IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 13, 2019, accepted August 24, 2019, date of publication August 30, 2019, date of current version September 17, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2938565

Semantic Integration of Plug-and-Play Software Components for Industrial Edges Based on Microservices

WENBIN DAI[1], (Senior Member, IEEE), PENG WANG[2], WEIQI SUN[1], XIAN WU[1], HUALIANG ZHANG[2], VALERIY VYATKIN[3,4], (Senior Member, IEEE), AND GENKE YANG[1]

[1]Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China
[2]Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China
[3]Department of Electrical Engineering and Automation, Aalto University, FI-00100 Espoo, Finland
[4]Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, 971 87 Luleå, Sweden

Corresponding author: Wenbin Dai (w.dai@ieee.org)

This work was supported in part by the National Research and Development Key Program of China under Project 2017YFA0700603, and in part by the Natural Science Foundation of China under Project 61973216.

ABSTRACT The industrial cyber-physical system enables collaboration between distributed nodes across industrial clouds and edge devices. Flexibility and interoperability could be enhanced significantly by introducing the service-oriented architecture to industrial edge devices. From the industrial edge computing perspective, software components shall be dynamically composed across heterogeneous edge devices to perform various functionalities. In this paper, a knowledge-driven Microservice-based architecture to enable plug-and-play software components is proposed for industrial edges. These software components can be dynamically configured based on the orchestration of microservices with the support of the knowledge base and the reasoning process. These semantically enhanced plug-and-play microservices could provide rapid online reconfiguration without any programming efforts. The use of the plug-and-play software components is demonstrated by an assembly line example.

INDEX TERMS Service-oriented architecture, industrial automation, IEC 61499 function blocks, plug and play, microservices, interoperability, REST API, SQWRL, OWL.

I. INTRODUCTION

The Industrial Cyber-Physical System (iCPS) refers to a system composes a number of network-connected smart devices that collaborate with each other subject to changes raised by customization requirements [1]. Various distributed interconnected devices form resilient and flexible production systems, on the other hand, also add to their complexity. One big issue is the interoperability between legacy devices and systems. Service-oriented architecture (SOA) is considered as the key to enabling adaptive, flexible and self-manageable systems. In the past few years, SOA has been piloted as a novel system engineering concept in several industrial automation areas, particularly in large projects, such as IMC-AESOP [2] and Arrowhead [3].

The associate editor coordinating the review of this article and approving it for publication was Zhangbing Zhou.

As shown in Fig. 1, the information and communication side of existing industrial automation control systems are traditionally depicted as a ''Pyramid'' following the ISA-95 standard [4]. In the ISA-95 pyramid, 5 layers are defined from top to bottom including enterprise resource planning (ERP), manufacturing execution systems (MES), supervisory control and data acquisition (SCADA), the controller (PLC) and field level (sensors and actuators).

The information exchange can only be achieved between two adjacent layers in the ISA-95 setup. For example, a controller can only communicate with SCADA and read/write values from/to sensors and actuators. If a change request is raised from MES, the message needs to be transmitted via SCADA to PLC. This mechanism limits the efficiency of information exchange especially when the frequency of data sampling is high. Within iCPS, the 5 layers are divided into two groups: the top two layers are moved to industrial clouds due to low real-time requirements; the remaining layers retain

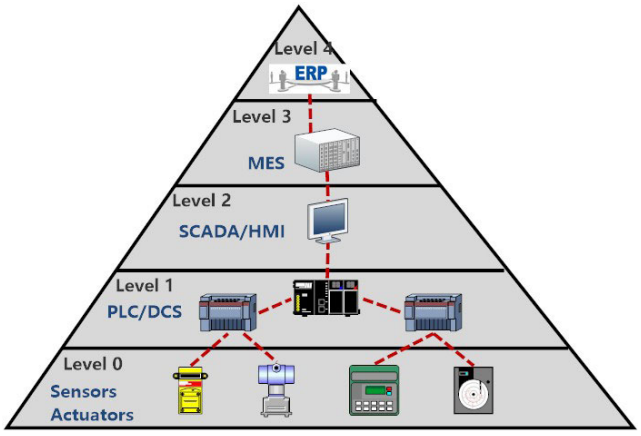

FIGURE 1. ISA-95 architecture for industrial automation systems.

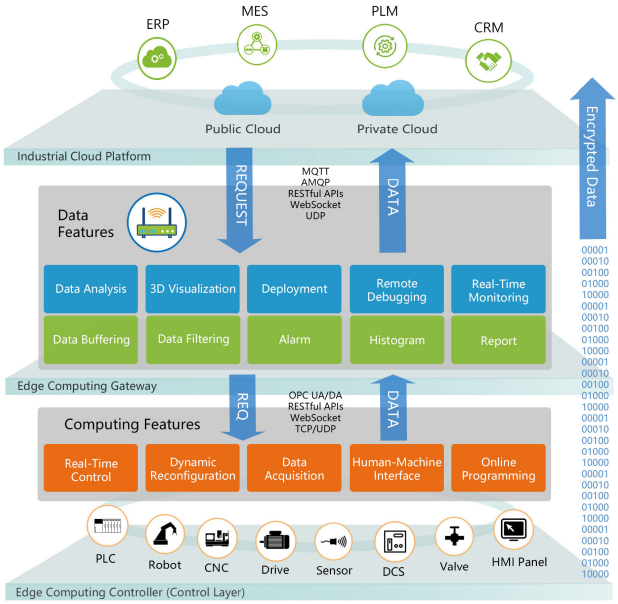

FIGURE 2. Service-oriented architecture enabled industrial cloud and edge computing systems.

as industrial edges for handling high real-time constraints. By adopting the SOA, information between industrial clouds and edge devices could be exchanged easily via flexible interfaces without any restrictions as shown in Fig. 2.

In the previous work [5], SOA is introduced for programmable logic controllers (PLC) to enable dynamic reconfiguration of distributed automation systems with minimum disruption to the real-time control processes. Each software component, deployed to the controller, is published as a service that can be accessed individually. However, the previous work relies on Web Services protocol stacks that are relatively heavy for edge devices with limited computing and storage resources. It was not capable of providing rapid self-configuration according to constantly changing requirements. In this paper, the Microservices-based plug-and-play software components are enhanced with knowledge bases to achieve dynamic reconfiguration for industrial edge devices.

This paper is organized as follows: In section II, state-of-the-art works related to SOA and plug-and-play software services are revised. In section III, the knowledge-driven Microservice-based plug-and-play software components for industrial edge computing are defined. The Microservices design pattern is applied for industrial edge devices in section IV. In section V, the automatic service orchestration process with knowledge reasoning support is discussed. Finally, a case study of the assembly line is presented to demonstrate the plug-and-play of software components and results are discussed in section VII.

II. RELATED WORKS

Cyber-physical systems (CPS), as defined by Lee [6], integrate computation and physical processes. CPS is considered as an enabling technology for Industry 4.0 [7], which gave rise to a new research domain of industrial CPS [8].

A. SERVICE-ORIENTED ARCHITECTURE AND MICROSERVICES RELATED PLUG-AND-PLAY

A service-oriented architecture is tightly connected with the future factory and CPS research [9]. The SOA is introduced to industrial automation systems by Jammes and Smit [10]. The intelligence of computing and communications is driven by SOA from enterprise level all the way down to device level as network-connected devices with Web Services standards implemented on-board.

A flexible and interoperable architecture is proposed by Girbea *et al.* [11] for designing distributed industrial automation systems. OPC UA servers along with multiple levels of services are proposed to extract information from controllers. An adapter is developed for integration of legacy devices. Large concurrent connections can be managed by the OPC UA server by adopting this adapter software. Real-time constraints of the manufacturing process are proved by using service-based software solution in that paper. Jirkovsky *et al.* [12] utilized Semantic Web technologies along with OPC UA to achieve horizontal and vertical data integration. The plug-and-play software components are based on semantic big data historian. With ontology, SPARQL and workflow, those components can be self-described and data can be easily gathered and integrated for analysis.

Yang et. al. proposed a plug and play peripherals for the Internet of Things devices [13]. The proposed solution provides support for automatic integration, discovery and remote access to third-party peripherals. The solution only requires minimal memory footprint and a series of event handlers are used to handle the event exchange between drivers, interconnected libraries and network protocol stacks for rapid response times.

Rufino et. al. proposed an orchestration of lightweight Microservices based on Docker [14]. The proposed architecture offers better modularity and scalability for Industrial IoT. The Microservices based virtualization enables simple distributed deployment across different devices and improves

robustness by self-healing. Li. et. al. from Siemens also applied the Microservice patterns for industrial edge software life cycle management [15]. Four Microservices patterns are proposed to cover four stages of industrial edge software including deployment, monitoring, adaptation, and testing. These patterns could improve load-balancing and QoS for industrial edge software. Dobaj et. al. also proposed a lightweight Microservice-based architecture for the Industrial IoT [16]. The Microservice design pattern provides better flexible deployment from industrial clouds and continuous integration through all levels including cloud, fog, and edge devices. From these experiments, the Microservices design pattern could bring flexibility for industrial edge devices.

B. WEB SERVICES PROTOCOLS

Cândido *et. al.* [18] proposed a service-oriented evolvable production system for device lifecycle tracing in industrial automation systems. The WS-Management protocol is implemented on the engineering tool as well as functions on PLCs to assist deployment process. Newly deployed management services in PLCs start to provide track and trace information and exchange with other devices or systems. Continuing from that, Cucinotta et. al. further enhanced the proposed web service-based functions with real-time support for industrial automation applications [19]. Web services enabled the client's ability to negotiate with other clients to eliminate interference by measuring QoS. The QoS is ensured from the enterprise level all the way down to the shop floor level for real-time applications.

The gap between sensor level and enterprise application level is bridged by adopting SOA as proposed by Kyusakov *et al.* [17]. By adopting the Simple Object Access Protocol (SOAP) and web services, these two layers are directly connected and intermediate gateways are eliminated. Sensor data can be directly sent using SOAP to legacy SCADA, MES and ERP systems with service adapters. Adding Web Services on sensor nodes had minimal effects of performance as proved by the authors.

Mathes *et al.* [20] enhanced legacy programmable logic controllers with flexibility and interoperability by inserting web services. A SOAP engine was developed for enabling SOA on legacy PLCs for manufacturing processes. The developed engine only requires a small memory footprint and low computational power.

Farrag *et al.* [21] provided a direct mapping from WSDL to OWL-S for web service discovery. Information extracted from WSDL files is automatically generated into OWL-S format that allows further to be registered and published online. Local service repository based on ontological knowledge base is created for storing OWL-S based service contracts.

From these results, the Web Services protocol stacks are commonly adopted for improving flexibility for industrial automation systems. However, the WS-* features including centralized discovery, context-based identification, shared models, static code generation and explicit conversation are not suitable for lightweight Web applications.

C. SEMANTIC WEB TECHNOLOGIES AND KNOWLEDGE BASE

Plug-and-play software components are discussed in the computer science domain since decades ago [22], [23]. Schulte *et al.* [24] discussed the feasibility of introducing plug-and-play components in virtual factories. Technologies including service-oriented computing and Internet-of-Things are adopted in the business process management to assist establishing, managing and monitoring in a plug-and-play way.

Schulte *et al.* [24] described an engineering knowledge base framework for integrating software components from multidisciplinary to create a software-intensive system. Data models could be exchanged seamlessly while no change is made to existing engineering tools. The results were demonstrated on a software-intensive production automation system. The programming effect could be reduced by reusing existing software components.

Puttonen *et al.* [25] presented a semantic web service approach for managing production processes. Web service interfaces are introduced to devices and web ontology language for services (OWL-S) is used to compose services automatically. The semantic model of the production system is demonstrated to be automatically updated based on three web services for compositions of different processes.

Guinard *et al.* [26] integrated SOA with embedded devices to introduce intelligence in Internet-of-Things. A suitable system architecture is proposed for large numbers of networked, resource limit devices to query, select and invoking web services dynamically. Device Profiles for Web Services (DPWS) and RESTful services are recommended for integration between physical devices and enterprise information systems.

Overall, existing works provide the fundamental architecture for vertical integration through multiple layers. The aim of this work is to enable lightweight scalable, flexible and interoperable components for industrial edge software. By integrating Semantic Web Technologies including OWL and SQWRL with the Microservices design pattern, plug-and-play software components for industrial edge software can be achieved.

III. PLUG-AND-PLAY SOFTWARE COMPONENTS BASED ON MICROSERVICES

Existing industrial applications like control and HMI are pre-programmed and commissioned by engineers prior to the production stage. These applications are normally composed of a set of functions or function blocks that must be downloaded onto devices they assigned to. Control applications are usually designed following the bottom-up principle that is: reusable functions and function blocks with pre-defined interfaces are created prior to construct system functionalities. Online editing of control programs is allowed only for minor code changes while program structures remain unchanged.

The plug-and-play software component provides a new top-down design process for creating distributed applications

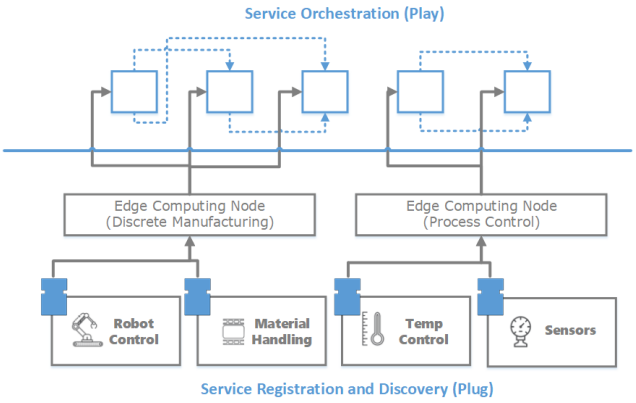

FIGURE 3. Design process for plug-and-play service-based software components.

for industrial edge as shown in Fig. 3. In the iCPS, each field-level device such as sensors and actuators contain atomic functions registered as software services on controllers. These atomic services can be composed into composite services to provide a hierarchical structure to hide complexities. Atomic services together with composite services can be organized as different service flows automatically to meet massive customization requirements.

The Microservices design pattern perfectly suits for these scenarios. The Microservices design pattern is designed for scales software components for flexibility. It can be described as a three-dimensional model: The first dimension is the duplication that scale components by cloning services; the second dimension is functional decomposition that scale component by splitting services by various functionalities; The last dimension is the data partitioning that scale components by splitting services with similar functionalities but separated data.

By adopting Microservices design pattern, the following benefits could be introduced to industrial edge applications: Firstly, monolithic industrial edge applications can be decomposed into a set of services and each service can be developed and deployed independently. Secondly, each service can be replaced with new technology or another programming language without affecting other services. Finally, these services can be scaled independently that can be assigned to hardware best suit these services.

To enable Microservices-based industrial edge applications, the Microservice design pattern must be applied to the distributed modeling language by defining a set of mapping rules. Service discovery, service orchestration and deployment patterns based on Microservices will be explained in the following sections.

IV. PUBLISH AND DISCOVER MICROSERVICES FOR INDUSTRIAL EDGE

To achieve plug-and-play, the first step is to publish services that can be discovered by other distributed nodes. As defined in the previous work [5], the IEC 61499 standard [27] is used for modelling iCPS by encapsulating IEC 61131-3 programming languages (such as ST and LD) [28] and other high-level programming languages (such C/C++, Java and JavaScript) into event-triggered function blocks (FB) as internal algorithms [30]. These FBs with a common interface that can be composed into function block networks to perform various functionalities. Following that, each IEC 61499 function block instance is wrapped as a software service. A runtime environment was also developed for creating, modifying, deleting and invoking FB services. Continuing from there, these service-based IEC 61499 FBs will be published and discovered through the Microservices design pattern.

A. MICROSERVICE ARCHITECTURE FOR IEC 61499

As shown in Fig. 4, a typical Microservice architecture contains an API gateway and a set of services that can be accessed via REST APIs.

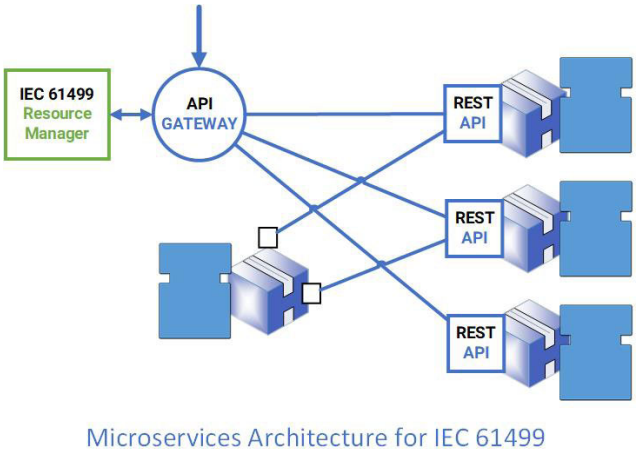

FIGURE 4. Microservices Architecture mapping with IEC 61499 FB.

Rule 1: *Each IEC 61499 Resource Manager is mapped to an API gateway.*

The IEC 61499 resource manager is responsible for manage function block networks by creating and deleting function block types, instances, and connection between function blocks on a device. It can also fetch and write data variables from function blocks as well as control operation of applications by using *START*, *STOP* and *KILL* command. In the Microservice architecture, the entrance of the application is managed by the API gateway. The API gateway contains all services running on this device and interfaces to these services. The IEC 61499 resource manager shall be acted as an API gateway in the Microservice architecture.

The IEC 61499 management commands are redefined by using one of the four HTTP methods adopted in REST APIs: *GET, PUT, POST* and *DELETE*. Firstly, the HTTP *GET* method is normally used to retrieve the data element or collection. The *QUERY* and *READ* action of the IEC 61499 management commands can be mapped to the *GET* method as shown in Table 1 below. Both actions retrieve data including

TABLE 1. REST API mapping between HTTP methods and IEC 61499 actions of the management commands.

MGT Action	GET	PUT	POST	DELETE
CREATE			✔	
DELETE				✔
START		✔		
STOP		✔		
KILL		✔		
QUERY	✔			
READ	✔			
WRITE		✔		
RESET		✔		

status and variable data from the IEC 61499 resource manager. Secondly, the HTTP *PUT* method is used to update the existing data element or collection with new values. In the IEC 61499 version, device operation control commands including *START, STOP, KILL* and *RESET* are set to use the HTTP *PUT* request as these actions only updates controller modes. Also, the *WRITE* action is also using HTTP *PUT* request to update variable values. Next, the HTTP *POST* method creates a new entry of an element or a collection. It could be used to create new FB types and instances as well as connections of events and data. Finally, the HTTP *DELETE* method is used to remove an element from the target node. In this case, it can be used to delete function block types, instances, and connections from function block networks.

Rule 2: *Each Function Block Instance is mapped to a backend service.*

Each function block instance is created as an independent service. The IEC 61499 resource manager handles requests by simply routing them to the appropriate backend services, in this case, trigger function block instances and aggregating the results. For example, a management command that is defined in the XML format to create a new FB instance could be written as an HTTP POST request:

```
<Request ID="1" Action="CREATE">
  <FB Name="FBI1" Type="ev3_tacho_motor" />
</Request>
```

```
POST /FB
Body Message: Name=FBI1&Type=ev3_tacho_motor
```

Rule 3: *The contents of the management commands are converted into the HTTP URI and parameters if there is no child node. Otherwise, the original XML formats will be used as the contents.*

In the example above, since the target *FB* has no child node in the original XML format, it is converted to the URI */FB* in the HTTP *POST* request. In addition, all attributes of the original XML are transferred into the HTTP parameters in the HTTP *POST* body message. On the other hand, for creating a new FB type, the *FBType* node that with interfaces and algorithms attached in the XML will be embedded directly as the body message.

TABLE 2. Inter-process communication mapping for function block network.

Methods	One-to-One	One-To-Many
Synchronous	Adapter	N/A
Asynchronous	Event Connection	Publish/Subscribe

Next, the inter-process communication between distributed services must be defined for service invocation. There are two mechanisms could be applied here: the synchronous approach that requires a response from the service and the asynchronous approach that doesn't wait for a response immediately. Also, there are two interaction styles: one-to-one where each request is handled by exactly one service; one-to-many where multiple function block instances could be invoked by a single request.

Each IEC 61499 application can use single or a combination of these inter-process communication mappings as shown in Fig. 5 below.

In the synchronous approach, a FB sends a request to another downstream FB and waits for a response, for example, triggering an external timer and receive time up alarm. This shall be modeled as an adapter connection in the IEC 61499 implementation as indicated in Fig. 5 (a). The IEC 61499 adapter is designed as a group of bidirectional communication interface to hide complexities of massive connections. In this case, the request and response side are used as the *Plug* and the *Socket* on the two ends of the adapter connection where two adapter instances are mirrored. In the asynchronous version as shown in Fig. 5 (b), the one-to-one communication shall be modeled as separated event connections. Once the execution of the downstream FB is completed, another event output will be triggered to notify the requester FB. In the scenario of one-to-many asynchronous communication, a set of Publish/Subscribe SIFBs with identical IDs shall be used as indicated in Fig. 5 (c).

However, in the one-to-one synchronous approach, if the downstream FB is faulted, the upstream FB will wait for response cause the execution halted. To handling these partial failures, the execution should never be blocked indefinitely waiting for downstream FBs. Instead, a watchdog function as shown in Fig. 6 below must be used to monitor the real-time constraints. The state-machine contains three states *IDLE, WAIT* and *FAULT*. When a requester FB sends a message to the downstream FB, the state machine will move to the *WAIT* state and start monitoring the time elapsed. A threshold time is set for the download FB to return the response message. If the threshold time expires, it will jump into

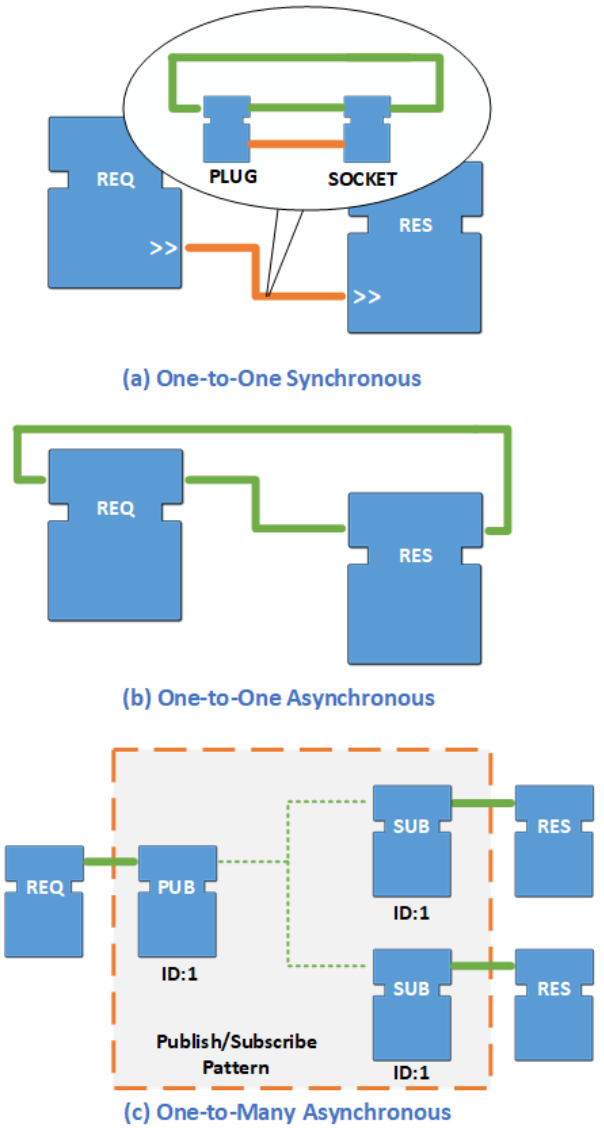

FIGURE 5. Inter-process communication pattern for microservice-based IEC 61499.

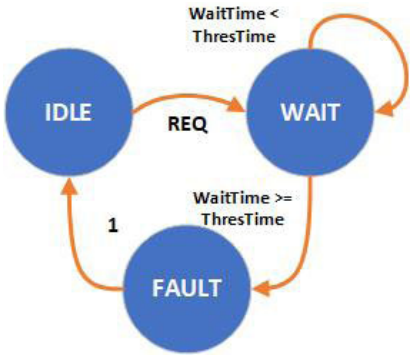

FIGURE 6. Watchdog function for handling partial failures.

the *FAULT* state and current waiting process will be forced to terminate. The execution will continue without further delay.

Finally, interfaces of FB instances must be exposed for data exchange purposes. When creating a FB instance, the variables of FB data inputs and outputs can be accessed via REST APIs.

Rule 4: *For each input, output and internal variable in any function block instance, a unique URI is registered that can be read or write via the HTTP GET and POST method.*

The URIs for accessing the interface of a FB instance is defined as:

http://<IPAddress>:<Port>/<FBI>/<Variable>

where *<IPAddress>* represents the current IP address of the resource; *<Port>* refers to the port number where REST services are running; *<FBI>* indicates the name of a particular function block instance at the late binding stage; *<Variable>* refers to input/output variable name of this function block instance.

By applying all the rules described in this section, the FB network could be managed as well as data integration can be achieved via simple REST APIs.

B. PUBLISH AND SERVICE DISCOVERY

Once Microservices are created, the next step is to register these services on the network and allow discovery from other connected devices. In the Microservices pattern, service providers first register their contracts with a service repository. To invoke services from providers, service consumers query repository for fetching contracts. Service consumers invoke services from providers through addresses and interfaces as defined in the REST APIs.

IEC 61499 function blocks are managed by interpreting management commands that are defined in the IEC 61499 compliance profile for feasibility demonstrations as illustrated by [29]. The resource manager is responsible for handling external requests and modify function block networks according to received commands. In the industrial edge computing, IEC 61499 resource managers are turned into service repositories for register and publish services.

The IEC 61499 management commands that are defined in the XML format can be directly applied as the application-level protocol for service management. As defined in the IEC 61499 compliance profile [29], management commands support a set of actions including *create, delete, read, write, start, stop, reset, kill* and *query*. The target object could be selected from function block instance, connection, function block type, adapter type, data type, and parameters. Every request received by the device manager contains an ID. The device manager shall process the request and compose a response with the corresponding ID and result with reasons (such as not ready, unsupported command, unsupported type, no such object, etc.).

The microservice discovery is designed as a peer-to-peer design pattern. Each resource manager is also running as a service registry with REST APIs. The resource manager is responsible for determining available service instances on the network and performing load balancing requests across

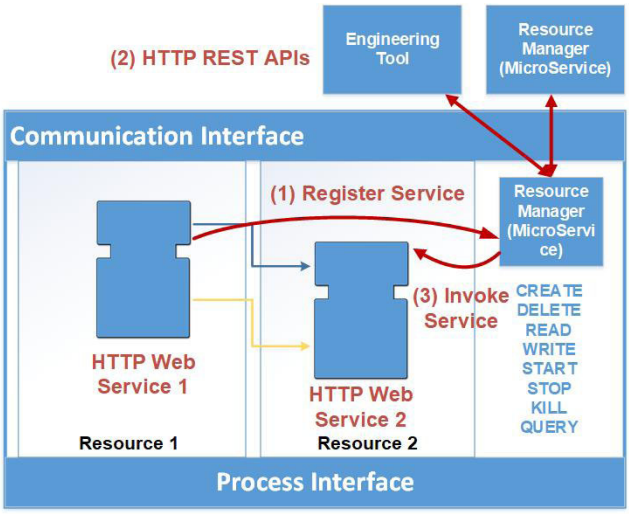

FIGURE 7. Bridging IEC 61499 management model with microservice repository.

```
<!ELEMENT Request
(FB|Connection|FBType|AdapterType|DataType|Parameter|Res)>
<!ELEMENT Res (FB*)>
<!ATTLIST Res
  Name CDATA #REQUIRED
  Type CDATA #REQUIRED>
```

LISTING 1. Extensions to IEC 61499 compliance profile definitions.

these resources. When creating a FB instance, the management command is posted to the resource manager for publishing a new microservice (Step 1). The resource manager will create add all endpoints (events) of this new service to the service repository. Other device managers shall be able to query this new service from where it has been registered. However, there is an issue: managing services and query list of available services are not covered in existing management commands. To manage services in an IEC 61499 resource, IEC 61499 compliance profile needs to be extended.

A new element is introduced to IEC 61499 compliance profile definitions to enable service query:

Res: refer to an individual resource on a device. It also has two attributes: *Name* of this resource and *Type* of this resource. In addition, FB instances created on this resource will also be attached during query.

The following management command is used to query all available services on a particular resource (Step 2 in Fig. 7).

<Request ID = "2" Action = "QUERY">
  <Res Name = "Dev1.Res1" Type = "EMDRES" />
</Request>

The resource manager will return all registered services as:

<Response ID = "2" Reason = "RDY">
  <FB Name = "Dev1.Res1.App1.FB1" Type = "ConvControl" />
  .... //All other FB Instances
</Request>

Once the target FB service is located, the following command is used to fetch the service endpoint (Step 3 in Fig. 7):

<Request ID = "3" Action = "QUERY">
  <FB Name = "Dev1.Res1.App1.FB1" Type = "ConvControl" />
</Request>

The resource manager will return with the registered endpoint for this FB instance:

<Response ID = "3" Reason = "RDY">
  <FB Name = "http://192.168.1.3:8080/ Dev1.Res1.App1.FB1" />
</Response>

Alternatively, if the FB service is not found, the response reason will be set to INVALID_OBJ with no content internally. Finally, the resource manager of the service consumer will interpret the service contract and invoke service directly.

V. AUTOMATIC SERVICE ORCHESTRATION BASED ON KNOWLEDGE BASE AND REASONING PROCESS

The final step is to organize individual microservices into a logical order to perform control applications. In the SOA paradigm, individual services are linked by service orchestration process that is usually described by a modeling language namely business process execution language (BPEL) [33]. Using BPEL as the description language for organizing distributed control applications is also a feasible option. On the other hand, the IEC 61499 standard is also used as the modeling language for distributed automation systems [30]. More importantly, it provides a system-level software model that can be directly deployed and executed on networked controllers. This brings huge benefits from the engineering perspective by cost-saving and rapid prototyping.

To automatic construct function blocks in the IEC 61499 architecture, the industrial software agent is adopted. In the previous work, a knowledge-driven service-oriented industrial software agent is proposed for managing SQWRL queries [34]. The Monitor-Analyse-Plan-Execute-Knowledge (MAPE-K) approach is used to form closed-loop with the control software. In this case, this industrial software agent is sitting on the top of all service repositories to monitoring requests from external conditions. When reconfiguration is required, the software agent will plan the necessary changes and send commands to corresponding service repositories to perform service orchestration.

The service orchestration process is as illustrated in Fig. 7:

1) Check for all existing services and interfaces from service repositories.
2) Remove all bindings (event and data Connections) between microservices.
3) Select required services according to reasoning results
4) Create bindings for required service invoking (event and data connections)

The MAPE-K agent utilizes the knowledge base and the reasoning process to assist service orchestration. The knowledge base is designed based on ontologies and the semantic query-enhanced web rule language (SQWRL) is used

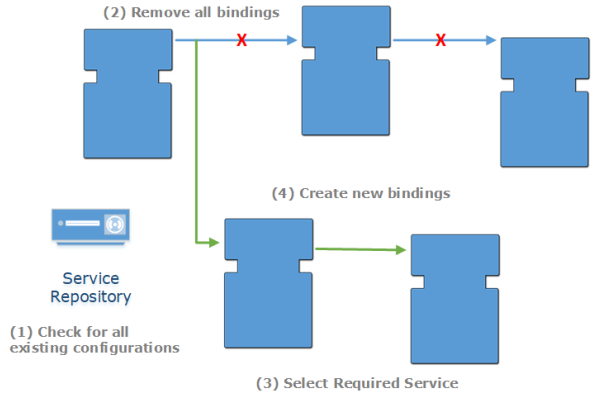

FIGURE 8. Service orchestration process for distributed control applications using IEC 61499 function blocks.

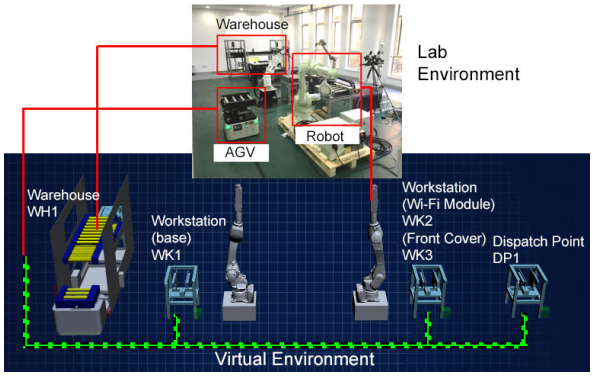

FIGURE 9. Case study example of the assembly line with AGV: layout and architecture.

to query the knowledge [31]. The SQWRL is an extended version of the semantic web rule language (SWRL) with query abilities [32]. The ontological knowledge base contains orchestration rules defined in SWRL.

In the monitor and analyze the process, information is collected and decision on any necessary change is made. In the plan and execute the process, the service orchestration is handled by the reasoning of semantic rules. An individual *Profile* is created for each IEC 61499 application that contains a FB network. The rule for searching a particular with a given type is shown as:

Profile (?p) ∧ serviceCategory (?p, ?sc)
∧ categoryName(?sc, ?scname)
∧ swrlb:equal (?scanme, ''APP_NAME'')
∧ serviceName (?p, ?name) -> sqwrl:select (?name)

The *serviceCategory* defines type name of an IEC 61499 application. The *swrlb:equal* is a SWRL built-in function for comparing two operands. When a profile is selected, the service flow could be built by connecting events and data variables sending an HTTP POST request to the resource manager via the REST API with the following message body:

<Request ID = ''4'' Action = ''CREATE''>
<Connection Source = ''FB1'' Destination = ''FB2'' />
</Request>

Once the orchestration process is completed, the system is dynamically reconfigured so that newly plugged software components are now in action. In the next section, the detailed knowledge-driven service orchestration will be illustrated using a material handling example.

VI. CASE STUDY OF PLUG-AND-PLAY MICROSERVICES

The knowledge-driven plug-and-play software-defined system will experiment through the dynamic routing for AGV in the assembly line.

As shown in Fig. 9, the assembly line for power sockets has two workstations and each work station equips with an industrial robot for installing sockets partially. The power sockets can be customized with one or two ports and with/without Wi-Fi feature. The first workstation is used to assemble socket bases. The second one is designed as a shared workstation for installing the Wi-Fi module as well as the front cover for the sockets. An AGV is used to transport parts between the warehouse and the workstations.

To assemble a two-ports power socket with Wi-Fi module, the AGV needs to pick up parts from the warehouse and then travel through all three workstations in sequence to complete the process. Before starting the assembly process, the routing plan for the AGV must be dynamically reconfigured. The knowledge base is queried by the agent to fetch the proper profile as shown in the previous section. As shown in Fig. 9, the selected profile contains a route from the warehouse WH1 to the dispatch point DP1 via the two workstations WK1(base), WK2/3 (Wi-Fi module and Front Cover). The orchestrated function block network is given in Fig. 10. Functions block instances are invoked in sequence according to motion action steps by emitting event outputs to downstream FB instances.

The orchestration process starts by loading a detailed routine for the AGV. The routine can be further divided into several steps. For each step, a set of management commands are sent to the target controller via REST APIs. For example, after the parts are loaded at the warehouse, the AGV will start moving from the warehouse to the workstation A by moving 1m forward. Several changes are required to perform that movement. First, two motors (*FB28* and *FB39*) and a position sensor (*FB11*) are linked to the routing control module (*FB59*). For example, the following management commands are sent to the resource manager using HTTP PUT methods by creating connections between the routine control *FB59* and the motor control *FB28* as well as set movement distance to 1m with 30% of the motor full speed:

<Request ID = ''1'' Action = ''CREATE''>
<Connection
From = ''Dev1.Res1.App1.FB59.RUN_TO_REL_POS''
To = ''Dev1.Res1.App1.FB28.RUN_POS'' />
<Connection From = ''Dev1.Res1.App1.FB59.position_out'' To = ''Dev1.Res1.App1.FB28.position'' />
<Connection From = ''Dev1.Res1.App1.FB59.speed_out1'' To = ''Dev1.Res1.App1.FB28.speed'' />
<Parameter Name = ''Dev1.Res1.App1.FB59.position_in'' Value = ''100'' />

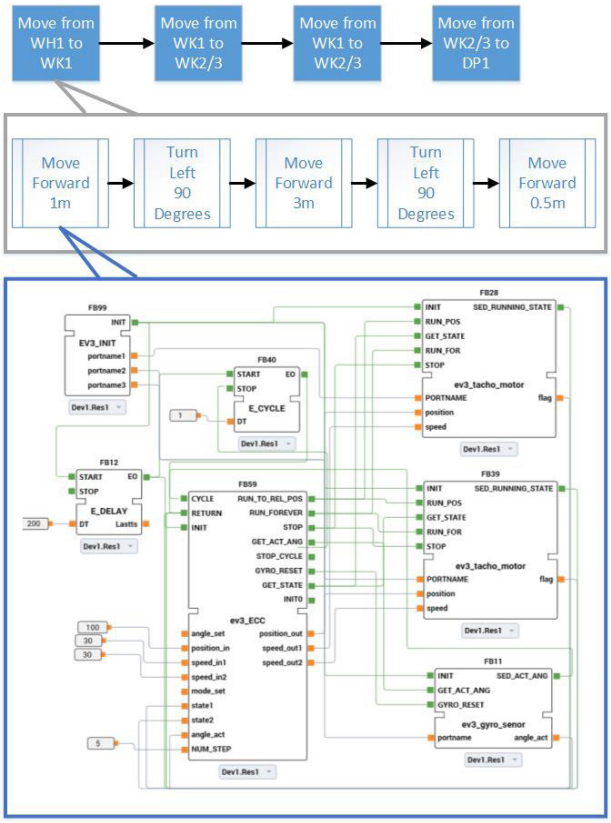

FIGURE 10. An IEC 61499 application example of AGV routing.

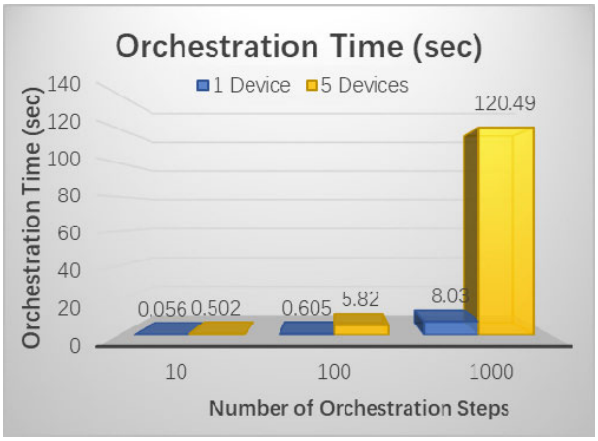

FIGURE 11. Plug-and-play processing time comparison.

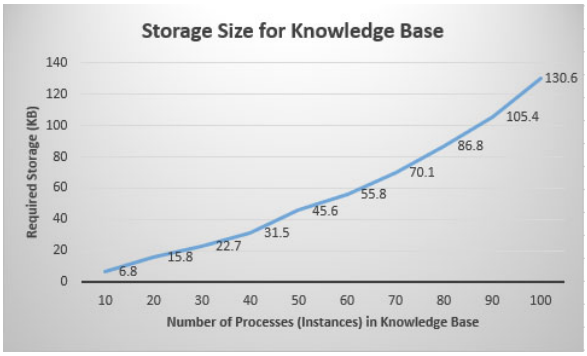

FIGURE 12. Storage analysis for knowledge base support on controllers.

<Parameter Name = "Dev1.Res1.App1.FB59.speed_in1" Value = "30" />
</Request>

Similar commands could be used to create connections between the routine control *FB59* and the other motor control *FB39* and the sensor *FB11*. Once the FB network is orchestrated according to the routing plan fetched from the knowledge base, the start running command is sent to the resource manager to perform this movement. When the AGV is reached at the endpoint of the 1m, these connections will be removed for the next step orchestration. By repeating these steps, the AGV will reach its final destination at the dispatch point.

VII. MEASUREMENTS AND DISCUSSIONS

Two measurements are taken for the plug-and-play process and a knowledge base is created manually for these tests. The hardware used is a Core-i7 2.6Ghz quad-core CPUs with 16GB RAM. The first one is the processing time of service orchestration. The results in Fig 11 demonstrate that the orchestration time for 1 device is significantly faster than that for 5 devices. The orchestration time on a single device can be of the millisecond level, however, with multiple devices, it takes seconds or even minutes.

The reason is that the orchestration time consists of two parts: query time and reconfiguration time. For single devices, a query and search time is minimal, the entire processing time could be limited to milliseconds. With multiple devices, the number of management commands is continuously increasing. When it reaches a limit, the orchestration time will increase exponentially due to the previous orchestration task is already completed and messages are pushed into buffers. For manufacturing lines or process control, the orchestration process can be completed within real-time requirements. However, it is not yet suitable for high real-time constrained applications such as motion control systems.

Secondly, the storage memory required for knowledge support is measured. Embedded controllers usually have limited memory for storage. With increasing of elements (function block instances and connections), the storage size required on controllers is gradually increasing as one function block may create several connections. With 100 combined instances, the storage size required is still at hundreds of KB which is acceptable for modern industrial controllers.

The plug-and-play feature brings several benefits for controllers. Although all industrial controllers claim they are compatible with the IEC 61131-3 standard, there are still lots of platform-dependent implementations. As a result, control software implemented in one platform cannot be ported to other platforms. By adopting the proposed knowledge-driven microservice-based plug-and-play software components, massive re-development time could be saved. When the target platform is switched from one to another, the difficulty for the integration of distributed automation systems is reduced significantly.

Also, plug-and-play software services enable dynamic reconfigurations with no programming operations that may affect manufacturing operations. In legacy manufacturing lines, entire control systems must be stopped and re-programmed to add extra new functionalities. With plug-and-play, new features are easily introduced to existing control systems during normal operations. This could also save a large portion of site commissioning time by minimizing system downtime. Finally, the IEC 61499 management command and SQWRL based service composition process can be extended easily and future-proof.

There are also some downsides of the proposed method that need to be addressed. Since the HTTP REST protocol requires long interpretation time [35], management commands could be encoded as binary XML for speed up sending and receiving HTTP request messages [36]. To the maximum, the plug-and-play feature, automatic code generation is needed [37], [38].

Last but not least, plug-and-play software components in IEC 61131-3 PLCs are also achievable by using the proposed approach. The PLCOpen XML format can be used to store and manage IEC 61131-3 function blocks [39]. However, there are several issues that need to be solved, for example, choose a proper system modeling language to represent service orchestration process as top-level entity of the IEC 61131-3 software model is limited to a single device; how to perform code generation from created system model is also a challenge as the system model cannot be directly executed.

VIII. CONCLUSION

Industrial cyber-physical systems bridge various devices and systems by enabling loosely coupled collaborative automation systems. To achieve coordination between distributed industrial edge devices, plug-and-play software components based on microservice architecture are adopted for industrial edge computing. Interfaces of software components are defined as microservices with REST APIs. Dynamic composition of software components is achieved by service orchestration using reasoning rules and knowledge base between multiple industrial edge controllers.

This work presents a foundation for implementing automatic code generation based on plug-and-play components with the support of knowledge bases. Also, integration between microservice-enabled controllers and legacy IEC 61131-3 PLCs will be investigated. Finally, semantic enrichment process will be introduced to iCPS by interpreting flow diagrams as knowledge. Together with industrial software agents, knowledge-driven plug-and-play software services could provide autonomous intelligent control for iCPS.

REFERENCES

[1] J. Schlick, "Cyber-physical systems in factory automation—Towards the industrial revolution," in *Proc. 9th IEEE Int. Workshop Factory Commun. Syst. (WFCS)*, May 2012, p. 55.

[2] A. W. Colombo, T. Bangemann, S. Karnouskos, J. Delsing, P. Stluka, R. Harrison, F. Jammes, and J. Lastra, *Industrial Cloud-Based Cyber-Physical Systems: The IMC-AESOP Approach*. New York, NY, USA: Springer-Verlag, 2014.

[3] P. Varga, F. Blomstedt, L. L. Ferreira, J. Eliasson, M. Johansson, J. Delsing, and I. M. de Soria, "Making system of systems interoperable—The core components of the arrowhead framework," *J. Netw. Comput. Appl.*, vol. 81, pp. 85–95, Mar. 2016.

[4] B. Scholten, *The Road to Integration: A Guide to Applying the ISA-95 Standard in Manufacturing*. Paris, France: ISA, 2007.

[5] W. Dai, V. Vyatkin, J. H. Christensen, and V. N. Dubinin, "Bridging service-oriented architecture and IEC 61499 for flexibility and interoperability," *IEEE Trans. Ind. Informat.*, vol. 11, no. 3, pp. 771–781, Jun. 2015.

[6] E. A. Lee, "Cyber physical systems: Design challenges," in *Proc. 11th IEEE Int. Symp. Object Compon.-Oriented Real-Time Distrib. Comput. (ISORC)*, May 2008, pp. 363–369. doi: 10.1109/ISORC.2008.25.

[7] J. Schlick, "Cyber-physical systems in factory automation—Towards the 4th industrial revolution," in *Proc. 9th IEEE Int. Workshop Factory Commun. Syst.*, May 2012, p. 55.

[8] A. Fisher, C. Jacobson, E. Lee, R. Murray, A. Sangiovanni-Vincentelli, and E. Scholte, "Industrial cyber-physical systems—iCyPhy," in *Complex Systems Design & Management*. Cham, Switzerland: Springer, 2014, pp. 21–37.

[9] A. Colombo and S. Karnouskos, "Towards the factory of the future: A service-oriented cross-layer infrastructure," in *ICT Shaping the World: A Scientific View*, vol. 65. Hoboken, NJ, USA: Wiley, 2009.

[10] F. Jammes and H. Smit, "Service-oriented paradigms in industrial automation," *IEEE Trans. Ind. Informat.*, vol. 1, no. 1, pp. 62–70, Feb. 2005.

[11] A. Girbea, C. Suciu, S. Nechifor, and F. Sisak, "Design and implementation of a service-oriented architecture for the optimization of industrial applications," *IEEE Trans. Ind. Informat.*, vol. 10, no. 1, pp. 185–196, Feb. 2014.

[12] V. Jirkovský, M. Obitko, P. Kadera, and V. Mařík, "Toward plug&play cyber-physical system components," *IEEE Trans. Ind. Informat.*, vol. 14, no. 6, pp. 2803–2811, Jun. 2018.

[13] F. Yang, N. Matthys, R. Baciller, S. Michiels, W. Joosen, and D. Hughes, "μpnp: Plug and play peripherals for the Internet of Things," in *Proc. 10th ACM Eur. Conf. Comput. Syst.*, 2015, Art. no. 25.

[14] J. Rufino, M. Alam, J. Ferreira, A. Rehman, and K. F. Tsang, "Orchestration of containerized microservices for IIoT using docker," in *Proc. IEEE Int. Conf. Ind. Technol.*, Mar. 2017, pp. 1532–1536.

[15] F. Li, J. Fröhlich, D. Schall, M. Lachenmayr, C. Stückjürgen, S. Meixner, and F. Buschmann, "Microservice patterns for the life cycle of industrial edge software," in *Proc. 23rd ACM Eur. Conf. Pattern Lang. Programs*, Jul. 2018, Art. no. 4.

[16] J. Dobaj, J. Iber, M. Krisper, and C. Kreiner, "A microservice architecture for the industrial Internet-of-Things," in *Proc. 23rd ACM Eur. Conf. Pattern Lang. Programs*, Jul. 2018, Art. no. 11.

[17] R. Kyusakov, J. Eliasson, J. Delsing, J. van Deventer, and J. Gustafsson, "Integration of wireless sensor and actuator nodes with IT infrastructure using service-oriented architecture," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 43–51, Feb. 2013.

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

[19] T. Cucinotta, A. Mancina, G. F. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, "A real-time service-oriented architecture for industrial automation," *IEEE Trans. Ind. Informat.*, vol. 5, no. 3, pp. 267–277, Aug. 2009.

[20] M. Mathes, C. Stoidner, S. Heinzl, and B. Freisleben, "SOAP4PLC: Web services for programmable logic controllers," in *Proc. 17th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process.*, Feb. 2009, pp. 210–219.

[21] T. A. Farrag, A. I. Saleh, and H. A. Ali, "Toward SWSs discovery: Mapping from WSDL to OWL-S based on ontology search and standardization engine," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 5, pp. 1135–1147, May 2013.

[22] F. Bronsard, D. Bryan, W. Kozaczynski, E. S. Liongosari, J. Q. Ning, Á. Ólafsson, and J. W. Wetterstrand, "Toward software plug-and-play," in *Proc. Symp. Softw. Reusability*, Boston, MA, USA, May 1997, pp. 19–29.

[23] M. Mezini and K. Lieberherr, "Adaptive plug-and-play components for evolutionary software development," in *Proc. 13th ACM SIGPLAN Conf. Object-Oriented Program., Syst., Lang., Appl.*, vol. 24, Oct. 1998, pp. 97–116.

[24] S. Schulte, D. Schuller, R. Steinmetz, and S. Abels, "Plug-and-play virtual factories," *IEEE Internet Comput.*, vol. 16, no. 5, pp. 78–82, Sep./Oct. 2012.

[25] J. Puttonen, A. Lobov, and J. L. M. Lastra, "Semantics-based composition of factory automation processes encapsulated by Web services," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2349–2359, Nov. 2013.

[26] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-based Internet of Things: Discovery, query, selection, and on-demand provisioning of Web services," *IEEE Trans. Service Oriented Comput.*, vol. 3, no. 3, pp. 223–235, Jul./Sep. 2016.
[27] *Function Blocks, International Standard*, Standard IEC 61499, IEC, Geneva, Switzerland, 2nd ed., 2012.
[28] *Programmable Controllers—Part 3: Programming Languages*, Standard IEC 61131-3, 2nd ed., 2003.
[29] *Compliance Profile for Feasibility Demonstrations*, Standard IEC 61499, 2014. doi: 10.1049/PBCE095E.
[30] A. Zoitl and H. Prähofer, "Guidelines and patterns for building hierarchical automation solutions in the IEC 61499 modeling language," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2387–2396, Nov. 2013.
[31] M. O'Connor, and A. Das, "SQWRL: A query language for OWL," in *Proc. 6th Int. Conf. OWL: Experiences Directions*, vol. 529, Oct. 2009, pp. 208–215.
[32] *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. Accessed: Mar. 19, 2019. [Online]. Available: http://www.w3g.org/Submission/SWRL/
[33] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson, *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Upper Saddle River, NJ, USA: Prentice-Hall, 2005.
[34] W. Dai, V. Dubinin, J. Christensen, V. Vyatkin, and X. Guan, "Toward self-manageable and adaptive industrial cyber-physical systems with knowledge-driven autonomic service management," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 725–736, Apr. 2017.
[35] S. Kumari and S. Rath, "Performance comparison of SOAP and REST based Web services for enterprise application integration," in *Proc. Int. Conf. Adv. Comput., Commun. Inform. (ICACCI)*, Aug. 2015, pp. 1656–1660.
[36] A. Zoitl, I. Hegny, and A. Schimmel, "Utilizing binary XML representations for improving the performance of the IEC 61499 configuration interface," in *Proc. 7th IEEE Int. Conf. Ind. Inform.*, Jun. 2009, pp. 66–71.
[37] E. Latronico, E. A. Lee, M. Lohstroh, C. Shaver, A. Wasicek, and M. Weber, "A vision of swarmlets," *IEEE Internet Comput.*, vol. 19, no. 2, pp. 20–28, Mar. 2015.
[38] P. Leitao, V. Marik, and P. Vrba, "Past, present, and future of industrial agent applications," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2360–2372, Nov. 2013.
[39] M. Marcos, E. Estevez, F. Perez, and E. Van Der Wal, "XML exchange of control programs," *IEEE Ind. Electron. Mag.*, vol. 3, no. 4, pp. 32–35, Dec. 2009.

WENBIN DAI (GS'09–M'13–SM'16) received the Bachelor of Engineering degree (Hons.) in computer systems engineering from The University of Auckland, New Zealand, in 2006, and the Ph.D. degree in electrical and electronic engineering from the Department of Electrical and Computer Engineering, The University of Auckland, in 2012. He was a Postdoctoral Fellow with the Luleå University of Technology, Sweden, from 2013 to 2014. He was also a Software Engineer for airport baggage handling system provider, from 2007 to 2013. He is currently an Associate Professor with Shanghai Jiao Tong University, China. His research interests include IEC 61131-3 PLC, IEC 61499 function blocks, industrial cyber-physical systems, knowledge-drive industrial automation, automatic code generation, and industrial edge computing.

PENG WANG received the bachelor's degree in aerospace engineering and the master's degree in electrical engineering from the Harbin Institute of Technology, China, in 2011 and 2013, respectively. He is currently pursuing the Ph.D. degree in control theory and engineering with the Shenyang Institute of Automation, Chinese Academy of Sciences. His research interests include in the areas of reconfigurable manufacturing system with special focus on model-driven engineering of the reconfigurable software.

WEIQI SUN received the B.E. degree in electrical and electronic engineering from the Harbin Institute of Technology, China, in 2018. He is currently pursuing the master's degree with Shanghai Jiao Tong University, China. His current research interests include automatic code generation for smart manufacturing and AI in industry.

XIAN WU received the B.E. degree in electrical and information engineering from Shanghai Jiao Tong University, China, in 2019. She is currently pursuing the master's degree with Shanghai Jiao Tong University, China. Her current research interests include automatic code generation for smart manufacturing, requirement engineering, and data analysis.

HUALIANG ZHANG was born in Beian, Heilongjiang, China, in 1976. He received the B.S. and M.S. degrees in measuring and testing technologies and instruments from the Huazhong University of Science and Technology and Shenyang University of Industry, respectively, and the Ph.D. degree in mechatronic engineering from the University of Chinese Academy of Sciences, in 2010. He is currently a Researcher with the Shenyang Institute of Automation, Chinese Academy of Sciences.

VALERIY VYATKIN (M'03–SM'04) received the Ph.D. degree from the State University of Radio Engineering, Taganrog, Russia, in 1992. He was a Visiting Scholar with 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, and in Japan and Russia. He is on joint appointment as a Chaired Professor of Dependable Computation and Communication Systems, Luleå University of Technology, Luleå, Sweden; and a Professor of Information and Computer Engineering in Automation at Aalto University, Helsinki, Finland. 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 article, in 2012.

GENKE YANG received the B.Sc. degree in mathematics from the Shanxi University, China, in 1984, the M.Sc. degree in mathematics from Xinan Normal University, China, in 1987, and the Ph.D. degree in systems engineering from Xi'an Jiaotong University, China, in 1998. He is currently a full-time Professor with the Department of Automation, Shanghai Jiao Tong University, Shanghai, China. His research interests include supply chain management, logistics, production planning and scheduling, and discrete event dynamics systems.