Dynamic Edge and Cloud Service Integration for Industrial IoT and Production Monitoring Applications of Industrial Cyber-Physical Systems

David Hästbacka, *Member, IEEE*, Jari Halme, Laurentiu Barna, Henrikki Hoikka, Henri Pettinen, Martin Larrañaga, Mikael Björkbom, Heikki Mesiä, Antti Jaatinen, *Member, IEEE*, and Marko Elo

***Abstract*—Industrial cyber-physical systems rely increasingly on data from Internet-of-Things (IoT) devices and other systems as continuously emerging use cases implement new intelligent features. Edge computing can be seen as an extension of the cloud in close physical proximity, in which some of the typical cloud computing loads are beneficial to run. This article studies data analytics application development for integration of industrial IoT data and composition of application services executed on edge and cloud. A solution is designed to support heterogeneous hardware and run-time platforms, and focuses on the service layer that enables flexible orchestration of data flows and dynamic service compositions. The unified model and system architecture implemented, using the open Arrowhead framework model, is verified through two representative industrial use cases.**

***Index Terms*—Arrowhead framework (AHF), condition monitoring, cyber-physical systems (CPSs), interoperability, production monitoring, system architecture.**

Manuscript received June 26, 2020; revised October 4, 2020 and January 28, 2021; accepted March 21, 2021. Date of publication April 6, 2021; date of current version September 29, 2021. This work was supported in part by the EU ECSEL JU under Grant 737459 and by Business Finland in the Productive4.0 project and in part by the Academy of Finland under Grant 310098. Paper no. TII-20-3109. *(Corresponding author: David Hästbacka.)*

David Hästbacka is with Tampere University, 33100 Tampere, Finland (e-mail: david.hastbacka@tuni.fi).

Jari Halme and Martin Larrañaga are with the VTT Technical Research Centre of Finland Ltd., 02044 Espoo, Finland (e-mail: jari.halme@vtt.fi; larraunanue@gmail.com).

Laurentiu Barna is with Wapice, FI-6520 Vaasa, Finland (e-mail: laurentiu.barna@wapice.com).

Henrikki Hoikka and Antti Jaatinen are with Metso Outotec Group, 33900 Tampere, Finland (e-mail: henrikki.hoikka@metso.com; antti.jaatinen@metso.com).

Henri Pettinen and Marko Elo are with CrossControl, FI-33100 Tampere, Finland (e-mail: henri.pettinen@crosscontrol.com; marko.elo@crosscontrol.com).

Mikael Björkbom and Heikki Mesiä are with Konecranes Plc, 05830 Hyvinkää, Finland (e-mail: mikael.bjorkbom@konecranes.com; heikki.mesia@konecranes.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2021.3071509.

Digital Object Identifier 10.1109/TII.2021.3071509

I. Introduction

INDUSTRIAL production systems are cyber-physical systems (CPSs) that integrate raw materials, equipment, humans, and processes on multiple sites into complex system of systems (SoS) that depend on data to operate efficiently and sustainably [1], [2]. Also, equipment and devices themselves rely on data throughout the life-cycle as smart product systems with the aim to offer support services optimizing operation or improving designs [3], [4]. Modern cyber-physical production systems (CPPSs) are based on composition of heterogeneous systems and components from different suppliers that need to exchange data and integrate services seamlessly at run time.

Cloud computing builds on the economies of scale in leveraging performance, scalability, and reliability in a cost-efficient and manageable manner. For Internet of Things (IoT) applications, cloud computing can supplement the limited storage and large scale compute and analytics demands. IoT data are being generated at an increasing pace with new applications making use of this data [5] and also in the physical context with real-time requirements [6]. The challenge is to analyze data efficiently when striving for data-driven operations [1], [7]. In cloud computing, transmission of data, allocation of processing resources, and finally, making results available typically includes delays and transfer costs [8], [9].

Edge computing can be seen as cloud computing brought closer to the actual applications and users, and it is sometimes also referred to as the extension of the cloud in a close physical proximity [10]. Edge computing is characterized by short response times and reduced transfer of data to the cloud.

Application areas that benefit from low latency include typical production and equipment control tasks, autonomous decision making or monitoring applications. With edge computing data does not necessarily need to be transferred over the Internet at all, especially if it is only used locally, or that only a subset needs to be aggregated to the cloud. For applications where large amounts of data are processed, this can be a benefit in terms of cost of transfer, storage, and processing.

Despite efforts in Industry 4.0, smart manufacturing, and related paradigms standardization [11], [12], the integration challenges have not been solved. The challenges are not only

related to data interoperability produced by the heterogeneous devices and systems [13] but also to interoperability of system architectures [5] and dynamic compositions in their interplay [2] as well as whole ecosystems. The production domain challenges are emphasized by the still yet immature information and communication technologies available for implementing a seamless, open cloud-edge-IoT continuum [6], [14], [15].

This article researches functionalities and technical features required to flexibly orchestrate cloud- and edge-based services in production IoT applications. A unified approach is proposed to manage heterogeneous data flows from IoT devices and other production systems, and an integration architecture is suggested to orchestrate configurations to and between application service components both on edge and cloud. The architectural approach is based on the service integration model of Arrowhead framework (AHF) [16]. The approach is designed to accommodate the reconfiguration of data flows that can span to hundreds or thousands in numbers and/or produce huge amounts of data. More flexible orchestration of computational services across edge and cloud is required due to new emerging data use cases, evolving application systems, and balancing network traffic and computing resources. Two complementing industrial use cases, production assembly monitoring and condition monitoring in rock crushing, are presented implementing and evaluating the approach. The contributions of this article can be summarized as follows.

1) Outlining requirements and designing of a novel conceptual architecture for interoperability and dynamic cloud-edge orchestration configuration of both data flows and services in production IoT.
2) Evaluating the AHF service integration model and the implemented conceptual architecture for meeting the requirements in two real industrial use cases when developing SoS applications for the CPS.

The rest of this article is organized as follows. Section II presents related work integrating IoT, edge computing, and cloud computing in industrial production. The requirements derived from two representative industrial use cases are explained in Section III. A conceptual architecture outlining the objectives and opportunities is presented in Section IV, and enabling software technologies and components used for implementing the technical architecture are described in Section V. Implementation in two industrial use cases and the results are presented in Sections VI and VII. A discussion is provided in Section VIII, and finally, Section IX concludes this article.

II. Related Work

Internet-based monitoring of production assets and as connected smart products have been previously discussed in [1], [4], [7], and [17]. These approaches, however, do not sufficiently consider the need for interoperability and how to engineer compositions efficiently. Integrated solutions are needed to deliver a unified approach to orchestrate services across technologies and heterogeneous devices, further increasing with IoT as studied and proposed in [6], [15], and [18].

In [19], a system design method and system architecture is proposed for the smart product-quality monitoring systems for intellectual system design considerations for different needs, and how these can be implemented in Industry 4.0 settings based on big data, IoT, and artificial intelligence (AI). A system for machine tools based on fog computing, claimed to improve autonomy and collaboration using data available, is presented in [20]. In [21], an approach is proposed where data acquisition sensors are distributed across machines, and feature extraction and health condition classification is on fog nodes. Maintenance service architectures for the CPS making use of data have been presented in [22]–[24]. These approaches are either only conceptual, mainly focused on the data, or lack interoperability and integration to wider ecosystems.

The AHF has been developed as an infrastructure for services in various fields of application, initially for IoT integration [25], and an SoS composition model has been proposed with requirements for industrial information distribution using graphs [26]. Using the AHF, an approach decentralizing ISA 95 production functions to the lower levels has been presented [27] also utilizing edge computing in a service-based SoS.

Offloading computation to the edge has been studied in [8] and [9], and edge computing and related paradigms have been surveyed in [10], [14], [28], among others. In [29], the authors propose a method for the dynamic management of service resources for cloud, fog, and edge-based CPPS. The model includes descriptions of the environment, resources, evaluation metrics, and algorithms to optimize the configuration to tackle different needs and heterogeneous systems and hardware. The placement of heterogeneous edge services has been studied by [30] to optimize response times in the mobile edge-cloud context. The use of edge and fog computing for data management has been studied by [31], where the load of sensor data preprocessing has been distributed away from the cloud.

III. Technical Needs and Requirements

Two real industrial use cases provide requirements and the context for validating the developed approach and the solution.

A. Manufacturing Monitoring

Product assemblies require multiple manufacturing steps until a final product is ready. Hoisting equipment is typically used to handle different components to the assembly station or cells. Manufacturing is usually divided into several subprocesses, each with their own set of steps using different tools from different manufacturers. Operation is often manual and independent from other manufacturing cells as technology to connect the disparate systems has not yet become mainstream.

The state information and performance metrics need to be gathered from the monitored systems and subsystems in a manufacturing process and immediately communicated further. The information needed can be different for each manufacturing process, depending on the monitoring needs. Furthermore, the data collection solution can be device specific. Thus, for manufacturing monitoring, the system should enable independent and exchangeable software components and communication. In this use case, it would be beneficial to be able to deploy and manage arbitrary software services in a diverse set of devices to enable the different monitoring needs.

B. Condition Monitoring

Vibrating screens are devices used to separate materials with different granularities such as different kinds of ores. These machines are not mechanically complex but monitoring is required as large accelerations and masses cause wear. In a typical use case, the screen is part of a production line where an unexpected failure will cause larger halts and production losses. Condition monitoring measurements can be gathered from the screen movement acceleration levels and the bearings.

Different use cases have varying needs ranging from "something is wrong" to identifying the problematic part and on to predicting future failures. For example, a predictive system using a digital twin model may need only a daily acceleration sample to detect something has failed, whereas preventing additional failures may require millisecond-level measurements.

It is straightforward to build systems based on transferring all sensor data to the cloud for processing but this is not always feasible in practice. The costs associated with data transfers, storage, and analysis become prohibitive for an equipment manufacturer on a large scale. This is especially the case for mining operations in remote locations across the globe. The situation could be different if data and analysis results were utilized more extensively as part of the production.

The objective of the condition monitoring use case is to simplify the architecture of the condition monitoring system so that it would be similar on different levels, even when the physical components and measurement systems are different. Also, evolving the systems and shifting processing from cloud to edge should be possible as the product matures and knowledge of the measured phenomena increases.

C. Summary of Requirements

As functional requirements, the system is expected to fulfill the following:

1) exchange IoT data and other data streams from various sensors and systems in a unified and standardized format, supporting rerouting to different consumers, e.g., AI;
2) (pre)process data and execute advanced data analytics functions locally on the edge and/or in the cloud, balancing and optimizing where computation is taking place;
3) centrally manage dynamic system configurations and decentralized information flows, and authorizations of service compositions into applications.

Additional features and requirements of the system are as follows.

1) *Information security*—Ensure information security from devices to processing, and that dynamic compositions are only formed for trusted service compositions, and hence, also enable different parties to operate on the same data and in the same networks.
2) *Performance, scalability and reliability*—Support communicating simple values to massive raw measurement data reliably to existing and new application services.
3) *Manageability and support*—Methods for managing applications without system specific dependencies on individual IoT devices or participating systems, and scalability to develop new application system integrations.

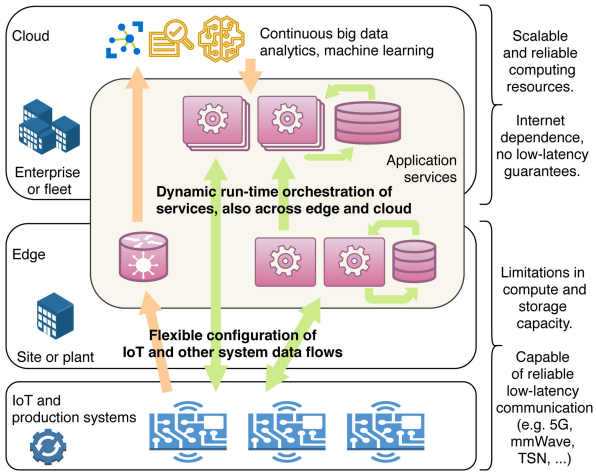

Fig. 1. Architectural concept for composing data-based applications in which data flow from IoT and other plant floor level systems are configured and dynamically orchestrated with application services on edge and cloud.

4) *Interoperability*—Based on open models and specifications not limited to specific hardware or protocols, and capable to cater for a wide range of application uses.

IV. Flexible Architecture for Edge and Cloud

To enable the requirements presented, a conceptual architecture for data and software services is outlined as shown in Fig. 1. Paramount to the design is to enable the development of new applications based on production data or IoT measurements for various purposes. An important goal is also supporting various existing hardware and software platforms.

Traditional IoT relies on transferring measurement data to the cloud for analysis and development. Many IoT applications, however, produce significant amounts of data and there are applications where it is neither feasible nor desired to transfer everything to the cloud. Preprocessing or analytics on the edge saves data transfer and can also reduce Internet dependence. Reliable low-latency and real-time communication can be implemented locally, whereas Internet services cannot be guaranteed the latencies sometimes required.

Edge computing is not a substitute for cloud computing but a complement [6]. Edge computing resources are limited in the sense that additional compute resources cannot be easily provisioned as in the cloud. Therefore, the solution builds on the premise that computational functions can be deployed both to edge and cloud. If new computationally demanding tasks emerge they can be executed on the cloud and connected similarly to the applications as they were in the local premises.

Fleet wide analytics, and, e.g., training of machine learning (ML) models, is typically taking place in the cloud (as shown in Fig. 1) because data may need to be gathered from several sites. In the future, edge computing demands are expected to increase in this area not only with ML inference [5] but also with contextualized training at separate edge locations. These are tasks that occur sporadically on the edge, whereas preprocessing, for instance, is something that needs to always run and for which the capacity and storage needs to be reserved.

The set of available services on edge and cloud can be defined as $S = \{S_{11}, \ldots, S_{ij}\}$. The composition of edge and cloud services can be formulated as a graph $G_s = \{V, E\}$, where V denotes the vertex set between all services and E the interactions between services. Considering the quality-of-service (QoS) attributes of different service instances, we can further formulate the composition problem into a service graph $G_s = \{V, E, \text{QoS}\}$. By selecting based on components of QoS, e.g., cost, delay, and accuracy, the composition of suitable services for the composition can be achieved. At run time, this composition needs to be done when new systems come online or when QoS attributes are changing. The QoS component or its optimization is not discussed further as this article focuses on the enabling technology for orchestrating and configuring the composition of edge and cloud services at run time.

As application needs differ, it is essential to have flexibility in the way data flows are configured starting from field level IoT devices and sensors. IoT devices initiating the connection get configurations set from managing infrastructure, e.g., where to push acquired data. Similarly, application services that retrieve (pull) data from IoT devices or data stores can retrieve their configurations on startup and update them periodically in case compositions or information flows need updating.

Central to the concept is managing the configurations and service compositions within the infrastructure in a unified method independent of characteristics of the different participating devices, services, or systems integrated. This requires agreement upon application programming interfaces (API) as well as setup and configuration mechanisms.

Application systems developed on the architecture need to use existing standards or agree on message structures per application system or per integration. Messages and application service APIs need to follow a unified, standardized structure in order to be interchangeable in flexible compositions.

V. Technical Architecture and Implementation

A. Approach Overview

To realize the concept, a set of novel tools and practices must be adopted. In this section, Arrowhead is first presented as the chosen open service-oriented architecture (SOA) model used as the basis. Then, the various computing devices residing on the edge of the network are presented. Particularly of interest is their functionality, hardware and software capabilities and connectivity interfaces. Finally, the cloud-side integration to the production systems is examined. Together they demonstrate the multitude of different hardware and software platforms often needed.

B. Arrowhead for Interoperable Service Integration

A central part of the software architecture is built around AHF (version 4.1.3) as the means to discover, bind and authorize application compositions. In the SoS approach of the AHF [26] systems are used as building blocks to form the larger systems. In the AHF, systems are divided into the following three groups.

1) Application systems implement the business logic of actual solutions by providing and consuming services.

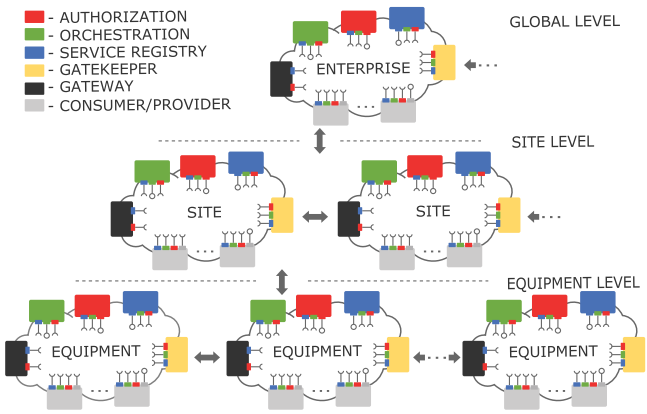

Fig. 2. Arrowhead core services include Service Registry, Authorization, and Orchestration. Service clouds and their application services can be relayed to other clouds through gateways, also to support logical hierarchical structure.

2) Core systems include orchestrator, authorization, and service registry [16], which provide means for service discovery, authorization, and registration, respectively. The application systems form the SoS via them.
3) Supporting core systems provide extra functionality for the SoS. These include functions like event-handling, message brokering via gateways, QoS monitoring, etc.

A main component within an AHF SoS is the orchestrator that queries the service registry on behalf of the consuming application systems for available services that providing application systems have registered. Before responding to the consumer, the orchestrator makes sure that the consumer is allowed to use that particular service. The orchestrator has the following two modes: Static store mode, hosting a defined set of rules on who should provide and consume which service; and Dynamic mode, in which the orchestrator responds with a list of available services, which can be reduced with additional request parameters set by the consuming system including, for example, desired provider and key-value metadata.

One SoS working under one set of mandatory core systems is often referred to as an AHF local cloud. These local clouds can be connected to each other via Gateway and Gatekeeper systems [32], which belong to the supporting core systems. The Gatekeeper system is responsible on relaying service discovery requests from one local cloud to another in an intercloud service negotiation process. After a successful negotiation, the Gateway system is leveraged, and a tunnel between the local clouds is established. Multilevel AHF clouds can be arranged hierarchically, e.g., into global level clouds, local site level clouds, and even device or cell clouds (see Fig. 2).

To summarize, the AHF process of registering providers and consumers, authorizing parties, orchestrating, and finally, enabling systems to interact on the local level is illustrated in Fig. 3. Depicted here is the dynamic orchestration method. The sequence has the following steps.

1) Both providing and consuming service connect to the Service Registry core system and register themselves.
2) Authorization system is informed about the access rights of the new service.

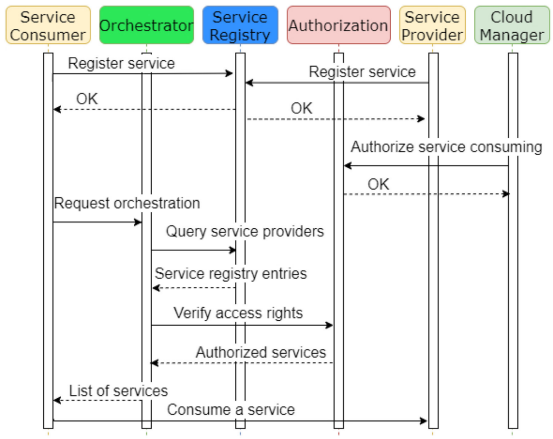

Fig. 3. Before service consumption, the AHF dynamic service orchestration requires that consumers and providers are first registered and authorized for each other.

3) Service consumer requests a suitable service end point from the Orchestrator. By having the possibility to specify multiple search attributes, the consumer can be certain to receive the wanted provider's address.
4) The Orchestrator looks from the Service Registry for a match to all the specified attributes. If it finds one or more, it will check that the consumer is authorized to use them from the authorization system.
5) Consumer will receive a list of suitable providers that it is allowed to consume and can start interaction.

C. Edge Gateways and Software

The AHF core services are installed in Raspberry Pi 3 Model B+ and industrial embedded Linux PCs in the local network. The gateways host services, such as data storage registered with AHF, providing a (HTTPS) REST service for storing the messages received to a local database. The data model that has been used is the open MIMOSA data model, which is an ontology for defining and managing operation, condition-based maintenance, and predictive maintenance activities based on data and metadata. In addition, edge gateways also host communication protocol translation services, registered as AHF services, e.g., to convey MQTT messages to REST, or vice versa, to improve the interoperability of IoT devices and application services.

The lightweight edge devices, presented in the next section, are AHF compliant service consumers for the aforementioned data storage service as well as other services adhering to the same API and information model. They request orchestration from the AHF that searches for registered and authorized service providers for the particular consumer based on the dynamic rules specified by the consumer.

D. Edge Devices and Measurements

Edge level data acquisition (DAQ) instances are realized with two lightweight devices and one industrial grade DAQ for hoist movement and load monitoring. The two first ones are a Raspberry Pi 3 Model B and a NodeMCU (ESP8266). Both devices measure the signals from the hoist and consume data storage services registered in the AHF. Regarding the Raspberry Pi, it measures the analog load value (ADC 10 bits) with 10-Hz sampling frequency through an extension board attached directly to the pins of the Raspberry. Whereas the NodeMCU is concerned, it has an internal ADC pin (10 bits) for an analog signal that is used for measuring the load signal. For each second ten load values are read during the first 400 ms with a 40-ms interval between samples (25 Hz). After the sampling is complete, a message is formed containing both ten load values and three bits for the hoist movement.

The reference hoist DAQ is a robust IoT edge device for remote management, measurement, and control. The real-time linux device features an ARMCortex-A5 processor, has wireless connectivity (3G, WLAN, BLE), comes with built-in sensors incl. acceleration, and supports a multitude of standard wire protocols. This AHF-compliant DAQ instance only needs to be aware of the core services after which it dynamically configures itself to send configured measured hoist process data to all destinations returned by the AHF orchestration, in addition to its own IoT platform.

An industrial computer can be used to visualize the data collected at the edge level. The computer is based on an ARM Quad Cortex-A9 CPU and has a 12-in multitouch display attached to it. It runs a custom-made, and thus, lightweight, Linux distribution as its operating system and likewise comes equipped with multiple connectivity options. On the software side, it uses Qt runtime to render the visualization dashboard. Alternatively, it can show a dashboard embedded into a web page by using a QtWebEngine powered browser.

E. IoT Platform and Cloud Integrations

A first step in integrating an existing IoT platform to the AHF would be to simply define it as a single system with multiple interfaces. However, in the AHF world, the tools and applications offered by an IoT platform should be ultimately visible as completely independent services, programmatically configurable and individually accessible, and easy to interchange with other similar services. Such tools include, for instance, data storage for big data and historic data, data analytics, data visualisation, alarms and events, reports, data forwarders (e.g., from one communication path or protocol to another), and device management and control, including legacy devices. The goal in integrating different IoT platform tools into the AHF should be in building up flexibility of selecting and dynamically interchanging tools and applications offered by a specific platform or manufacturer with separate similar or aggregated components offered virtually by any other. From the AHF point of view, this only requires that IoT platforms update the authorization service with their own information, which is then stored and relayed upon request by the AHF.

The IoT platform integrated in this implementation is cloud native and can be deployed to any environment. It is a complete IoT tool suite and platform that allows to build web, mobile, cloud, and reporting applications in minutes with easy to use tools ranging from the drag-and-drop content building to plug-and-play connectivity. Initially, the simpler approach was taken for integrating the IoT platform with AHF, i.e., a single system

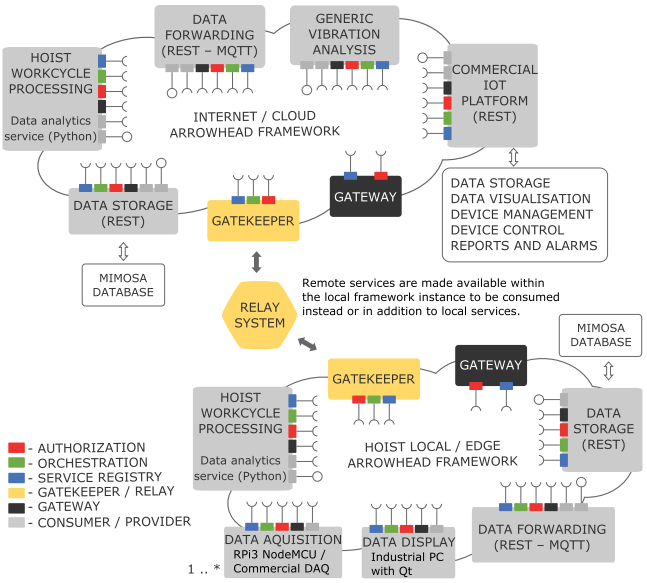

Fig. 4. Services deployed in the production monitoring use case can be flexibly exchanged between edge and cloud levels, e.g., based on availability or capability, using the AHF relay system linking trusted framework instances.

definition was made with separate interfaces for some of the tools offered, instead of transforming the platform into separate units, as required by proper AHF architecture.

VI. Industrial Use Case Examples

The architectural concept of edge-cloud orchestration is demonstrated using two representative industrial production use cases with complementing features. The use cases also evaluate the solution and the suitability of the AHF for this purpose and the requirements outlined in Section III-C.

A. Production and Assembly Monitoring

This use case aims at creating an intelligence system that provides end-to-end process visibility and task optimization recommendations for material handling production processes. The system is deployed as a local factory cloud on the edge and there can be many such instances globally.

The solution uses a highly decoupled architecture based on SOA and edge computing for gathering, processing, analyzing, and exchanging data, and for providing monitoring and intelligence services to the factory floor staff. Only local site and global level AHF clouds are used, as shown in Fig. 4.

The implementation involves enhancing the material handling equipment with data gathering devices (RPi3 and NodeMCU as well as reference commercial DAQ; see Section V-D) for sensing acceleration, load, torque, current, location, and material handling events. The measurement devices are part of the site level local cloud.

Real-time data gathered by the equipment level devices is continuously processed on the edge gateway in the site level cloud, using dedicated intelligence algorithms. These are implemented as data analytics services either as traditional Python-based applications or as Docker-based microservices. Both the raw data and processed results are saved in data storage service (REST HTTP and Mimosa SQL DB). The analytics services as well as storage services implement a similar API expecting a predefined JSON format content. The results are further visualized for the factory personnel using an industrial PC with an embedded display, as well as transmitted to global cloud for long-term storage and future analysis. For local visualization, a local storage service is used for querying the desired analysis results.

B. Condition Monitoring of Vibrating Screens

The second use case leverages AHF's dynamic service discovery functionality in condition monitoring application of mining screens. The setup at the edge is collecting vibration data with wireless sensors equipped with Bluetooth LE radios and accelerometers. The raw data are collected with an industrial Linux PC, which utilizes various signal processing techniques, functioning both in frequency domain and time domain, and transforms the data to a format where the physical state of the screen can be presented in noise free and more dense format. Later on, these data points are transferred to upstream processes residing in cloud to perform refinement and analysis, which finally enables detailed information on the state of the product to be given to the owner.

In the demo application, the details of the calculations and data collection were taken as is, and the backend producing the refined data was thought as being a black box. The demo application interfaced with the backend producing the data via an OPC UA server, from which the fresh data were pulled to a database. Additionally to the fact that data producing mechanism already existed, this was also done because the AHF aims to provide a generic approach on service composition, and therefore, does not dictate any implementational details on the level of the business domain or the larger scale purpose of the application. In other words, from the perspective of the AHF, the set of services used in transferring the data can be made in a domain agnostic manner. The demo application consist of one AHF local cloud at the edge of the network and one in a public cloud, as shown in Fig. 5.

The communication between the clouds is handled via Gatekeeper and Gateway systems. However, since the AHF only provides the means for service discovery, authorization, and orchestration, various tooling that fulfill different vacant roles, were needed as well. As a whole, the two clouds in the demo have the following components.

1) Two systems at the edge: Screen Data that consume a service used for pushing the data to the cloud and Screen Control that consumes services through which the edge computers configuration is fetched. The configuration includes a payload filter and interval of the data pushed to a particular counterpart.
2) Two systems in the cloud: One application system is responsible for providing a service that the data are pushed to and another one is responsible for offering services related to the configuration of the edge computers, both toward the edge and toward an user interface (UI).

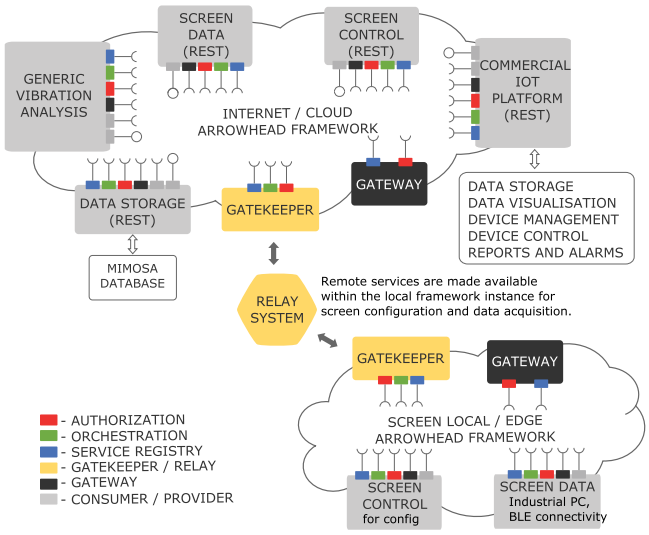

Fig. 5. In the condition monitoring use case, the AHF is used to connect data services and retrieve screen control settings. This is done uniformly independent of their location thus future-proofing increased use of the edge.

All application system components are Docker containers and Docker-compose is used for deployment in cloud instances. PostgreSQL database is used for caching data and configuration, and PostgREST is used between the application systems and the database as a medium serving the contents of the DB through a REST API. The AHF Gatekeeper and Gateway are used as described in Section V-B.

While the demonstration is limited to two AHF cloud instances, the implementation supports multiple sinks for data and multiple sources for configuration. Each destination can be configured separately and different subsets of available data points can be directed to each at different intervals, as long as the sink is discoverable through AHF's orchestrator system and a configuration determining the interval and subset exists.

One additional goal tackled with the AHF was easing installation and replacement of the edge computers. The systems responsible for configuration do not only configure the data flow, but are also capable of sending generic configuration objects. These objects are stored in the database and can be used system wide, e.g., when various parameters are initially set on system boot up. The parameters can be related to many arbitrary use cases, but one particularly interesting use is the bootstrapping of the data collection black box used as the data source. To enable this kind of behavior, where the AHF makes sure its dependencies downstream are up, the AHF has to be started during the boot-up process and it has to be configured in a way that discovering the service needed for the configuration is possible. This can be easily achieved, for example, with commonly used init and service managers found in Linux.

VII. Results and Evaluation

A. Meeting Requirements and Use Case Experiences

The presented approach focuses on orchestration of software services and data flows across device, edge, and cloud levels in data-driven applications. The objectives, presented as requirements in Section III-C, were to flexibly redirect data flows from IoT devices and other production systems, configure system compositions dynamically from services executed both locally on the edge and on Internet cloud platforms, and manage the composition centrally in a unified manner regardless of the hardware or software platforms.

Benefits of the solution architecture and using the AHF are as follows.

1) Flexible operation based on the set of running services, i.e., the composition of needed services can be accomplished based on services available at a given time.
2) New services can be dynamically added to the system. For instance, new data analysis services can be deployed to the local site cloud and measurement devices can send data to them without extra device configuration.
3) Services at global cloud can provide their services in case the local services are not running, or in case, the data storage or processing is desired to be done at global level. Services can be deployed to local cloud, e.g., to reduce latency and cloud storage and data transmission costs.
4) An orchestration template can facilitate duplicating the setup to other local clouds in similar commissioned systems, thus, making it easy and efficient to scale.

The developed solution proved in both industrial use cases that systems can be composed interoperably and securely from different kinds of IoT devices, sensor systems, and application services both on the edge and in the cloud. Data flows can be changed at run time and the configuration is centralized to the core AHF services utilized in the approach. In the production monitoring use case, it was especially observed that information from multiple sources can be managed and integrated with services running both on edge and cloud. For the condition monitoring use case, the uniform architecture further enables evolving the system and increase the use of edge computing in analytics applications. In summary, the solution meets the earlier identified functional requirements.

Additional required features were information security, reliable performance and scalability, uniform manageability, and interoperability with hardware and protocols.

In the implementation, the AHF is used as the SOA infrastructure providing an open, unified model how services are discovered, authorized, and bound at run time. Returning to the composition problem $G_s = \{V, E, \text{QoS}\}$, presented in Section IV, it can be stated that the developed solution enables compositions to be formed from a set of possible services. This is illustrated in Fig. 6 where selected data processing services are connected by the solid lines and other possible compositions are denoted by the dotted lines. For this, the AHF provides the technical functionality for run-time composition, but in the dynamic orchestration, requires consumers to filter the services to be returned. As a result, the optimization of what services are to be composed is not given by the AHF, unless the static mode is used, and it is on the application system or SoS responsibility to maintain. This can be considered a benefit, as this is application dependent by nature, but in that sense, the AHF does not fully solve of this and requires supporting management operations to be integrated along the AHF.

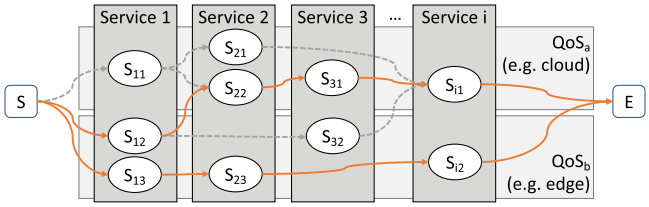

Fig. 6. Composition of edge and cloud services can be seen as a directed acyclic graph of data flowing between service instances part of different environments with different QoS attributes (possibly even per service).

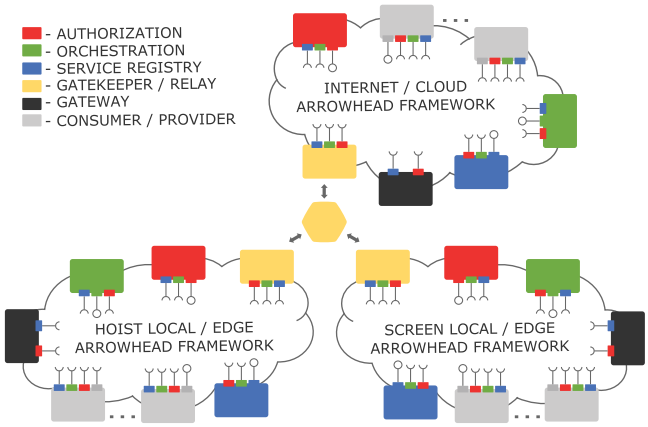

Fig. 7. In addition to relaying between edge and cloud service consumers and providers, the AHF can manage authorizations across multiple domains, e.g., supply chains. In the demonstration, the use cases shared the Internet cloud.

The AHF solves several of the security challenges in dynamic service compositions as authorization is shifted from individual components to a central entity, building a chain of trust between consumers and providers of an AHF cloud. Similarly, the AHF Gatekeeper relay paves the way for application service ecosystems extending this model between multiple AHF clouds. In this regard, the AHF can be seen meeting the technical requirements of the architectural concept outlined.

The AHF does not limit communication protocols or message semantics of application systems, and thus, performance and reliability are solely dependant on the application systems once the orchestration and authorizations have been set up. This means that even hard real-time communication is possible, e.g., for the CPS, given that such protocols are being used between the interacting application systems. In our use case examples, both high-frequency measurements as well as aggregated analytics were communicated successfully between consumers and providers as they are not conveyed via the AHF or any other possible central bottleneck.

Our solution focuses mainly on the software service layer without considering how the applications are developed or deployed in their run-time environments. Using the AHF, the individual consumers and providers are composed, authorized, and configured similarly independent of the different underlying implementations, as demonstrated in both use cases. Interoperability of application services is ensured through uniformly defined service interfaces and agreed data models. This leaves freedom in implementation for different hardware platforms, operating systems, and execution environments.

B. Engineering Efficiency Evaluation

The AHF is expected to improve the engineering efficiency of SOA-based SoS through easily developing new services and easily integrating services and developing new compositions.

In earlier versions, like 4.1.2, development of application systems included writing boilerplate code for service registration and orchestration calls via common HTTP client libraries to the core service interfaces. There were some templates, where users could insert their code, but this was not ideal, and the development was stiff due to lack of clear boundaries between user code and boilerplate. This meant that the development efforts needed were high compared to what the framework offered, since if the templates were not used, every developer effectively had to write their own "library."

Recently, more systematically maintained and well-designed libraries for application system development have emerged and much of the logic needed by every application system is wrapped behind an API with clear boundaries. One example is Arkalix for Java [33], which makes implementation of AHF compliant application systems streamlined. Arrival of libraries is a good sign since they also enable smoother introduction of new features for the AHF, which could lower the learning curve for new developers and save time for more experienced ones.

Regarding integration of services and developing new compositions, the AHF can significantly improve the efficiency once the infrastructure is deployed. New applications can easily be developed tapping into the services and, by using the authorization and orchestration model, no changes are necessary, e.g., IoT devices or existing systems providing services.

The AHF shifts binding of services, authorization of consumers, and orchestration of compositions away from the individual components (e.g., compute modules, storages,...). It does this also in a secure way, required by today's connected production systems, based on certificates and tokens issued in the service clouds. Doing all these in an uniform manner compared to multiple different proprietary solutions can have a big impact on the SoS engineering efficiency. Additionally, the AHF supports integrating AHF clouds, e.g., for larger ecosystems such as supply chains. In the use cases presented, the same Internet level AHF cloud was used as illustrated in Fig. 7.

Analyzing composition engineering efficiency, we can define the total effort simplified as $E_{\text{tot}} = M * E_i + N * E_f$, where M is the number of different integration models, E_i the average integration effort for a unique model, N is the number of unique services or components to be integrated, and E_f the effort for functional integration of such a component (that is consider similar independent whether the AHF is used).

As a theoretic example, assuming integration E_i is modestly one third of the functionality effort E_f, $N = 10$, and reducing the number of different integration models M to one from $M = N = 10$, we can estimate an effort reduction of over 20%. This oversimplified example does not count for individual

implementation differences but shows the improvement potential shared integration models could bring, and further transfer to operation and maintenance. This requires that chosen integration models used in integration are in the same effort magnitude, as also brought up earlier for the individual services and libraries.

VIII. Discussion

Edge computing can provide real value for many industrial IoT applications but introduces additional management complexity in addition to the cloud layer management. As new data use cases emerge, it is important to be able to move computational loads flexibly across edge and cloud. Due to heterogeneity of execution platforms, and lack of federation techniques to provision computational resources also on the edge, there is yet no established model for implementation.

A fact often misunderstood by new adopters of the AHF is that although it facilitates the development of services, it does not implement the actual services or act as a middleware for the data exchange. Thus, it is capable to support a multitude of different protocols (even real time), hardware and devices, and existing platforms and their integrations. Compared to approaches such as OPC UA and MQTT/AMQP integrations, it is more generic and provides a broader range of support for different protocols and systems to be integrated while retaining uniform configuration and authorization means. However, compared to OPC UA or MQTT/AMQP, it does not provide defined application system APIs or the detailed information exchange semantics OPC UA offers. This means that such application system interfaces and messaging semantics need to be agreed upon and this is also AHF's intention.

As the AHF only provides an SOA infrastructure, the following tools are also needed: run-time environment for services, management and monitoring of services, deployment and service update mechanisms, and management of service orchestrations for different deployments. In the use cases, dedicated edge devices were installed and proprietary systems were used to manage these devices, as well as deploy, run and monitor the different services running in the local clouds.

A lesson learned from the use case implementations is that a configuration or management agent or service could become useful as soon as the SoS is comprised of more than two or three systems. This would make the necessary registration and authorization steps easier on behalf of the systems involved.

Through its core and support services and run-time configuration flexibility, the AHF could relief significant burden from the implementation costs of complex SoS solutions, especially in dynamic environments. However, in its current implementation, the AHF still has noteworthy limitations: For instance, service authorizations are based on the numerical ids of consumers and providers assigned by the AHF upon their successful registration. This requires that before the authorization can even be configured, all the consumers and providers are registered and their AHF numeric ids are known to the cloud manager. Though quite manual and error prone, this is perhaps acceptable when it has to be done only once. However if systems reboot often their AHF internal ids change and the authorization process has to be repeated. Same happens if for any reason the AHF itself has to be restarted. This numeric id approach becomes even a bigger challenge when services are on different clouds handled by different managers. This means authorization can only be done at run time and currently it cannot be preconfigured. Another challenge is that even though dynamic orchestration will provide all the services fitting the specified criteria, it will not check nor signal possible service incompatibilities in terms of, e.g., data format, communication protocol, version, etc. Simply put a consumer service might not be able to consume the provided service. As a consequence of the previous limitation, complex orchestrations are also not possible yet. That is, dynamic employment of data and protocol translators or requesting one service to be pipelined through another service at the registration time (i.e., service specification) and orchestration time. For instance, orchestrating a data analysis service A to consume process data from service B and only provide the results in a specified format (unknown internally to service B but available through a translator service T) to service C. One can see that this would be a key feature especially for edge devices with limited resources.

Such problems are currently being overcame through use case specific components implemented externally to the AHF but solid work is undergoing enabling their eventual implementation in the AHF through SySML and AHF support core services and systems (e.g., translation and plant description).

Considering that both AHF and most IoT platforms originate in the concept of IoT middleware aimed at linking hardware and devices to application layers, one could reasonably argue that they are partly redundant. There are a multitude of IoT platforms already implementing the functionality found in the AHF. In practice, however, there are plenty of reasons for them being still quite complementary. Whereas some may be interested in a one-stop IoT platform solution, others may be inclined to put in the extra effort of setting up their own in order to, e.g., reduce costs, avoid vendor lock-in, or take advantage of the other benefits of using open source. Another important thing to consider is that by abstracting IoT to services and enabling their interoperability, the AHF does offer the means for easily building an open IoT-based automation solution but it does not yet come with all the tools and extensive communication that proprietary IoT platforms typically offer. As the AHF is a maturing open source project with expected improvements in certain configuration and utilization aspects, it has the potential to become the open SoS integration method.

IX. Conclusion

Edge computing has several interesting uses for industrial IoT and industrial CPS. In many cases, IoT devices generate large amounts of data that combined with their huge amount in industrial production applications requires using computing also closer to the source of data. Also, as new data-based applications emerge, new models are required to build application systems securely and efficiently.

This article presented an approach to orchestrate services and industrial IoT data flows across edge and cloud levels for data-driven applications in production environments. The

developed model and architecture was based on the AHF for SOA infrastructure and evaluated in two industrial use cases. As a result, application services running both on edge and cloud could be dynamically composed in a secure, unified way to operate with IoT device and production system data. In conclusion, the AHF does offer the means for building an open IoT solution but it does not yet come with all the tools and ease of use as a proprietary IoT platform. As a maturing open source project with expected improvements, it has the potential to become the open SoS integration method.

An interesting future research direction in industrial CPS is provision of edge resources that could also take QoS and real-time constraints into consideration. Truly edge native solutions could be developed when these would be combined into a pool of compute function building blocks across edge and cloud.

References

[1] A. W. Colombo, S. Karnouskos, O. Kaynak, Y. Shi, and S. Yin, "Industrial cyberphysical systems: A backbone of the fourth industrial revolution," *IEEE Ind. Electron. Mag.*, vol. 11, no. 1, pp. 6–16, Mar. 2017.

[2] 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.

[3] A. Ghanbari, A. Laya, J. Alonso-Zarate, and J. Markendahl, "Business development in the Internet of Things: A matter of vertical cooperation," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 135–141, Feb. 2017.

[4] B. Liu, Y. Zhang, G. Zhang, and P. Zheng, "Edge-cloud orchestration driven industrial smart product-service systems solution design based on CPS and IIoT," *Adv. Eng. Informat.*, vol. 42, 2019, Art. no. 100984.

[5] H. Ning, Y. Li, F. Shi, and L. T. Yang, "Heterogeneous edge computing open platforms and tools for Internet of Things," *Future Gener. Comput. Syst.*, vol. 106, pp. 67–76, 2020.

[6] L. Bittencourt *et al.*, "The Internet of Things, fog and cloud continuum: Integration and challenges," *Internet Things*, vol. 3/4, pp. 134–155, 2018.

[7] B. Cheng, J. Zhang, G. P. Hancke, S. Karnouskos, and A. W. Colombo, "Industrial cyber-physical systems: Realizing cloud-based big data infrastructures," *IEEE Ind. Electron. Mag.*, vol. 12, no. 1, pp. 25–35, Mar. 2018.

[8] N. Hassan, S. Gillani, E. Ahmed, I. Yaqoob, and M. Imran, "The role of edge computing in Internet of Things," *IEEE Commun. Mag.*, vol. 56, no. 11, pp. 110–115, Nov. 2018.

[9] C. Cicconetti, M. Conti, and A. Passarella, "Architecture and performance evaluation of distributed computation offloading in edge computing," *Simul. Model. Pract. Theory*, vol. 101, 2020, Art. no. 102007.

[10] A. Yousefpour *et al.*, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *J. Syst. Arch.*, vol. 98, pp. 289–330, 2019.

[11] F. Fraile, R. Sanchis, R. Poler, and A. Ortiz, "Reference models for digital manufacturing platforms," *Appl. Sci.*, vol. 9, no. 20, 2019, Art. no. 4433.

[12] T. Burns, J. Cosgrove, and F. Doyle, "A review of interoperability standards for industry 4.0," *Procedia Manuf.*, vol. 38, pp. 646–653, 2019.

[13] V. Jirkovský, M. Obitko, and V. Mařík, "Understanding data heterogeneity in the context of cyber-physical systems integration," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 660–667, Apr. 2017.

[14] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Gener. Comput. Syst.*, vol. 97, pp. 219–235, 2019.

[15] L. M. Vaquero, F. Cuadrado, Y. Elkhatib, J. Bernal-Bernabe, S.N. Srirama, and M. F. Zhani, "Research challenges in nextgen service orchestration," *Future Gener. Comput. Syst.*, vol. 90, pp. 20–38, 2019.

[16] P. Varga *et al.*, "Making system of systems interoperable—The core components of the arrowhead framework," *J. Netw. Comput. Appl.*, vol. 81, pp. 85–95, 2017.

[17] J. Halme *et al.*, "Monitoring of production processes and the condition of the production equipment through the internet," in *Proc. 6th Int. Conf. Control, Decis. Inf. Technol.*, 2019, pp. 1295–1300.

[18] D. Hästbacka *et al.*, "Dynamic and flexible data acquisition and data analytics system software architecture," in *Proc. IEEE Sensors*, 2019, pp. 1–4.

[19] K. Shin and H. Park, "Smart manufacturing systems engineering for designing smart product-quality monitoring system in the industry 4.0," in *Proc. 19th Int. Conf. Control, Automat. Syst.*, 2019, pp. 1693–1698.

[20] Z. Zhou, J. Hu, Q. Liu, P. Lou, J. Yan, and W. Li, "Fog computing-based cyber-physical machine tool system," *IEEE Access*, vol. 6, pp. 44 580–44 590, 2018.

[21] A. Xenakis, A. Karageorgos, E. Lallas, A. E. Chis, and H. Gonzalez-Velez, "Towards distributed IoT/cloud based fault detection and maintenance in industrial automation," *Procedia Comput. Sci.*, vol. 151, pp. 683–690, 2019.

[22] C. Hegedüs, P. Varga, and I. Moldován, "The mantis architecture for proactive maintenance," in *Proc. 5th Int. Conf. Control, Decis. Inf. Technol.*, 2018, pp. 719–724.

[23] J. Lee, H. D. Ardakani, S. Yang, and B. Bagheri, "Industrial big data analytics and cyber-physical systems for future maintenance & service innovation," *Procedia CIRP*, vol. 38, pp. 3–7, 2015.

[24] G. Di Orio, P. Maló, J. Barata, M. Albano, and L. L. Ferreira, "Towards a framework for interoperable and interconnected CPS-populated systems for proactive maintenance," in *Proc. IEEE 16th Int. Conf. Ind. Informat.*, 2018, pp. 146–151.

[25] J. Delsing, "Local cloud Internet of Things automation: Technology and business model features of distributed internet of things automation solutions," *IEEE Ind. Electron. Mag.*, vol. 11, no. 4, pp. 8–21, Dec. 2017.

[26] H. Derhamy, J. Eliasson, and J. Delsing, "System of system composition based on decentralized service-oriented architecture," *IEEE Syst. J.*, vol. 13, no. 4, pp. 3675–3686, Dec. 2019.

[27] H. Derhamy, M. Andersson, J. Eliasson, and J. Delsing, "Workflow management for edge driven manufacturing systems," in *Proc. IEEE Ind. Cyber-Phys. Syst.*, 2018, pp. 774–779.

[28] P. Bellavista, J. Berrocal, A. Corradi, S. K. Das, L. Foschini, and A. Zanni, "A survey on fog computing for the Internet of Things," *Pervasive Mob. Comput.*, vol. 52, pp. 71–99, 2019.

[29] M. Engelsberger and T. Greiner, "Dynamic management of cloud- and fog-based resources for cyber-physical production systems with a realistic validation architecture and results," in *Proc. IEEE Ind. Cyber-Phys. Syst.*, 2018, pp. 109–114.

[30] K. Cao, L. Li, Y. Cui, T. Wei, and S. Hu, "Exploring placement of heterogeneous edge servers for response time minimization in mobile edge-cloud computing," *IEEE Trans. Ind. Informat.*, vol. 17, no. 1, pp. 494–503, Jan. 2021.

[31] K. Matsui and H. Nishi, "Error correction method considering fog and edge computing environment," in *Proc. IEEE Ind. Cyber-Phys. Syst.*, 2019, pp. 517–521.

[32] C. Hegedus, P. Varga, and A. Frankó, "Secure and trusted inter-cloud communications in the arrowhead framework," in *Proc. IEEE Ind. Cyber-Phys. Syst.*, 2018, pp. 755–760.

[33] E. Palm, U. Bodin, and O. Schelén, "Kalix: A Java 11 library for developing eclipse arrowhead system-of-systems," in *Proc. 25th IEEE Emerg. Technol. Factory Automat.*, 2020, pp. 1389–1392.

David Hästbacka (Member, IEEE) received the M.Sc. (Tech.) and D.Sc. (Tech.) degrees in automation science and engineering from the Tampere University of Technology, Tampere, Finland, in 2007 and 2013, respectively.

He is an Assistant Professor with Tampere University, Tampere. His research interests include system and software architectures and interoperability of software systems in production and energy systems applications.

Jari Halme received the M.Sc. (Tech.) degree from the Lappeenranta University of Technology, Lappeenranta, Finland, in 1995.

He is a Senior Scientist with Operation and Maintenance team, VTT, Espoo, Finland. Since 1995, he has been employed by VTT having various project responsibilities related to maintenance, condition monitoring, and diagnosis of rotating machinery. He is currently responsible for predictive maintenance substance with VTT.

Laurentiu Barna received the M.Sc. and Dr. Tech. degrees in information technology from the Tampere University of Technology, Tampere, Finland, in 2003 and 2008, respectively.

From 2000 to 2008, he was a Researcher with the Signal Processing Laboratory, Tampere University of Technology. He is currently working as a Senior Project Manager with Wapice Oy, Vaasa, Finland, and coordinates Wapice's efforts in several EU R&D projects.

Henrikki Hoikka received the M.Sc. degree in automation engineering from Tampere University, Tampere, Finland, in 2019.

He is currently working with Metso Outotec, Tampere, as a Software Engineer. He works on Industrial Internet of Things and edge-computing-related projects in mining and aggregates industries.

Henri Pettinen received the M.Sc. degree in automation engineering from Tampere University, Tampere, Finland, in 2020.

His research interests include architectures for cyber-physical systems and edge computing.

Martin Larrañaga is currently working toward the double Master's degree in cloud and network infrastructures with Sorbonne Université, Paris, France, and the University of Trento, Trento, Italy.

He is a Telecommunications Systems Engineer with VTT, Espoo, Finland.

Mikael Björkbom received the D.Sc. (Tech.) degree from Aalto University, Espoo, Finland, in 2010.

He was a Postdoctoral Researcher and the Research Coordinator with the Wireless Sensor Systems Group, Aalto University, until 2014. Currently he works as a Senior Chief Engineer with Konecranes. His main research interests include adaptive control and simulation of wireless control systems, computing, and machine learning at the edge.

Heikki Mesiä received the M.Sc. (Eng.) degree from the Tampere University of Technology, Tampere, Finland, in 1984.

He joined Konecranes, Hyvinkää, Finland, in 2008, where he currently works with Konecranes Research and Innovation as a Senior Research Specialist for crane intelligence. He has previously worked with VTI Technologies, expert in microelectromechanical systems technology-based acceleration sensors, and with Vaisala in the global humidity sensor business holding several positions from product design to product line management.

Antti Jaatinen (Member, IEEE) received the M.Sc. (Tech.) degree in automation and control from the Tampere University of Technology, Tampere, Finland, in 2003.

He is a Digital Development Manager with the Digital Platforms Department, Metso Outotec, Tampere. He has been working with R&D of automation and connectivity for industrial machines since 2008.

Marko Elo received the M.Sc. degree from the Tampere University of Technology, Tampere, Finland, in 1993.

He is currently an R&D Manager with CrossControl Oy, Tampere. He has a special interest in technology partnerships and cocreation. His research interests include applied research and architecture design of industrial IT systems in the segment of mobile vehicles, with focus on human–machine interfaces, edge computing, artificial intelligence/machine learning, and connectivity.