Evaluating Arrowhead Framework for Dynamic Condition Monitoring Applications and Edge Computing in Mining

Henrikki Hoikka, Antti Jaatinen
Metso Minerals
Tampere, Finland
{henrikki.hoikka, antti.jaatinen}@metso.com

David Hästbacka
Tampere University
Tampere, Finland
david.hastbacka@tuni.fi

***Abstract*—Condition monitoring and predictive maintenance of the equipment is an important topic of production environments, and many equipment manufacturers offer data services and build service business upon them. To further advance Metso's capabilities, the configuration on how the data collected at the field, and how it is moved to the cloud for further processing and analysis, needs more advanced solutions. Typical equipment manufacturer's customers from all over the world have numerous pieces of equipment on their sites. In this paper Arrowhead Framework and dynamic service discovery provided by it, are evaluated as a potential technology that could help machine and equipment providers to achieve better configurability of its data collection devices at the edge of the network.**

***Index Terms*—measurement system, data analytics, cyber-physical systems, dynamic orchestration, system architecture, system requirements, edge computing, arrowhead framework**

I. Introduction

Industrial internet builds upon utilization of data, advanced analytics and more informed decisions as its outcome in all imaginable sectors. This is also the case for manufacturing systems and production, Industry4.0, where a wide range of communication means will be used to collect industrial data and advanced functionality will be orchestrated across and between public clouds and local premises [1]. Industrial cyber-physical systems (CPS) are expected to enable the transformation to adaptive, networked and knowledge-based industry but there are key challenges in their integration and collaboration of industrial CPSs across ecosystems [2]. As these systems are getting more complex and also form systems of systems (SoS), more information is available across different domains and interoperability of both data and system functions becomes a key concern.

Industrial Internet of Things (IIoT) promises to solve several of the data acquisition and remote control challenges faced by the industry. There are, however, a lot of use cases where it is not feasible or still too expensive to transfer huge amounts of data to the cloud. In some cases data ownership or privacy concerns can also limit this. Edge computing, on the other hand, envisions cloud services and resources close to the user applications and benefits are expected in areas where vast amounts of data needs to be processed, near real-time performance is needed or where security is preferred [3]. Modern software based applications would benefit of being able to utilize resources on cloud as well as on the edge while using same system composition and configuration methods as well as security mechanisms.

Metso Minerals manufactures equipment for the mining and aggregates industries. In this paper this domain serves as an example use case where data sources and sensors need to be connected, data flows configured to analytics components on different levels of the process, and new edge components configured and managed in a scalable SoS. One key objective of open platforms and systems is to reduce dependency of big public cloud and industrial internet solution vendors, and instead focus towards open ecosystems.

In this paper a condition monitoring application is designed and implemented focusing on the use of the Arrowhead Framework (AHF) in composition and management of services. The research question proposed is how AHF supports configuration of data flows and management of installations ranging from edge to cloud. Its main contributions are 1) how to build such a system using AHF and 2) how AHF is suited and supports development of these kinds of applications. The main results presented in this paper are based on the outcomes of a Master's thesis [4] which further details and explains the implementation used to verify the results presented.

Following, section II presents related work for production environments and condition monitoring. In section III the example use case is explained along with background of tools used in the implemented solution that is presented in section IV. Results and discussion is provided in section V before concluding the paper in section VI.

II. Related Work

System and software architecture for maintenance services and condition monitoring applications for CPS have been presented in [5]–[9], among others. Condition monitoring based on edge computing, has been studied by, for example, [10] for distribution transformers and [11] for vibration monitoring applications.

Previously, a Maintenance4.0 framework was described in [12] and a concept for integrating condition monitoring to maintenance operations was introduced in [13].

Monitoring of production assets over the internet, also related to this paper, has been discussed by [14]. A general data acquisition and data analytics system software architecture has been specified in [15]. In the concept, AHF is proposed to orchestrate and manage the computational services and information flows.

III. Example use case and background technology

A. Example environment and test setup

The test system used for the experiments is a condition monitoring system for vibrating screens that is being developed in Metso. It comprises a data collection computer and several accelerometers mounted on a screen, providing data on its motion and vibration levels. The sensors are self-powered with energy harvesters for continuous operation when the screen is running. Several Key Performance Indicators (KPI) are calculated locally from these data and they need to be sent to the cloud for analysis and visualisation. The KPI's are presented in a PostgreSQL database that the AHF is able to read from using PostgREST. The system does measurements and updates the KPI's every minute.

B. Arrowhead Framework

The core idea of the evaluated framework, AHF, [16], [17] is introduced in figure 1. The framework has three mandatory core-systems; the Orchestrator, the Service Registry and the Authorization. These three form a System-of-Systems (SoS) by offering the means for so-called application systems, on service-registration, service-discovery and service-authorization. The SoS residing under one core is commonly referred as the Arrowhead local cloud or just local cloud, this terminology is used also in this paper. The roles of core systems are specified in more detail below:

- Service Registry - Keeps book on available services, and offers means for registering and querying them. The service provider registers it's services to the register, when the service are available and unregisters them when the services become unavailable.
- Ochestrator - The main component within the core systems. The consumer sends the service discovery requests to the Orchestrator, which, handles the hassle of deciding what service candidates should be sent as a response. Orchestrator has two modes 1) static so-called orchestration-store mode, where the consumers requesting a service are responded according to predefined rules and availability of the service in the registry. 2) dynamic mode, where the responses do not follow the rules and a list of available services are sent. This list can be reduced with restricting parameters in a requests body, including, for example, metadata key-value pairs and a desired provider.
- Authorization - Keeps book on authorization rules. Before responding to the consumer, the Orchestrator makes sure that the consumer is allowed to use the service. In so called secure-mode, in which the communication happens with HTTPS, the authorization system also generates access-tokens for the consumers, which are needed when a service is consumed. This prevents unauthorized access outside the local cloud, which is possible in so-called insecure-mode.

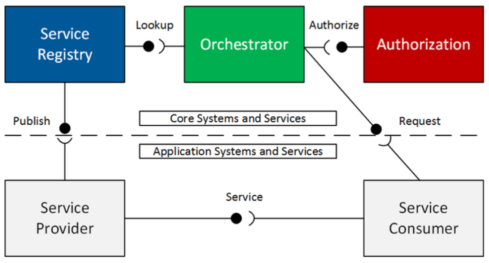

Fig. 1. The core idea of Arrowhead Framework [16].

Additionally, the core-systems have a dependency on a MySQL database, which is used to store the information on systems, services, neighbouring local clouds, authorization rules and other important bookkeeping needed by the core-systems.

1) Arrowhead Framework - supporting core-systems: On top of the core functionality offered by the core-systems, AHF also provides, so-called, supporting core-systems, which extend the local clouds capabilities. Multiple supporting core systems have been developed, such as for onboarding with device and system registries for a chain of trust from hardware [18], managing Quality of Service (QoS) [19], but in this paper the most relevant are the Gatekeeper and the Gateway systems, which co-operate with Gateway and Gatekeeper systems in another local cloud to enable so-called inter-cloud service discovery between them [20], [21].

The so-called inter-cloud negotiations between the Gatekeepers happen in HTTP [1], in successful negotiations a tunnel that is used by the application systems is formed. The Gateway systems, on the other hand, are responsible on offering a proxy service to this tunnel. All of this happens in the background and the orchestrator system is the one that initiates the negotiation process, either because service was not found in the local cloud, or the global discovery was specifically asked by the consumer. After the tunnel has been formed the orchestrator sends a response to the consumer where information needed to communicate via the gateway proxy are specified. This includes, for example, a reserved port and the address of the local Gateway system. The tunnel between the Gateways uses an AMQP broker, while the application system uses HTTP to communicate with the Gateway.

2) Supporting stack of Arrowhead local cloud: In the demo-application, each Arrowhead local cloud consists of Arrowhead-core, Arrowhead Gatekeeper system, Arrowhead Gateway system and application systems that implement the

[1]The version 4.1.3 of the framework moves also the inter-cloud negotiations to the broker. The demo application was developed with version 4.1.2

business logic of the application, which are covered in more detail later in section IV. On top of this the demo application also includes a PostgreSQL database and a PostgREST server, which turns the schema of the database automatically to REST services. The local cloud instance running on the cloud, also has a RabbitMQ AMQP broker, used by the Gateway systems.

Since AHF does not provide means for deployment, in a sense of actually getting the executable running on the target platform, each artefact enlisted above was containerized, or if possible, a readily available container was used. Docker and Docker-compose were used to deploy the containers.

IV. Implementation

The architecture of the demo application and the roles of various tools presented in section III-B is illustrated in figure 2. The demo application tries to tackle the problems faced in configuration of the edge computers. The main goal is to evaluate AHF's capability as a provider of dynamic service discovery. To help the installation required in the field and commissioning of new equipment to the customer, ideally, plug-n-play style of installation would be available, and afterwards, control on the configuration of the devices, ideally, as a fleet, would be easier to handle. Thus, dynamic service discovery, through which the data sinks and configuration sources could be discovered when new equipment comes online, offers an evaluation-worthy concept. The actual condition monitoring, is left outside the scope of the demo application and the scope is limited to mitigate the problems in the movement of the condition monitoring data of which, is produced on the edge computer.

A. The roles of application systems and services provided by them

The business logic of the demo application is implemented with four application systems presented in figure 3. The application systems are divided in two groups; data-path and control-path. Both groups have two application systems, one running on the cloud and one running at the edge. The data-path group is responsible of pushing the data from edge to the cloud, and the control-path group is responsible of configuring the interval of pushes and the subset of available data-points that are pushed. The configuration files that are sent trough control-path, can also contain arbitrary configuration objects that can be used, for example, for configuring the edge computer that the Arrowhead local cloud is running on. The services provided by the application systems and their roles are the following:

- Batch - Consumer uses the service to push batches of condition monitoring data according to the configuration. Configuration stored in the DB must be valid, each provider can have their own configuration, and the providers must be discoverable through AHF.
- CMD - Consumer uses the service to push [2] the configuration files to the edge computers, to query the currently available data-points, and the current configuration of an particular edge computer. This service has no consumer systems implemented in demo-setup, but application systems that try to drive the configuration at the edge to a certain state, could be developed against this service.
- CtrlPoll - Consumer uses the service to poll the provider in-case of a need for reconfiguration has emerged, the providers must be discoverable through AHF.
- CtrlUpdate - Consumer uses the service to notify the providers of the service on a change on the configuration or the set of available data-points, the providers must be discoverable through AHF.

[2]The current implementation caches the configurations, for a later fetch via Ctrlpoll service.

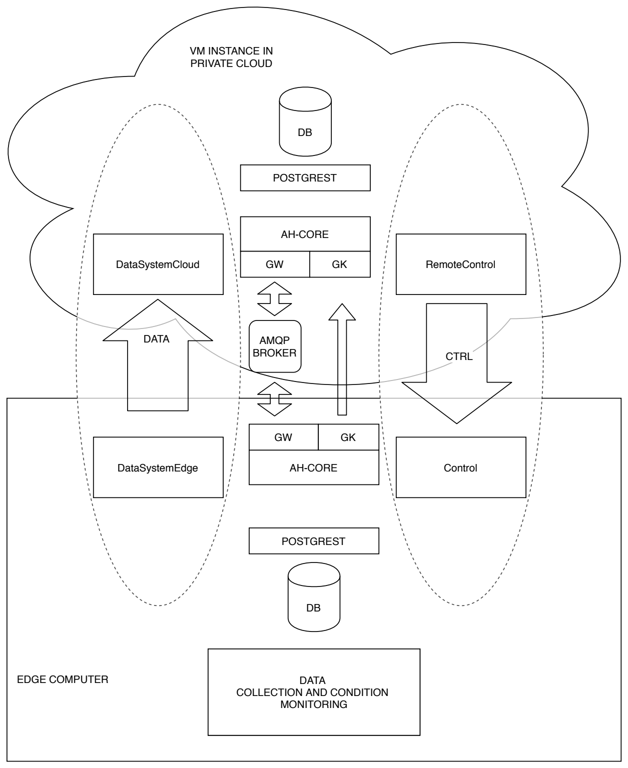

Fig. 2. Architecture of the demo application, the application systems form data and control paths marked with dotted ellipses.

Service discovery functionality of AHF is heavily utilized. Only services that are not discovered through AHF, are the REST services allowing access to the database offered by PostgREST. The reasoning behind not discovering these services through AHF as well, is that the location of the database is assumed to be always known, and therefore, there is no point in rediscovering these services. In other words, a local cloud instance in the context of the demo application is assumed to always have a database running "right next to it" on the same physical or virtual machine, available through same port and network interface.

Since the version 4.1.2 of the framework uses HTTP in inter-cloud negotiations happening between Gatekeeper systems, firewalls limit the initiation only to application systems residing at the edge. In other words, all the service providers that are providing a service that is meant to be consumed via the AHF Gateway, through edge - cloud boundary, are running on the cloud, this is the reason behind the polling scheme that the services take.

B. Support for multiple local cloud instances at the edge and on the cloud

The service providers at the cloud level can serve as a data sink or control source for multiple instances of data and control application systems residing at the edge. The data-system at the edge is also capable of pushing data to multiple sinks, which can reside either on the cloud or at the edge. This enables collaborative setups where multiple individual pieces of equipment possibly under the control of different stakeholders can share data, based on their interests and permissions. For example, in case of mining, various OEM's would be able to share data to improve the overall performance of the customers process.

An example setup with multiple local clouds at both the edge and on the cloud is presented in figure 4. The application systems in the demo-application have support for this kind of large-scale setup. However, the implementation and design decisions taken in AHF cause, wide multi local cloud setups to be hard to maintain. The problems are covered in more detail in the following section.

V. Results and Discussion

On a conceptual level, the SoS approach taken by AHF offers a potential way of implementing applications in the industry and it offers a clear mental model on how one would develop IoT applications in general. The way how the Gatekeeper and Gateway systems can be used to join local clouds could, at least in theory, enable large System-of-Systems to be deployed. However, there are problems, that make the AHF only suitable for small scale test setups, like the one that was used in the presented demo application.

A. Problems with the Current Authorization Scheme

In table I the system table of MySQL database, which is internally used as AHF's main means of persistent storage, is

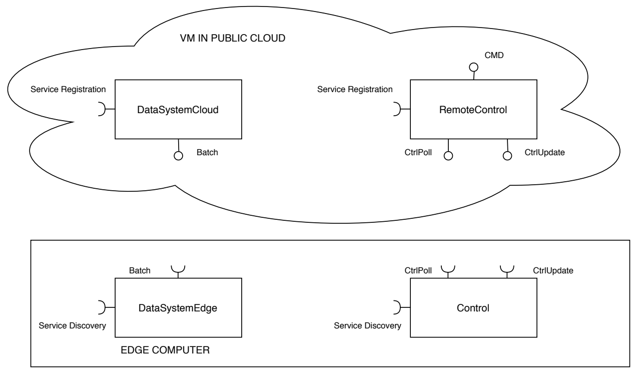

Fig. 3. Application systems and the services provided and consumed by them.

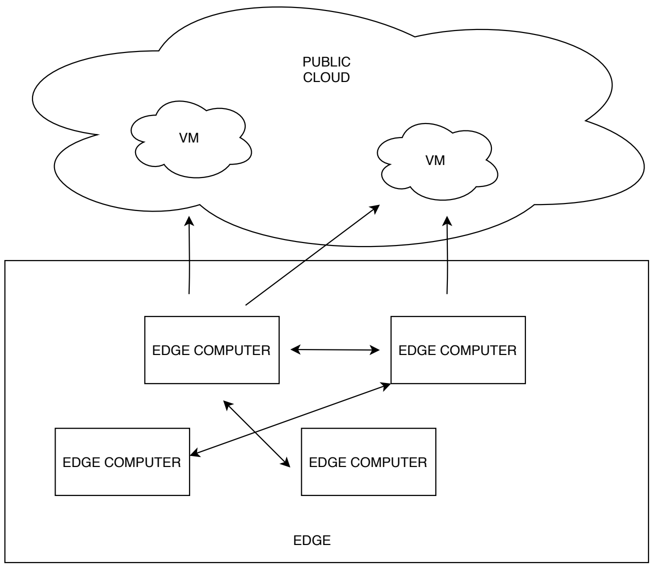

Fig. 4. An example setup with multiple local cloud instances, the arrows point towards the direction where service consumption is possible. The edge computers are assumed to be in a same LAN.

presented. The table contains information on what application systems are present on the local cloud. Port and Address are both mandatory fields, rest of the fields are voluntary, however, in practice the name of the system is always needed and in "secure-mode" where the authorization system generates the access tokens for the consumer systems, the auth_info field is also in practice mandatory.

TABLE I
System table

id	address	auth_info	port	system_name
1	0.0.0.0	-	222	foo
2	0.0.0.0	-	2222	bar

In turn, table II presents the local authorization rules in the same database. Local authorization table's columns "consumer_system_id" and "provider_system_id" refer to rows in the system table I. The last column specifies the service that the consumer system is authorized to consume. As can be seen, the system table does not make any difference between the consumer and provider roles, which one system can have both of. This leads to weird, probably unintentional requirements placed on the application systems.

Firstly, in HTTP the client, or in our case the consumer, does not have a port in a way that is assumed. However, since the port column on the system table I is mandatory, the user has to come up with a bogus port, to get his consumer system inserted in the system table so he can make an authorization rule, given that particular system does not also have a role as a provider, in which case the service registration process creates a row in the system table, the port however remains bogus also in this case.

Secondly, the user has to manually wire the authorization rules, in a similar way to orchestration rules in the static consumer-store scheme. This takes a toll on dynamism, which is basically lost, since the requirement in practice means that every time a new application system is introduced, or the authorization requirements are changed, the user has to alter the tables in a way where the most coarsely grained units are an application system and service, instead of a group of systems and services or something else more indirect. This makes the dynamic service discovery unscalable to larger setups, consisting of numerous application-systems and services. Similar problem also exists in the static orchestration-store scheme, which requires similar granularity on the rules that user sets to the database, i.e consumer-system, provider-system and the provided service, are needed for each rule. If multiple identical rules, for example, for various consumer systems with same provider system, are needed, the wiring must be done individually for each ”connection”.

TABLE II
LOCAL AUTHORIZATION TABLE

id	consumer_system_id	provider_system_id	service_id
1	1	2	4
2	2	1	5

TABLE III
GLOBAL AUTHORIZATION TABLE

id	consumer_cloud_id	provider_cloud_id
1	3	4
2	4	3

However, if the global authorization scheme is used, where the consuming system is in a different Arrowhead local cloud, the authorization works differently. In table III the global authorization table is presented. As can be seen, the table refers to rows in a table where all of the known local clouds are presented, in a similar manner like the systems are presented in the system table. The global authorization therefore uses the group of application systems that are residing in a particular local cloud as it's main unit. This means that all the consumer systems can consume any service provided by application systems in the authorized local cloud. Therefore in this case, there is grouping.

System level authorization however is missing from the global authorization, which is problematic, since it is possible that all of the services provided in a certain local cloud are not for all consumers in another local cloud. Also, since every local cloud has it's own instance of MySQL used for book-keeping, each of them needs to be configured separately. Since, even in the case of global-authorization, the table holding the information on known local clouds needs to be filled or altered before authorization rules can be issued. In large setups this will become tedious, and automating the configuration might be hard.

B. Problems with the registration of services

As the service provider registers its services to the service registry, it can not register the possible sub resources of the service's REST URI, with the same push. This means that if the system wants the sub-resources to be discoverable through AHF, it needs to register them separately. In case of local consumption, the authorization scheme, that forces the user to specify the authorization in a way that was described above, means that the user has to do lots of manual labor to make the discovery happen. This is because each sub-resource needs its own authorization rule, one per consumer.

In some cases where the REST service has a lots of sub-resources this leads to an unbearable situation, where the best option is to assume that consumer knows in advance what sub-resources a certain service has. Declarative API documentation schemes like Open-API [3] could help. However, if the authorization is needed, it can not be achieved if the service has sub-resources outside AHF's knowledge, i.e the sub-resources need to use the access-token of the parent resource. This adds pressure to design, since things that would be a natural fit as a sub-resource can not be authorized with means provided by AHF, if the sub-resources have different sets of acceptable consumers.

C. Other Problems

Serious thought must also be placed on the overall architecture decisions taken. One may ask, if the core-systems, Orchestrator, Authorization and Service Registry are always mandatory in a local cloud, why would they be separated in three independent systems? If the three systems were combined to one, the user would have way more easier job to manage the System-of-Systems as a whole, especially since the co-operation between the core systems is not the most robust one, for example, external start-up scripts with sleep statements that make sure that the systems are started in a ”correct order” are used.

VI. CONCLUSION

Metso's role in the Productive4.0 project is to provide an industrially relevant application use case for AHF orchestrated and managed service composition. That means not only looking at the specifications but looking at the system and software composition as a whole. It is obvious that more commercialisation is needed as the components, documentation, delivery and deployment systems etc. are not yet on a mature enough level. There is a steep learning curve, and given the usefulness of the framework and the things that you can do with it, the curve is currently not a particularly rewarding one either.

AHF provides a number of necessary features that any SoS service composition would need to implement. From an interoperability point of view it is essential to promote standard and acknowledge methods in contrast to many vendor or tools specific methods, e.g. mechanisms for service composition and

[3]https://swagger.io/specification/

configuration or uniform authorisation and access management in dynamic and evolving systems.

Considering the industrial application cases with Metso's equipment, almost all deal with having to get the data out from a control system in the field and moving it into a central storage. The general problem of data collection from end devices to the cloud still remains in industry and still requires a surprising amount of work. In our testing, AHF in its current form did not meet the requirements of a mature framework usable in an industrial application, in a sense, where it could be seriously considered as a platform for Metso's condition monitoring applications. The dynamism is lacking, because of the static authorization scheme and heavy need for configuration, and despite the promising SoS model, the scalability needed for a OEM like Metso, with 1000's of pieces of equipment of various types out in the field on customers sites, is not yet there.

Acknowledgment

This work is part of the project Productive4.0 and has received funding from the EU ECSEL Joint Undertaking under grant agreement no. 737459 and from the national funding authority Business Finland.

References

[1] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0," *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 17–27, March 2017.

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

[3] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, Aug. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18319903

[4] H. Hoikka, "Evaluation of Arrowhead Framework in Condition Monitoring Application," Master's thesis, Tampere University, Finland, 2019.

[5] C. Hegedüs, P. Varga, and I. Moldovan, "The mantis architecture for proactive maintenance," in *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)*, April 2018, pp. 719–724.

[6] 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, proceedings of the 4th International Conference on Through-life Engineering Services.

[7] 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 *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, July 2018, pp. 146–151.

[8] G. Wang, M. Nixon, and M. Boudreaux, "Toward cloud-assisted industrial iot platform for large-scale continuous condition monitoring," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1193–1205, June 2019.

[9] H. Fleischmann, J. Kohl, and J. Franke, "A reference architecture for the development of socio-cyber-physical condition monitoring systems," in *2016 11th System of Systems Engineering Conference (SoSE)*, June 2016, pp. 1–6.

[10] L. Thangiah, C. Ramanathan, and L. S. Chodisetty, "Distribution transformer condition monitoring based on edge intelligence for industrial iot," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, April 2019, pp. 733–736.

[11] H. Liu, Y. Wang, F. Li, X. Wang, C. Liu, and M. G. Pecht, "Perceptual vibration hashing by sub-band coding: An edge computing method for condition monitoring," *IEEE Access*, vol. 7, pp. 129 644–129 658, 2019.

[12] E. Jantunen, G. Di Orio, F. Larrinaga, M. Becker, M. Albano, and P. Malo, *A Framework for Maintenance 4.0.* Institute of Mathematics and its Applications, 6 2018.

[13] D. Hästbacka, E. Jantunen, M. Karaila, and L. Barna, "Service-based condition monitoring for cloud-enabled maintenance operations," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct 2016, pp. 5289–5295.

[14] J. Halme, E. Jantunen, D. Hästbacka, C. Hegedűs, P. Varga, M. Björkbom, H. Mesiä, R. More, A. Jaatinen, L. Barna, P. Tuominen, H. Pettinen, M. Elo, and M. Larrañaga, "Monitoring of production processes and the condition of the production equipment through the internet," in *6th International Conference on Control, Decision and Information Technologies, CoDIT'19 ; Conference date: 23-04-2019 Through 26-04-2019*, 2019.

[15] D. Hästbacka, J. Halme, M. Larrañaga, R. More, H. Mesiä, M. Björkbom, L. Barna, H. Pettinen, M. Elo, A. Jaatinen, and H. Hoikka, "Dynamic and flexible data acquisition and data analytics system software architecture," in *2019 IEEE SENSORS*, Oct 2019, pp. 1–4.

[16] Arrowhead. Project github-repository and documentation. Accessed: 2020-02-10. [Online]. Available: https://github.com/arrowhead-f

[17] J. Delsing, *Iot automation: Arrowhead framework.* CRC Press, 2017.

[18] A. Bicaku, S. Maksuti, C. Hegedűs, M. Tauber, J. Delsing, and J. Eliasson, "Interacting with the arrowhead local cloud: On-boarding procedure," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, May 2018, pp. 743–748.

[19] M. Albano, P. M. Barbosa, J. Silva, R. Duarte, L. L. Ferreira, and J. Delsing, "Quality of service on the arrowhead framework," in *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, May 2017, pp. 1–8.

[20] C. Hegedus, D. Kozma, G. Soos, and P. Varga, "Enhancements of the arrowhead framework to refine inter-cloud service interactions," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct 2016, pp. 5259–5264.

[21] C. Hegedus, P. Varga, and A. Frankó, "Secure and trusted inter-cloud communications in the arrowhead framework," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, May 2018, pp. 755–760.