Leveraging Eclipse IoT in the Arrowhead Framework

Lukas Römer
Bosch.IO GmbH
Berlin, Germany
lukas.roemer@bosch.io

Sven Erik Jeroschewski
Bosch.IO GmbH
Berlin, Germany
svenerik.jeroschewski@bosch.io

Johannes Kristan
Bosch.IO GmbH
Berlin, Germany
johannes.kristan@bosch.io

***Abstract*—The Arrowhead Framework is a holistic framework for industrial automation in a service-oriented manner. It provides means to orchestrate systems and services within local and remote clouds. Eclipse IoT (Internet of Things) provides three stacks of software and tools, which allow the creation of IoT applications for (i) constrained devices, (ii) gateways and (iii) cloud backends. Several of the Eclipse IoT projects appear to be useful also within the Arrowhead Framework. This paper identifies possible candidates for the integration of Eclipse IoT technologies into the Arrowhead Framework, discusses for one example (Eclipse Hono) how such integration could look like, and gives an outlook for the next candidates to be integrated.**

***Index Terms*—IOT, IIOT, Eclipse, Arrowhead, Device Hub, Update Management**

I. Introduction

With increasing computing power, storage capacity and simultaneously decreasing chip sizes and power consumption, more embedded devices than ever execute intelligent tasks and share the results. This development, more recently known as the Internet of Things (IoT), enables a variety of applications ranging from smart cities through interconnected cars to more intelligent households. However, not only the consumer market is evolving, but also the industrial sector adopts more efficient IoT methods in production and logistics. Currently, many vendors for these industrial IoT (IIoT) solutions develop proprietary systems to address various challenges such as the management of manufacturing devices or monitoring and controlling industrial processes.

The Arrowhead Framework [1] attempts to offer a generic and comprehensive framework addressing universal challenges on the path to a successful IIoT infrastructure. The Arrowhead approach is based on the concept of Service-Oriented Architecture (SOA) and provides several standardized modules and interaction patterns. However, as there already exist powerful and proven open source IoT libraries and frameworks, a technical implementation of the concepts in Arrowhead does not need to start all over. Instead, one can try to reuse as much as possible since integrating with existing technologies and projects bears chances of synergy effects to save efforts, time and money. As part of this work, we illustrate this by showing a number of projects that could be used as part of an Arrowhead deployment.

The landscape of existing open source projects in the IoT domain has become fairly large and heterogeneous. Therefore, the process of choosing relevant projects and defining valid criteria for that choice is an extensive task in itself. Hence, we decided to focus this work on the results from the Eclipse IoT working group which is one of the largest communities in the IoT domain and part of the Eclipse Foundation [2]. In the following, we analyze and show how technologies developed in the context of the Eclipse IoT working group can be used to either complement existing Arrowhead core systems or create new Arrowhead core systems. An analysis for the contrary direction, namely to integrate components of the Arrowhead Framework to Eclipse IoT technologies, is beyond the scope of this work. Moreover, the Eclipse IoT working group does not build one homogeneous framework and does not follow one single architecture, but instead creates a collection of heterogeneous software projects around aspects of IoT use cases.

II. Arrowhead and Eclipse IoT

As mentioned above, the Arrowhead Framework evolves around the concept of SOA. Within the Arrowhead Framework each physical computation resource can be referred to as a device. Each device is capable of running multiple software stacks that each are referred to as a system. Further, each of these systems provides a set of services that can either be used to interact with the system or that interact with another system through its service on behalf of the initiating system. In Arrowhead the different systems can be encapsulated in so called ”local clouds”. Often the services in one local cloud can reach each other over the same local network. One motivation of introducing local clouds is to cope with the complexity of larger systems of systems.

The base of every Arrowhead local cloud is the set of three so-called mandatory core systems which are the authorization system, the service registry system and the orchestration system. Those systems need to be present in each local cloud. For a basic service interaction, a service A, that wants to initiate a service exchange, would send a request to the orchestration system to get the address of a service B that can process the request. To answer the orchestration request, the orchestration system asks the service registry system for all known services matching the request. Then the orchestration system checks with the help of the authorization system for which matching services the requesting service has the authorization to use them. In the last step, the orchestration system then returns the information for a service that matches the request to service A which then can engage in the actual service interaction. For a more detailed description of the mandatory core, systems refer to [1].

Besides the described mandatory core systems, there are already some existing core services that do not provide their application-specific logic but can be used to improve and ease the service interaction and orchestration. Examples for such core services are a translation system to translate between different protocols [3], a plant description system [4] or a system to enable the communication between different local clouds [5].

As already introduced above, one large community working on topics and projects that are relevant in the context of the Arrowhead Framework is the Eclipse IoT working group. The Eclipse Foundation provides infrastructure, processes and further services such as marketing support for open source projects that are developed under its umbrella. The foundation itself is neutral regarding the developed technologies and the domains addressed by those projects shaping the actual developments. According to the charter of the IoT working group, its members aim to ”encourage, develop, and promote open source solutions that can be used to overcome market inhibitors found in most IoT ecosystems” [6]. The members of the working group are mostly private companies from the IT and industrial domains such as the strategic members Eurotech, RedHat, and Bosch.IO. Among the other members are currently IBM, Nokia, Siemens, and DB Systel [7]. Some of these companies also provide commercial offerings on top of Eclipse IoT software implementations [8].

Since the topics addressed in the Eclipse IoT working group evolve around different aspects of the actual solutions, the members came up with differentiation into three hierarchical layers as described in [9] and depicted in Fig. 1 to also provide a grouping of their projects. In the bottom layer are constrained sensors and actuators that measure data and execute commands but do not process or evaluate information due to low computing power. These devices interact with gateways at the edge, which build an intermediate layer to the IoT cloud platforms. The cloud platform or back-end enables high scalability, fast processing of large data volumes, data aggregation, and long-term analysis. Currently, approximately 35 different open source projects form the working group.

This paper evaluates to which extent these existing technologies do harmonize with the concepts of the Arrowhead Framework in an industrial IoT context. As most Eclipse IoT projects are not primarily designed for industrial use cases but more for general applicability, not all projects are equally suited to be used within the Arrowhead Framework. However, the Eclipse IoT working group identified its projects that are ready for potential Industry4.0 application and published a white paper on that [2]. Table I lists the identified projects and provides a short description of their functionality.

The concept of local clouds in the Arrowhead Framework leads to a certain degree of decentralization. At the same time,

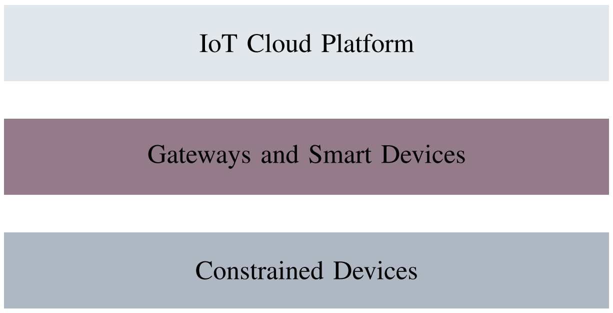

Fig. 1: Eclipse IoT Software Stack

Project	Category	Layer
Eclipse Milo	Data Aggregation	
Eclipse Mosquitto	Data Aggregation	Cloud
Eclipse Paho	Data Aggregation	Devices
Eclipse Unide	Data Aggregation	
Eclipse OM2M	Data Aggregation	
Eclipse 4diac	Data Aggregation	
Eclipse Kura	Data Aggregation, Security, Digital Twin	Gateways
Eclipse Leshan	Security, Device Management	Gateways, Cloud
Eclipse Keti	Security	
Eclipse Wakaama	Device Management	Devices
Eclipse Kapua	Device Management, Event Management and Data Analysis	Cloud
Eclipse hawkBit	Device Management	Cloud
Eclipse Hono	Event Management and Data Analysis	Cloud
Eclipse Ditto	Digital Twin	Cloud
Eclipse Vorto		Cloud

TABLE I: Eclipse IoT projects suitable for industrial use cases

many projects of the Eclipse IoT working group, especially from the IoT cloud platform layer, are mostly designed and used for more centralized deployments where a cloud instance processes the data from the devices. However, it is also possible to change the deployment to multiple local clouds and have a more ”on-premise” styled deployment model. Supporting this is the Arrowhead’s SOA-structure which allows simple replacement and integration of new modules and is well suited for incorporating existing software components.

The following provides a short evaluation of each eligible Eclipse IoT project and its usability within one or multiple Arrowhead systems. which is also depicted in figure 2. Almost none of the projects can be used without adaption or without at least combining them with other modules. However, many of them could potentially build the foundation of an Arrowhead component or at least function as a structural archetype.

A. Eclipse Ditto

Eclipse Ditto provides so-called digital twins [10]. In the context of Eclipse Ditto, a digital twin is a pattern where things from an IoT context, such as sensors or machines, are represented in the digital world. Each representation has its attributes, such as IDs or names, and features like a sensor value. Based on that Ditto can act as a single point of truth

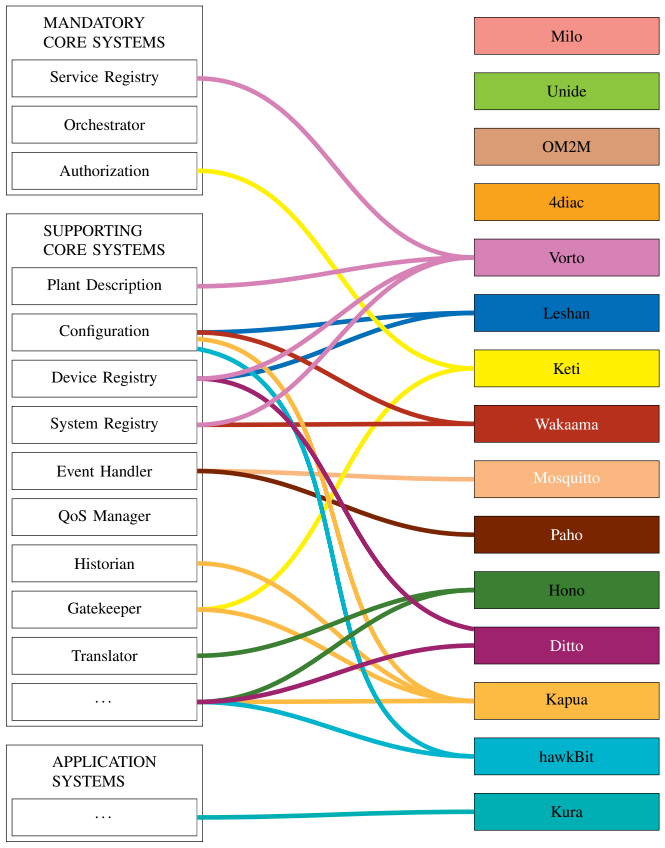

Fig. 2: Comparison of Arrowhead systems and eligible Eclipse IoT components. A connection indicates a potential application of an Eclipse IoT project in one of the generic Arrowhead systems.

for a given device or entity. Ditto further organizes access to these digital twins, provides a safe and multi-protocol-capable API to allow integration with other back-end infrastructure, and features individual access policies. Another aspect is that one does not always need to communicate directly with the device because Ditto can store its current state. This is a benefit, especially when one needs to interact with resource-constrained devices with limited communication or battery capabilities.

Within the Arrowhead context, Ditto can be part of a device registry. However, this device registry would not just be a repository for all existing devices, as the Arrowhead Framework declares it (compare [1]), but even more, handle changing device states. One could even argue that such an implementation exceeds the scope of a device registry in the Arrowhead context. Therefore, it seems more sensible to introduce a new digital twin core system. Besides acting as a registry for devices, this system would also store the current device state and make it available at any time. Moreover, Ditto integrates with Eclipse Hono, making it a good bundle in serving even more different protocols for the applications and devices.

B. *Eclipse Vorto*

Eclipse Vorto is a domain-specific language (DSL) inspired by Java. [11]. It offers a generic possibility to describe the characteristics of a hardware device, such as name or size and attributes like temperature or location, but also its functionalities. The Vorto community further provides a repository to share generic building blocks to reuse for new device definitions. Based on those definitions, Vorto allows the creation of code generators to generate code stubs for the device integration with various services and platforms.

One could use some of that functionality for building a device-, system-, or service-registry comparable to the approach described in [5] and [1]. Moreover, it possible to design a plant description model using a Vorto definition. The Arrowhead plant description system provides a basic overview of a plant's layout [4]. As Vorto also offers a meta-model to describe the relationship between different modules and their dependencies, Vorto could be an alternative to other modeling techniques.

C. *Eclipse Leshan & Wakaama*

Eclipse Leshan comprises a Java server and client implementation of the Lightweight M2M (LWM2M) protocol. The LWM2M protocol [12] developed by the Open Mobile Alliance (OMA) offers a standardized way to manage and operate a variety of low power hardware devices. LWM2M is often used on top of the Constrained Application Protocol (CoAP) [13] and features a REST-like interface. Hence, Leshan could complement an Arrowhead's configuration system introduced in [14]. It enables several management processes such as lock & wipe, device configuration, firmware updates, diagnostics, and data reporting [15].

A device-side counterpart to Leshan is Wakaama, which is compatible with every POSIX-compliant operating system [16].

D. *Eclipse hawkBit*

Eclipse hawkBit is a device management solution focusing on software roll-outs [17]. With Eclipse hawkBit it becomes possible to manage different software versions on IoT devices and define and monitor campaigns on when to perform software updates. The application comprises the update server, a management UI, and a set of APIs. The first API exposes a REST-based management interface for third-party applications. The second API provides a direct device integration via REST and polling, and the third API offers device integration via AMQP 1.0. This latter endpoint enables the incorporation of intermediate applications to feature different protocols, such as OMA-DM, LWM2M, or proprietary ones. Eclipse hawkBits' software rollout process allows sophisticated configuration options like as the grouping of devices, cascading deployment, and fine-grained monitoring. On a shop floor, it is required to run software updates on the various systems in a reliable and fast way as any production outages can yield severe losses. One option is to place Eclipse hawkBit as an update server that orchestrates the updates in a local cloud.

The update then can happen on system, service, application or configuration level.

Hence, Eclipse hawkBit appears to be a potential candidate for an Arrowhead configuration system implementation. Configuration and update procedures are highly critical, as they could potentially compromise a multitude of devices by implanting malicious code. This is another motivation to rely on already present, open, and proven technologies for this task. However, one open action point is the integration of the existing hawkBit authorization measures and the current authorization approach in the Arrowhead Framework.

E. Eclipse Keti

Eclipse Keti is an Attribute Based Access Control (ABAC) system that provides fine-grained authorization to an existing RESTful API [18]. The application allows the definition of complex access policies using XACML-based rules. Eclipse Keti offers authorization flows in compliance with the OAuth 2.0 specification [19]. Keti could serve as a starting point for an authorization system or the authorization aspect of a gatekeeper system. However, other Open Source projects in the Authentication and Authorization domain, such as Keycloak, have seen a wide adoption also within the Eclipse IoT working group as they are more flexible and feature-rich.

F. Eclipse Hono

Eclipse Hono is a multi-protocol IoT hub, connecting a large number of devices with a cloud-based business application [20]. One main benefit is that Hono supports diverse protocols such as HTTP, MQTT, CoAP, and AMQP 1.0 to interface with the devices via the so-called southbound API. For each of these protocols, Hono has a so-called protocol adapter that transports the data to the messaging network of Hono. It is then possible for a business application to retrieve that data from the network using an AMQP 1.0 endpoint from Hono. It is further possible to support additional protocols by implementing the corresponding protocol adapter.

Hono supports three types of data exchange. First, it allows the transmission of telemetry data to the business application. Examples are temperature or humidity information. Second, it is possible to transmit events to the business application to indicate situations like the completion of a process step in an industrial plant. The difference between telemetry and event data is the delivery semantics. For events, the delivery happens ”at least once”. In the case of telemetry messages, the delivery mode is either ”at least once”; or ”at most once”. Third, business applications can carry out the command and control interaction patterns to trigger actions on a device and send data to the device.

When it comes to possible scenarios for using Hono in the context of the Arrowhead Framework, there are two imaginable applications. Hono covers an important aspect of every IoT infrastructure, which is the linkage of the devices with business applications. Especially, devices that do not expose RESTful APIs via HTTP and may rely on a broker infrastructure as in publish-subscribe protocols (e.g. MQTT) benefit from a centralized device hub. Below, we introduce an approach to how Hono could be a part of such a device hub system. Moreover, Hono provides standardized communication patterns as described above.

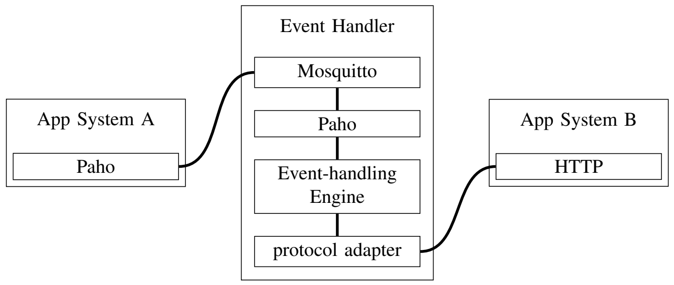

Fig. 3: Using Eclipse Mosquitto and Paho to build an Arrowhead event hadling system.

The multi-protocol capability of Hono seems promising to create a translation system as well. The purpose of the Arrowhead translation system is to provide a bridge between systems that do not use the same protocol for communication [3]. This translation system would enable an easy transformation of e.g. MQTT to HTTP or CoAP. However, the standardized communication patterns in Hono expect the involved systems to be aware of the translation and its limitations.

G. Eclipse Mosquitto & Eclipse Paho

Eclipse Mosquitto [21] for the broker and Eclipse Paho for the client [22] are implementations for using the MQTT protocol [23]. Together they could build the base for an Arrowhead event handler system [1], [24]. As the event handler should provide publish-subscribe functionality [1], an MQTT-based implementation does fulfill this requirement. The Arrowhead event handler exposes an API to request a transient instance that functions as a proxy between two services in the local cloud. This proxy then consumes a service (e.g. a sensor system), filters the results, and triggers an event at the initiator.

As depicted in Fig. 3 a potential event handler implementation uses Paho to transmit data, filtered by the event handling engine, to the Mosquitto broker. The source application system (App System B) could either use a request/response or publish/subscribe messaging pattern. The event handler uses a protocol adapter, and if necessary employs the translation system to connect. The receiving system (App System A) connects to the Mosquitto broker using Paho as well.

H. Eclipse Kura

Eclipse Kura is an OSGi-based framework for building IoT gateway applications [25] by abstracting the hardware layer of the underlying device and enabling remote management. In terms of applications within the Arrowhead Framework, one could think of installing Kura to the device and use it then as infrastructure for deploying own systems.

I. Eclipse Kapua

Eclipse Kapua is a comprehensive IoT cloud platform featuring device connectivity, device registration, message

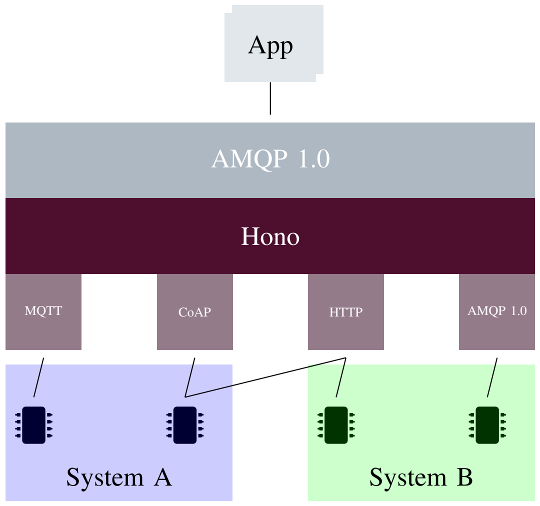

Fig. 4: Integration of producer systems under Eclipse Hono

routing, device configuration, and data persistence functionality [26]. Currently, devices connect with the southbound API using MQTT. Business applications use either AMQP 1.0 or Websockets with the northbound API. An LWM2M-integration, for standardized device configuration and management, is supposed to follow.

Kapua's device registry could be used in an Arrowhead context as a device registry system. Likewise, its device management component could function as a configuration system and the data persistence layer as a foundation for a historian system like mentioned by [1] and [24].

III. An Eclipse Hono based device hub

The Arrowhead Framework comprises an extensible number of supporting core systems to which we propose to add a "Device Hub" system. This system provides a central interface for the communication to and from devices. Devices often form a heterogeneous infrastructure of multiple connectivity solutions and protocols. Hence, a multi-protocol module that centralizes this disparate network traffic and provides a messaging broker could be beneficial.

The base for this module could be Eclipse Hono, which provides multiple protocol adapters that enable a wide variety of different devices to connect. The northbound API used by other business applications or services exposes an AMQP 1.0 interface on which one can consume all data produced by the devices. To achieve this, all incoming messages at the protocol adapter are translated into AMQP 1.0, processed by an underlying messaging network, and forwarded to the corresponding business application at the northbound API. Likewise, commands sent by the business applications get forwarded to the correct devices using the conversion in the protocol adapters.

To make Hono compatible with the Arrowhead ecosystem, the APIs [27] need to be extended by an administration interface, that offers management functionality, such as creation and deletion of respective services and corresponding devices. To integrate a new device via Hono, the Arrowhead administrator only needs to interact with the Device Hub service. The Device Hub service will then register the new device in Hono and a consumable service in the Arrowhead service registry for the produced data of the device. To comply with the Arrowhead system architecture, one can logically cluster devices within the same system. For instance, a temperature monitoring system, consisting of dozens of sensors could be grouped within the Service Registry.

Fig. 5. depicts the administration process in detail. After the services of the Device Hub have been registered at the service registry (1), it is possible to integrate a new hypothetical sensor system by using the new device hub API. This could either be done manually by an administrator or automated through an administrative gateway which requests the device integration. However, orchestration rules must get inserted manually, as they involve the consuming system as well. To authorizes as a device at the southbound API of Hono the sensor system then needs the credentials. These device credentials get generated as well during this process, and their provisioning to the sensor system depends on the actual implementation of that system. An application system seeking to consume a device service then initiates an orchestration process at the orchestrator system (2). The orchestrator system checks the privileges of the application system, with the help of the authorization system, and subsequently hands over the necessary information to access the sensor system. The access information then points to the address of the northbound API of Hono, supplemented by a service-specific identifier. Afterward, the Application System can now interact with the sensor system, for instance by subscribing to specific measurement data (3).

The described architecture of an Hono based device hub inherently restricts the possible use cases. All systems connected to the southbound side of the Device Hub can only be a service producer, as there is no possibility to initiate a service orchestration due to a missing link to the service registry.

However, a fundamental challenge remains to be addressed, which is authentication and authorization. To restrict access to Hono's southbound APIs, each device gets individual credentials, enabling fine-grained access control. Hono's northbound API does not offer this feature, as it relies on the Apache Qpid Dispatch Router [28]. Access is secured by a SASL-based mechanism and there is no user management API available. There are three potential solutions to this problem. First, each eligible consuming application system uses the same credentials, transmitted during the orchestration process, and consequently has the same access rights. Moreover, using Hono's tenant functionality would allow the creation of credentials that at least differ for each integrated device, service, or system. This ensures that the access is restricted to a group of consuming application systems belonging to the same tenant. But it does not prevent password leaks and misuse. Second, one could extend Hono's internal structure to integrate another authentication mechanism. However, that might involve complex changes. Third, supplementing Hono with a northbound wrapper seems most promising. This wrapper needs to handle

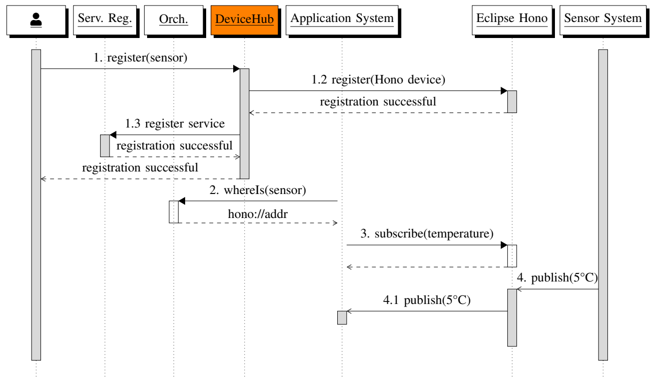

Fig. 5: System interaction between supporting core services and the Hono-based messaging hub.

authorization and authentication in more detail. For instance, Eclipse Ditto integrates with Hono's AMQP 1.0 API and has more fine-grained access control mechanisms.

IV. Next Steps

The Arrowhead Framework already contains various systems serving different purposes within an Arrowhead instance. As described above, some aspects are not covered yet but could be beneficial for automating a shop floor. We identified the mentioned open source projects that appear helpful in the context of Arrowhead. As a consequence of our analysis, we propose the following next action points. Eclipse hawkBit could be used to distribute software packages and other updates like configurations. One further should investigate how the services provisioned by Eclipse Ditto can be automatically registered in the service registry. To use Eclipse Vorto within the Arrowhead context it is also required to more deeply analyze whether the meta-models of the Arrowhead Framework and Vorto are compatible and whether the provided code generators and other supporting facilities are useable within the Arrowhead Framework.

V. Conclusion

The paper presented a high-level overview regarding the integration possibilities of Eclipse IoT technologies into the Arrowhead Framework. As the analysis revealed within Eclipse IoT there are not only projects of potential interest for IIoT applications but some that could be of special interest for the Arrowhead Framework. For Eclipse Hono the paper already shows a possible integration. The next step is to come up with approaches for similarly integrating the other discussed projects. A next integration candidate is Eclipse hawkBit, to pave the way towards more automation for the distribution of software and configuration updates in Arrowhead operations.

Acknowledgment

The research leading to these results has received funding from the EU ECSEL Joint Undertaking under grant agreement n° 826452 (project Arrowhead Tools) and from the partners' national funding authorities VDE on behalf of BMBF.

References

[1] J. Delsing, *IoT automation: Arrowhead framework*. CRC Press, 2017.

[2] Eclipse IoT Working Group, "Open Source Software for Industry 4.0," 2017. [Online]. Available: https://iot.eclipse.org/resources/white-papers/Eclipse IoT White Paper - Open Source Software for Industry 4.0.pdf

[3] H. Derhamy, J. Eliasson, and J. Delsing, "IoT Interoperability - On-Demand and Low Latency Transparent Multiprotocol Translator," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1754–1763, 2017.

[4] O. Carlsson, D. Vera, J. Delsing, B. Ahmad, and R. Harrison, "Plant descriptions for engineering tool interoperability," *IEEE International Conference on Industrial Informatics (INDIN)*, vol. 0, pp. 730–735, 2016.

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

[6] Eclipse IoT Working Group, "Charter," 2019. [Online]. Available: https://www.eclipse.org/org/workinggroups/iotwg_charter.php

[7] ——, "Explore Our Members." [Online]. Available: https://iot.eclipse.org/membership/members/

[8] ——, "Eclipse IoT Adopters." [Online]. Available: https://iot.eclipse.org/adopters/

[9] ——, "The Three Software Stacks Required for IoT Architectures," 2016. [Online]. Available: https://iot.eclipse.org/resources/white-papers/Eclipse IoT White Paper - The Three Software Stacks Required for IoT Architectures.pdf

[10] "Eclipse Ditto • open source framework for digital twins in the IoT." [Online]. Available: https://www.eclipse.org/ditto/

[11] "Eclipse Vorto." [Online]. Available: https://www.eclipse.org/vorto/

[12] Open Mobile Alliance, "Lightweight M2M (LWM2M) - OMA SpecWorks." [Online]. Available: https://www.omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/

[13] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," *Rfc 7252*, 2014.

[14] O. Carlsson, P. P. Pereira, J. Eliasson, J. Delsing, B. Ahmad, R. Harrison, and O. Jansson, "Configuration service in cloud based automation systems," *IECON Proceedings (Industrial Electronics Conference)*, pp. 5238–5245, 2016.

[15] "Eclipse Leshan." [Online]. Available: https://www.eclipse.org/leshan/

[16] "Eclipse Wakaama." [Online]. Available: https://www.eclipse.org/wakaama/

[17] "Eclipse hawkBit." [Online]. Available: https://www.eclipse.org/hawkbit/

[18] "Eclipse Keti." [Online]. Available: https://projects.eclipse.org/proposals/eclipse-keti

[19] D. Hardt, "The OAuth 2.0 Authorization Framework [RFC 6749]," *RFC 6749*, 2012.

[20] "Eclipse Hono." [Online]. Available: https://www.eclipse.org/hono/

[21] "Eclipse Mosquitto." [Online]. Available: https://mosquitto.org/

[22] "Eclipse Paho - MQTT and MQTT-SN software." [Online]. Available: https://www.eclipse.org/paho/

[23] OASIS, "MQTT Version 3.1.1," *OASIS Standard*, 2014.

[24] A. Zabasta, K. Kondratjevs, J. Peksa, and N. Kunicina, "MQTT enabled service broker for implementation arrowhead core systems for automation of control of utility' systems," *Proceedings of the 5th IEEE Workshop on Advances in Information, Electronic and Electrical Engineering, AIEEE 2017*, vol. 2018-Janua, pp. 1–6, 2017.

[25] "Eclipse Kura." [Online]. Available: https://www.eclipse.org/kura/

[26] "Eclipse Kapua." [Online]. Available: https://www.eclipse.org/kapua/

[27] "Eclipse Hono API." [Online]. Available: https://www.eclipse.org/hono/docs/api/

[28] "Apache Qpid." [Online]. Available: https://qpid.apache.org/