Industry 4.0 Solutions for Interoperability: A Use Case about Tools and Tool Chains in the Arrowhead Tools Project

This is the post peer-review accepted manuscript of:

R. Venanzi, F. Montori, P. Bellavista and L. Foschini, "Industry 4.0 Solutions for Interoperability: a Use Case about Tools and Tool Chains in the Arrowhead Tools Project," 2020 IEEE International Conference on Smart Computing (SMARTCOMP), Bologna, Italy, 2020, pp. 429-433, doi: 10.1109/SMARTCOMP50058.2020.00089.
The published version is available online at:
https://doi.org/10.1109/SMARTCOMP50058.2020.00089

Industry 4.0 Solutions for Interoperability: a Use Case about Tools and Tool Chains in the Arrowhead Tools Project

Riccardo Venanzi, Federico Montori, Paolo Bellavista, Luca Foschini
Department of Computer Science and Engineering (DISI)
University of Bologna - Bologna, Italy
riccardo.venanzi, federico.montori2, paolo.bellavista, luca.foschini@unibo.it

***Abstract*—Industry 4.0 outlines the trend of the massively adoption of Internet of Things (IoT) nodes in supply chains, manufacturing, and factories in general. The industry digitalization is the key enabler to ease the productive process, drastically reduce its costs, and boost up the associated business. In this context, Arrowhead Tools (AHT) is a H2020 EU project provided by ECSEL that targets automation and digitalization solutions for the industry in Europe. AT is based on a framework, named Arrowhead Framework (AHF), developed and provided by the previous Arrowhead (AH) project. AHF is open source and addresses IoT-based automation and integration by abstracting IoT objects to services. AHF enables IoT interoperability and provides real time data handling, security features, automation system engineering, and automation systems scalability. In this paper, after a rapid overview of the AT project and the AHF architecture, we originally introduce the concept of Tool and Tool Chain for Industry 4.0 in AH. We also present a vertical AHT use case along with its implementation, as well as all the steps to turn a service/application into an AH-compliant Tool.**

***Index Terms*—Industry 4.0, Interoperability, Framework Infrastucture, Arrowhead**

I. Introduction

The IoT paradigm is playing a pivotal role in our everyday life. Nowadays the number of connected devices has already surpassed the world population, and novel device integration approaches are presented ever more frequently. In addition, this phenomenon is intensifying with the massively extension of IoT adoption in the industry domain. In this context the concepts Industrial IoT (IIoT) and Industry 4.0 have emerged [1]. Among these largely heterogeneous devices (systems), interoperability has widely recognized as one of the hottest and most challenging issue of the last decade. At this purpose, several interoperability frameworks have been presented, everyone targeting its own use case. This trend leads to have many IoT Clouds acting as closed islands, with no interoperability among each other. However, the full potential of IoT can be exploited only if it is possible to combine and to make interoperable heterogeneous datasources. Hence, many researches in the field are focusing on connecting together heterogeneous IoT Clouds. AHF is an interoperability Service Oriented Architecture (SOA), that aims to interconnect several IoT-based clouds. It has been proposed within the AH EU project [2].

In this paper, for the sake of easy and full understandability, we present a description of AHF in the novel context of the AHT EU project. In particular, the core parts of AHF will be described along with their functionalities. In addition, the full procedure to interact with AHF and all the steps required for service fruition will be detailed. Then, we originally introduce and define the concept of AH tools and tool-chain process for Industry 4.0. By delving into finer details, we exploit AHF to provide common and interoperable tools to improve the associated engineering process. In addition, the paper provides implementation insights about a real deployment of an AH Local Cloud (LC) with all the steps to deploy the AH Core Services and to develop service producers/consumers.

II. Related Work

When first IoT ecosystems emerged, the predominant deployment approach was monolithic or ad-hoc in the mostly cases. More recently, innovative approaches have been adopted: novel IoT deployment solutions are moving close to SOA, in which any IoT object is an entity capable of consuming or producing services. In this context AHF emerges [2]. AHF is the result of an EU project, funded by the Artemis Industry Association, with the collaboration of more than 80 partners among companies and universities. AHF has been designed with the idea of supporting IoT automation scenarios at every level, and it bases its approach on the key guidelines that characterize SOA: service lookup, late binding, and loose coupling [3]. AHF is extensively used in many EU projects, i.e., AHT, Productive 4.0, Far-Edge, just for naming a few [4]–[6]. In [7], a first large overview on the framework is given. Varga et al. provide an high level description of the framework characteristics and how it supports the systems integration and interoperability by defining the concept of System of Systems (SoS). It also addresses the problem of inter-cloud communication and service fruition in AHF by presenting the two systems that enable the communication among two entities located in different Clouds [8].

AHF has already been used in real case applications. For example, in [9], the framework has been adopted to integrate devices for structural condition monitoring with services for analytic published on the framework. The integration of systems and the adaptation of legacy systems are two goals for

novel IIoT environments: an AHF-based service able to ease integration procedures has been presented in [10].

Several studies helped to overcome some limitation AHF originally had. In [11], the re-engineering of an AH application has been proposed to overcome maintainability issues in the application code and to ease the addition of new functionalities or the versioning upgrade. Another work to improve an AHF component is presented in [12]. Rocha et al. propose an upgrade of the Event Handler system based on the publish/subscribe paradigm to improve performance in terms of CPU and quality of service. With the emerging of the Industry 4.0 paradigm, some works tried to extend AHF to be more tailored to this perspective. In [13], a Non-Intrusive Load Monitoring (NILM) toolchain development based on AHF has been proposed. While a OPC Unified Architecture (OPC-UA) service integration with AHF has been presented in [14].

Of course, many other IoT frameworks for IIoT exist in addition to AHF, each one with its own specific features and peculiarities [15].

III. Arrowhead Framework For Interoperability

AHF has been designed with the goals of enabling IoT interoperability, automating system engineering, providing scalability, security, and real-time data management. The primary idea is that AHF should be able to turn any IoT device or system into a service available at the Cloud level, in order to ease availability, interoperability, and usability.

A. The Arrowhead Framework Architecture

The AHF architecture is based on the concept of Local Cloud (LC). A LC is defined as a separated set of IoT objects and industrial things geographically close to each other. Each LC has its own set of AH Core Services (CSs) that enable all the main features traditionally present in SOA scenarios, such as service lookup, late binding, and loose coupling. AHF aims to overcome system and device heterogeneity by providing a common infrastructure with standardized interfaces for providing and consuming services. More in detail, the framework aims to fully decouple service providers and consumers in order to reach the complete interoperability. AH defines a common and standard interfaces that allow the heterogeneous systems supporting HTTP to be AH compliant. A system in the AH environment is a entity capable of providing and/or consuming services. The service fruition is point-to-point between provider and consumer, while all the communication, negotiation, discovery and service lookup operations are demanded to the framework. The interoperability among different systems and devices has a pivotal importance in the IoT scenario where the number of connected devices has largely surpassed the world population, and it becomes even more crucial with the emerging of Industry 4.0 paradigm. The AH project provides several core services, partly mandatory and partly optional to be implemented and/or deployed. An AF-compliant LC must have at least the AH mandatory core services installed and running. The mandatory CSs allow industrial systems and IoT objects to be seen and used as AH Services. All the LCs are connected to each other with a mesh topology network, thus allowing an AH Service to use (or to be used by) other services located in a different LC. The mandatory AH CSs provide service registration, lookup functionalities, orchestration and security services. The AH mandatory CSs are those services that actually enable the LC and allow third-party services to interact to each others. An AF entity can be a Service Consumer (SC), a Service Producer (SP), or both of them. It is also possible to develop an AH adapter capable of wrapping legacy systems in order to make them AH compliant.

Fig.1 highlights the AH LC model. Fig.1 outlines the typical entities composing an AH LC, AHF services (colored boxes) and user-defined services (gray boxes). Any user-defined service can provide or consume other user-defined service through AH CSs. More specifically, the Fig.1 depicts all the three mandatory CSs, Service Registry, Orchestration System, and Authorization System, and the two main support CSs, Gatekeeper System, and Gateway. The concise description of the mandatory CSs is provided below.

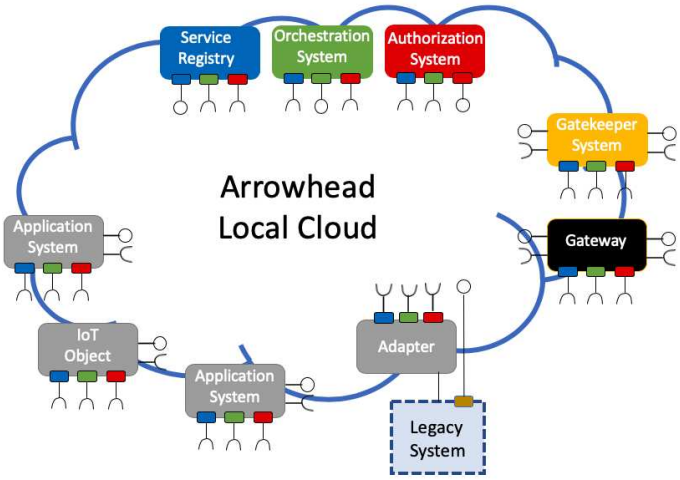

Fig. 1. Arrowhead Local Cloud Architectural Model.

Service Registry
It provides service registration functionalities and acts as service repository. It stores all the meta information of the registered services such as service description, IP and port, transport, communication protocols, interfaces, etc. It is also responsible for the service lookup by other services. The service lookup operation is executed by using the DNS-SD lookup protocol [16].

Orchestration System
It is the AH service in charge of coordinating the services' interactions through the framework. Orchestration System is responsible to match the consumers' requests with the services available on the Service Registry. This system can dynamically select the service producer that best fits the requester needs. In addition, the Orchestrator is also responsible for load

balancing and fault tolerance on service producer side.

Authorization System
This system manages the correct fruition of the services. Indeed, the Authorization System is responsible of granting the rights and permissions for the service fruition.

In the AH LC, there typically are two support services deployed, Gatekeeper System and Gateway System. These two services are optional, and a LC can work even if they are missing. Although they are not mandatory, Gatekeeper and Gateway Systems are often deployed in any LC because they are responsible for communication between different LCs. More in detail, the Gatekeeper System is responsible for the inter-cloud orchestration process. It manages the control information for the inter-cloud communication, it is not directly responsible of the data flow between producer and consumer. The Gatekeeper System internally consists of two services, Global Service Discovery (GSD), and Inter-Cloud Negotiation (ICN). GSD locates the best fitting service in the neighboring Clouds. The ICN is responsible for establishing mutual trusted connection between the two LCs, then the actual connection between the endpoints is built. The Gatekeeper System tightly works with the Orchestration Systems of the two LCs. The Gateway has two main tasks, it establishes a session between producer and consumer, and it is responsible to manage it once established. The Gateway is the mediator between producer and consumer. It provides the interfaces for connecting to the producer/consumer and then all traffic flows through it. Table I lists some other support CSs provided by AF along with their main features. As already stated, they are not mandatory and may be deployed depending on specific use case requirements.

TABLE I
AH SUPPORT CORE SERVICES

Service Name	Service Features
Event Handler	pub/sub event support
System and Device Registry	Un/Register Devices and their systems
Quality of Service (QoS) Manager and Monitor	Monitor Service QoS and Support to Service Level Agreement (SLA)
Translation	Supports languages translation
System Configuration	Tools to Define and Manage System Configuration

B. Service Fruition in AF

Interoperability is the key principle on which AF is based. Two completely different, or even legacy, systems can exchange services to each other through AF. More formally, an AF System is that entity in the framework which provides or consumes services. A System can provide one or more services and at the same time it can consume one or more other services. An AF service is defined as what it is exchanged from a providing to a consuming System. To provide and/or consume an AF service, a specific procedure has to be followed, as depicted in Fig.2 and detailed below.

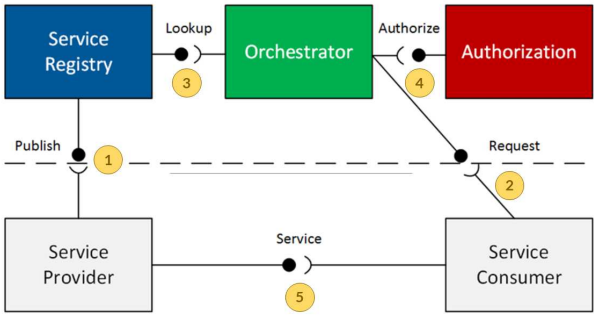

Fig. 2. In-Cloud Service Fruition Steps

1) In the first step the service producer publishes the service and its metadata on the AH Service Registry. In this way the service became available and theoretically accessible to all AH users. The metadata published on the Registry contains several types of information, for example producer endpoint, authorization information, service interface and protocols, etc.
2) Service consumer queries the AH Orchestrator System with the data about itself and the service it wants to consume.
3) The Orchestrator System queries the Service Registry with the requested service information and it tries to provide the service lookup.
4) The Orchestrator System queries the Authorization System and checks the rights for consuming that provider's service.
5) If consumer's service request matches with one or more services registered on Service Registry, and the Authorization Systems grants the service fruition, the Orchestration System provides the service look up information to the consumer. The information contains the service producer endpoint, the interface for invoking the service, the protocol used, etc. The service consumer starts to consume the service.

It is worthy to note that the actual service fruition is performed in a point-to-point manner between producer and consumer; AF is not involved any longer. From the stages above listed, it emerges that the framework is exploited only in the control flow of the service fruition, the discovery, the security matter, the providers' load balancing and dispatching, etc.. The actual data flow is directly performed producer to consumer, in a end-to-end manner.

IV. TOOLS FOR INDUSTRY 4.0

A common trend in Industry 4.0 is to shift from a SCADA/DCS-driven organization of components to a networked IoT ecosystem where each entity is responsible for producing or consuming services. AHF has proven to be a valid candidate for IoT integration not only in IIoT scenarios, but also in the engineering phases of industrial products, being

them software or hardware. In order to cover such needs, the AHT project[1] aims to use AHF as the glue between tools operating in different phases of the engineering process.

A. Arrowhead Tools and Tool Chain

Industry processes need an agile way to cover the engineering process of products and artifacts. Within the last months, AHF has shifted its use to promote the cooperation of engineering tools, rather than application systems. The major difference between the two concepts relies on the fact that application systems are components that interact within a steady-state instance of the System of Systems, whereas engineering tools are pieces of software (occasionally hardware) that interact throughout the engineering process of an artifact until its release and support its definition, development, deployment, and maintenance. More in detail, the AHT project assumes as a baseline the ISO/IEC 81346 waterfall model [17] which has been extended with two new phases: evolution and training. The resulting model is shown in Figure 3 with the eight process phases and their input and output interfaces. In particular, the engineering process is defined by four main entities:

- Engineering Process Phase (**EPP**) is a phase of the engineering process that cover all the actions performed against the realization of an artifact.
- Engineering Process Provider Interface (**EP-PI**) (sometimes called Engineering Process Output) is the way the outcome of the set of tools operating throughout an EPP is delivered for being consumed.
- Engineering Process Consumer Interface (**EP-CI**) (sometimes called Engineering Process Input) is the way the set of tools operating throughout an EPP receives inputs from the previous phases.
- Engineering Process Unit (**EPU**) is a superclass identifying one of the three concepts expressed above.

It is important to state that the EPPs are not in chronological orders, as many use cases may use the EPPs in a discontinuous way as well as performing more iterations. For this reason, this guideline is considered more as a meta-process where phases can be scheduled in different orders on top of the correspondence between EP-PIs and EP-CIs.

Now, the AHF brings a significant boost to this engineering process – we will hereafter refer to it as the Extended Automation Engineering Model (EAEM)– by defining a number of AH Engineering Tools that must support the industrial process within and between each phase. An AH Tool – we will hereafter refer to it only as Tool – is a software or a hardware (together with adequate software) artifact that supports Cyber-Physical System of Systems. Tools are not mandatory, i.e. phases of the engineering process can still be carried out without them, however it means that there will be a strong human component. Depending on the EPP it refers to, it can pursue tasks at design-time or at run-time. A Tool is necessarily either an AH Provider or a Consumer, in short, an application system with the specific purpose of supporting one or more EPPs and it cannot be broken down conceptually in subtools that work separately, in short, it is an atomic entity. Most importantly, a Tool is defined as such only when it can be part of a Toolchain, by interacting automatically with other Tools throughout other EPPs. A Toolchain, as a consequence, is a collection of Tools that interacts without human intervention thanks to the AHF, which allows, loose coupling, late binding and Tool reuse/replacement. In fact, the trend within the project would be to put a set of orchestration rules within the Orchestration System involving the Tools, an example of it can be seen in [18]. This is how AH Toolchains are defined. A simple example might be given by a sensor monitoring scenario: within the procurement and engineering phase the right sensors have to be listed and purchased for the task, in the deployment and commissioning phase sensors have to be configured with the right parameters and placed in the right spot on the field, finally, in the operation and management phase the wakeup-sleep life cycle of sensors have to be scheduled correctly in order to preserve battery life and still monitor the phenomenon of interest. These three phases can be covered by three Tools and such Tools need, in turn, to expose their outputs in a unified way, so that it is consumable by the next one. An example of a more complex AH Toolchain for Home Automation Scenarios is found in [19].

B. Tool Implementation Insight

We have implemented and deployed our own AH LC at the University of Bologna (Unibo). We created our AH-compliant environment by deploying the three mandatory CSs on our private server; this environment is now working and is used by the whole Italian cluster (IUNET) inside AHT [4]. The server hosting the AH CSs have the characteristics listed in Table II. The entire IUNET refers to this server as the AH CSs server, all the interactions among tools, systems, and services that requires AHF mediation pass through it.

TABLE II
AH UNIBO SERVER

CPU	Intel Core i7-920 @ 2.67GHz
Number of Cores	4
Thread per Core	2
Cache	8 MB
RAM	8 GB
Hard Disk	500GB

As already stated in Section III, the mandatory AH CSs defining a LC are Service Discovery, Orchestration System, and Authorization System. These three components constitute the minimum set of services to create an AH LC and to allow third party services and systems to interoperate to each other through AHF. The installation of such services requires some additional components. The first necessary accessory component for the CSs installation is Docker [20]. Docker is crucial in the services installation because AHF exploits the containerizaion to deploy the CSs themselves and a database.

[1] https://www.arrowhead.eu/arrowheadtools

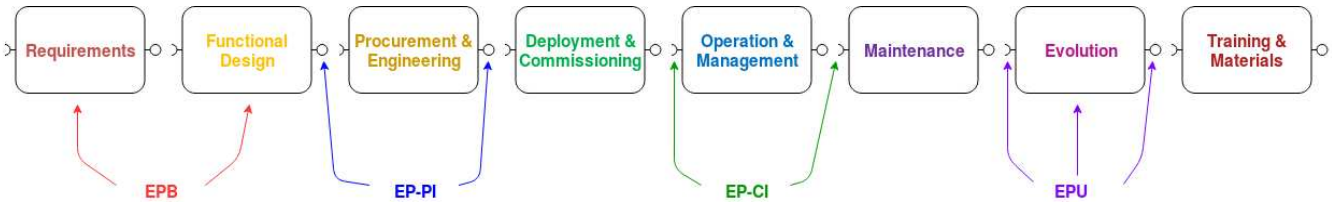

Fig. 3. Engineering process use in the Arrowhead Tools project

The database is another crucial accessory component needed to correctly install and run the CSs. Indeed, it is used by Service Registry to store the information about the registering services. The RDBMS adopted by AH project is Oracle MySql version 5.7+. The CSs are provided by AH project, they are written in Java and it is required at least the JDK/JRE 11 installed on the server. The AH CSs are based on Spring Framework and deployed as Spring services. AH exploits the functionalities provided by Spring framework in order to obtain flexibility, modularity, and a easy web access. The code of the AH CSs is open source and available at [21]. Once all the required components are installed, the code can be compiled and the services are ready to be configured. At this point, the LC is created but it is necessary to configure the networking settings in order to allow the services to see and interact to each other. Each CS has a specific file named *application.properties*. This file contains all the network settings that can be tuned in order to proper configure the AH LC. Each CS has its own properties file, this file has a standard format among all the CSs and it is composed by three main section:

Application Parameters:
: This section contains all the settings of the general application, such as the database connection parameters, the Spring Framework parameters to interact with the persistency layer, and the specific service end point information.

Custom Parameters:
: It addresses all those parameters that have to be properly tuned according with every specific deployment. These parameters are: service name, the Service Registry endpoint, and some other service-specific parameters (i.e. in the Orchestrator file, a flag to enable the interaction with the Gatekeeper System and the Gateway).

Security Mode:
: This section contains all the security related parameters. More in detail, the AHF lays the security features on the certificates system. This section contains all the certificate information and authorization parameters.

Once the property files of the CSs are properly tuned the services jars can be launched and the services will be up and running at the specified endpoints. The default deployment properties will expose the services at localhost, ports: 8443, 8445, 8441, for Service Registry, Authorization, and Orchestrator respectively. A Swagger with the services API will be available at the each CS entry point. Once the CSs are running, a provider can publish a service and a consumer can search for a specific service, obtain the provider's endpoint, and finally, consume the service. In [22], the skeletons to develop a full AH compliant producer and consumer are published, but some further steps are needed in order to proper perform the service fruition. First of all, the producer has to define the interface of the service it wants to provide. The service definition contains all the information about the service itself and the service producer; this definition is written in JSON. Each service has to be registered on the Service Registry. The registration has to be performed with a HTTP/POST call to Service Registry in JSON.

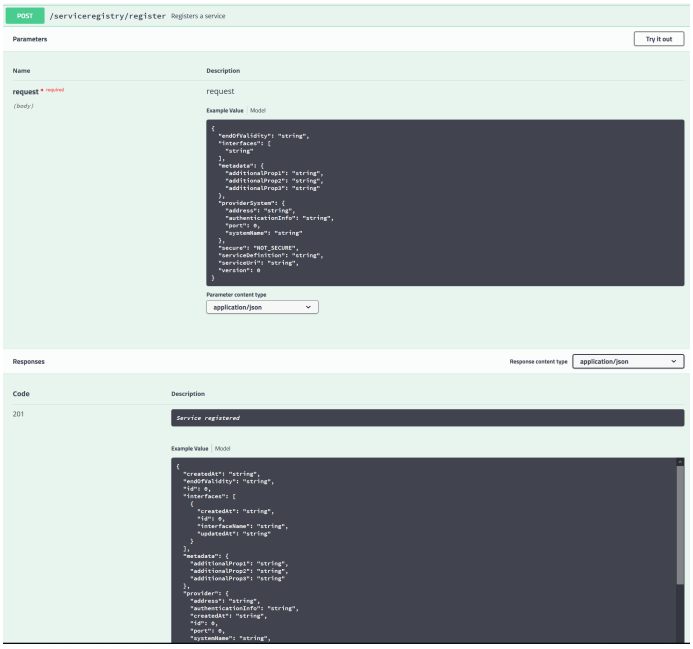

Fig. 4. Example of Service Registration Call and possible Response

As shown in Fig.4, the input JSON parameter of the registration call contains information about the service provider endpoint, some additional metadata, a property for the service validity, and the information about the service itself. After the service registration, it is necessary to create an authorization rule to grant the access to the specific consumer. The authorization rule has to be created by a HTTP/POST call to Authorization System. This call has to be done to the management interface of the Authorization System by specifying which service can be consumed. This rule can be

valid for inter-cloud, and/or intra-cloud requests. An example of authorization rule creation is depicted in Fig.5.

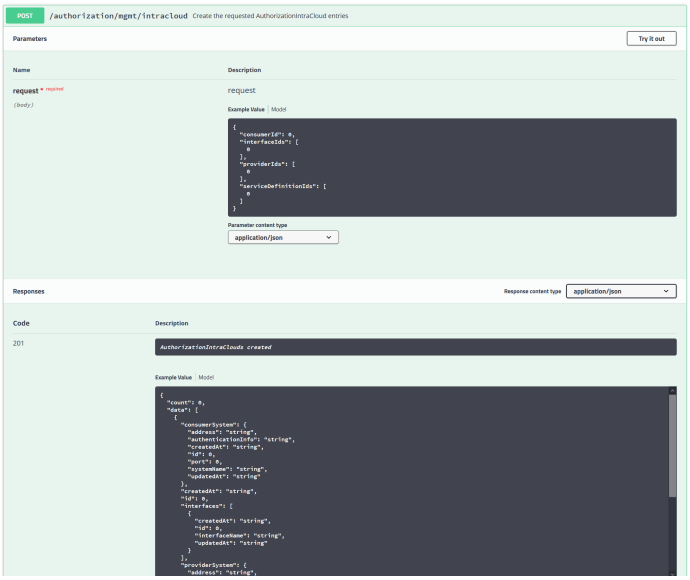

Fig. 5. Example of Authorization Rule Creation

As shown in Fig.5, the HTTP/POST call requires as input parameter a JSON containing the consumer ID, the interface (can be multiple), the provider (can be multiple), and the service definition (can be multiple). At this point, all the preliminary steps for service fruition are done, and the consumer can look for the service and consume it. The producer and consumer are developed as Java Spring Boot applications. Their skeletons are available on this repository [22].

V. Conclusive Remarks

In this paper we have presented AHF and the procedure needed for an AH-compliant service fruition. Most importantly, this paper has originally described the concept of tool and toolchain for AH environments. Moreover, a real implementation of the AHF ecosystem has been presented: we have successfully deployed an AHF instance at Unibo and originally described here its main implementation insights; this deployment is currently used as reference framework for the development of the IUNET use cases in the AHT project.

Acknowledgment

This research is funded by the ECSEL JU under grant agreement No 826452 (Arrowhead Tools), supported by the EU H2020 programme and by the member states.

References

[1] H. Geng, *THE INDUSTRIAL INTERNET OF THINGS (IIoT)*. Wiley, 2017, pp. 41–81. [Online]. Available: https://ieeexplore.ieee.org/document/8044783

[2] Artemis. (2013) Arrowhead Framework. [Online]. Available: https://www.arrowhead.eu/arrowheadframework/

[3] H. P. Breivold and M. Larsson, "Component-based and service-oriented software engineering: Key concepts and principles," in *33rd EUROMICRO Conf. Software Engineering and Advanced Applications*. IEEE, 2007, pp. 13–20.

[4] (2019) Arrowhead Tools european investment for digitalisation and automation leadership. [Online]. Available: https://www.arrowhead.eu/arrowheadtools

[5] (2017) Productive 4.0. [Online]. Available: https://productive40.eu/

[6] (2016) Far-Edge factory automation edge computing operating system reference implementation. [Online]. Available: http://www.faredge.eu/#/

[7] P. Varga, F. Blomstedt, L. L. Ferreira, J. Eliasson, M. Johansson, J. Delsing, and I. M. de Soria, "Making system of systems interoperable – the core components of the arrowhead framework," *Journal of Network and Computer Applications*, vol. 81, pp. 85 – 95, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516301965

[8] P. Varga and C. Hegedus, "Service interaction through gateways for inter-cloud collaboration within the arrowhead framework," *5th IEEE WirelessVitae, Hyderabad, India*, 2015.

[9] J. Campos, P. Sharma, M. Albano, E. Jantunen, D. Baglee, and L. L. Ferreira, "Arrowhead framework services for condition monitoring and maintenance based on the open source approach," in *2019 6th Conf. Control, Decision and Information Technologies (CoDIT)*, April 2019, pp. 697–702.

[10] S. Maksuti, M. Tauber, and J. Delsing, "Generic autonomic management as a service in a soa-based framework for industry 4.0," in *IECON Conf.*, vol. 1, Oct 2019, pp. 5480–5485.

[11] T. Pedersen, M. Albano, and B. Nielsen, "Reengineering the lifecycle of arrowhead applications: from skeletons to the client library," in *IECON Conf.*, vol. 1, Oct 2019, pp. 5519–5524.

[12] R. Rocha, C. Maia, L. L. Ferreira, and P. Varga, "Improving and modeling the performance of a publish-subscribe message broker," in *IECON Conf.*, vol. 1, Oct 2019, pp. 5493–5498.

[13] D. Brunelli, T. S. Cinotti, H. Woehrl, C. Aguzzi, F. Montori, and L. Benini, "An interoperable tool-chain for energy monitoring applications," in *AEIT Conf.*, Sep. 2019, pp. 1–6.

[14] A. N. Lam and O. Haugen, "Implementing opc-ua services for industrial cyber-physical systems in service-oriented architecture," in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, Oct 2019, pp. 5486–5492.

[15] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller, "A survey of commercial frameworks for the internet of things," in *2015 IEEE 20th Conf. Emerging Technologies Factory Automation (ETFA)*, Sep. 2015, pp. 1–8.

[16] S. Cheshire and M. Krochmal, "Dns-based service discovery," RFC 6763, February, Tech. Rep., 2013.

[17] ISO Central Secretary, "Industrial systems, installations and equipment and industrial products — structuring principles and reference designations," International Organization for Standardization, Geneva, CH, Standard IEC 81346, 2019. [Online]. Available: https://www.iso.org/standard/75265.html

[18] C. Paniagua, J. Eliasson, and J. Delsing, "Efficient device-to-device service invocation using arrowhead orchestration," *IEEE Internet of Things Journal*, 2019.

[19] D. Brunelli, T. S. Cinotti, H. Woehrl, C. Aguzzi, F. Montori, and L. Benini, "An interoperable tool-chain for energy monitoring applications," in *2019 AEIT Conf.* IEEE, 2019, pp. 1–6.

[20] Docker Inc. (2019) Docker containerization technology. [Online]. Available: https://www.docker.com/

[21] Arrowhead Framework Development Organization. (2019) Arrowhead Framework repository. [Online]. Available: https://github.com/arrowhead-f

[22] ——. (2019) System of Systems Example repository. [Online]. Available: https://github.com/arrowhead-f/sos-examples-spring