in various industry sectors, including smart grid, material handling, building management systems, and reconfigurable manufacturing. Dr. Vyatkin was awarded the Andrew P. Sage Award for the best IEEE Transactions paper in 2012.

Xinping Guan (M'02, SM' 04) received the B.S. degree in mathematics from Harbin Normal University, Harbin, China, in 1986, and the M.S. degree in applied mathematics and the Ph.D. degree in electrical engineering, both from the Harbin Institute of Technology, Harbin, in 1991 and 1999, respectively. He is currently a Professor with the Department of Automation, Shanghai Jiao Tong University, Shanghai, China. He is a Cheung Kong Scholars Programme Special Appointment Professor. His current research interests include functional differential and difference equations, robust control and intelligent control for time-delay systems, chaos control and synchronization, and congestion control of networks. He has (co)authored over 200 papers in mathematical and technical journals, and conferences.