Joint Power Control and Structural Health Monitoring in Industry 4.0 Scenarios using Eclipse Arrowhead and Web of Things

Carlo Puliafito[†], Lorenzo Gigli[*], Ivan Zyrianoff[*], Federico Montori[*‡],
Antonio Virdis[†], Stefano Di Pascoli[†], Enzo Mingozzi[†], Marco Di Felice[*‡]
[*] Department of Computer Science and Engineering, University of Bologna, Italy
[†] Department of Information Engineering, University of Pisa, Italy
[‡] Advanced Research Center on Electronic Systems "Ercole De Castro", University of Bologna, Italy
Correspondent author's email: antonio.virdis@unipi.it,

***Abstract*—The integration of legacy IoT ecosystems in Industry 4.0 scenarios requires human effort to adapt single devices. This process would highly benefit from features like device lookup, loose coupling and late binding. In this paper, we tackle the issue of integrating legacy monitoring systems and actuation systems in an industrial scenario, by looking into the Web of Things (WoT) as a communication standard and the Eclipse Arrowhead Framework (AHF) as a service orchestrator. More specifically, we propose a general architectural approach to enable closed-loop automation between the above mentioned legacy systems by leveraging the adaptation of the WoT to the AHF. Then, we develop a rule-based engine that enables the control of the actuation based on sensor values. Finally, we present a proof-of-concept use case where we integrate a Structural Health Monitoring (SHM) scenario with a power control actuation subsystem using the developed components.**

***Index Terms*—Industry 4.0, Arrowhead, Web of Things, interoperability**

I. Introduction

The fourth industrial revolution is an ongoing phenomenon that leverages on new concepts like the Internet of Things (IoT), Systems of Systems (SoS), Cyber-physical systems (CPS) and Digital Twins (DT). As a matter of fact, Industry 4.0 scenarios are all about relying on solid and complex ecosystems of sensors, actuators and computational elements that automatize human activities in factory premises [1]. The need for reusing single components in multiple articulated toolchains calls out for modular architectures, which should support concepts like loose coupling, late binding and lookup [2]. Moreover, interconnected systems often belong to different manufacturers and domains. As a result, a challenge for modern Industry 4.0 scenarios is to support a high level of flexibility and interoperability to fully unleash the potential of end devices as well as to support seamless on-boarding processes [3].

As an example, consider the use case shown in Figure 1, which presents an Industry 4.0 scenario. Let us assume that a factory is monitored using a set of vibration sensors (highlighted in green in the figure), used to detect potential damages to the building, e.g., through a graphical dashboard. The factory is also monitored using other environmental sensors (highlighted in blue), such as humidity meters that keep track of the moisture level in the walls. Finally, the factory machines, such as robotic arms, are powered through a set of remotely-controlled circuit breakers (highlighted in purple), which allow to remotely power on and off the various machines.

In the above use case, there could be the need to automate the power management depending on the measurements taken by the monitoring systems. For instance, for safety reasons one might decide to power off all the robotic arms as soon as a high-level of vibration is detected, which might be for example due to damages in the structure. However, the three systems described above are deployed for three independent monitoring and control tasks of the factory maintenance, using different technologies and distinct communication infrastructures, either wireless or wired. Unless addressed by design, the monitoring and actuation systems are unable to interact, and can only

Fig. 1. An exemplary use case that shows three heterogeneous legacy sensors/actuators ecosystems, represented in different colors. [Credits to icograms.com]

be accessed using a dedicated interface. To enable a closed-loop interaction between such systems, there is the need for an integrated SoS that gets inputs from sensors, applies a configurable logic to detect a critical status, and issues commands to actuators when needed.

The aim of this paper is to define the integrated SoS described above, requesting as less modification to the legacy ecosystems as possible, and relying on existing technologies. For this purpose, we propose an architecture exploiting two existing frameworks, which are well known in the Industry 4.0 and IoT domains respectively. The first one is the W3C Web of Things (WoT), a standard for describing the interfaces of IoT devices in the form of Web Things (WT) that expose interaction affordances. WoT ecosystems have however no standardized ways of performing discovery and lookup operations. Therefore, integrating multiple WoT deployments is a non trivial task that most of the times would require either the modification of WTs or a middleware that acts as a bridge. For this reason, we complement WoT with a second framework, i.e., the Eclipse Arrowhead Framework (AHF), which is a service-oriented architecture (SOA) allowing lookup, late binding and loose coupling within IoT-based industrial scenarios [4]. Recently, the AHF has included the WoT-Arrowhead Enabler (WAE) within its core systems, which enables the seamless integration of different WoT ecosystems via the AHF by offering discovery, authorization and orchestration capabilities [5]. We leverage the AHF together with the WAE to integrate a monitoring and an actuation WoT ecosystem. Moreover, we develop a control logic that empowers both. More in detail, our main contributions are the following:

1) We propose an architecture based on the AHF and WoT to enable closed-loop automated interaction between different legacy systems.
2) We develop a configurable logic to support the control loop in a Structural Health Monitoring (SHM) use case.
3) We validate the proposed architecture on a proof-of-concept (PoC) deployment.

The rest of the paper is organized as follows. Section II provides a brief background on the AHF and WoT, whereas Section III describes the proposed architecture. In Section IV, we discuss the implementation of our control logic. Then, Section V describes our proof-of-concept deployment in a realistic use case. Finally, Section VI concludes the paper.

II. Background

A. Eclipse Arrowhead

The Eclipse AHF was designed within the Arrowhead EU project[1] between 2014 and 2017 and has been endorsed by Eclipse in 2020[2]. The AHF was designed with the goal of supporting the transition from industrial SCADA/DCS systems, typically presented as monolithic architectures, to distributed SoS that are self-configurable, intelligent and integrated [4]. More in detail, the AHF is defined as a SOA wherein each ecosystem is called a "local cloud". Within a local cloud, systems interact with each other through service interfaces. In particular, each system is a service provider and/or a service consumer, and interactions are mediated by the Arrowhead Core Systems, which is a set of systems instantiated in the local cloud to support various operations. Local clouds can then communicate with each other through dedicated core systems [6]. Among the many core systems released, we use three of them in particular, which are labeled as "mandatory" in order for a SoS to be Arrowhead-compatible: Service Registry, Authorisation and Orchestration. The Service Registry system enables discovery and service lookup, as it stores in a service record the information of each service registered within the local cloud, along with its endpoints and metadata. The Authorisation system stores a set of rules that keep track of which service can be consumed by which system within the local cloud. Besides, it acts as a certificate authority. The Orchestration system provides a local cloud administrator with the capability of setting up orchestration rules, which are basically able to instruct systems on which service they have to query in order to participate in a certain process [7]. Furthermore, a recently introduced core system, the WAE, presented in Section II-C, supports the automated integration of WoT scenarios [5].

B. W3C Web of Things

The last few years have shown an exponential increase in the number of devices connected to the Internet. However, the emerging technologies provided specific interfaces and protocols, generating an unprecedented heterogeneous ecosystem. The W3C Web of Things seeks to counter the IoT fragmentation by extending existing Web technologies and standards through uniform interfaces [8].

In WoT, physical or virtual entities are abstracted as WTs, and those entities capabilities are described by a machine- and human-readable description, namely the Thing Description (TD) [8]. The TD is a fundamental building block of the WoT since it describes the WT metadata and all the information needed to interact with it through a JSON-LD document. The TD encompasses the thing information such as its *affordances*. The affordances describe: (i) properties, which are the WT state variables; (ii) actions, used to invoke functions of a WT; and (iii) events, which are occurrences that push data asynchronously from the WT.

As an IoT-based system often involves multiple things, a way to organize and search for these things is needed. In the WoT paradigm, the Thing Description Directory (TDD) implements such functionalities to store and retrieve TDs. There are several different implementations of TDDs. We highlight MODRON [9] and LinkSmart[3]. Finally, a WoT servient is a software stack that implements the WoT building blocks and can expose and/or consume WTs.

[1] http://www.arrowheadproject.eu

[2] https://projects.eclipse.org/projects/iot.arrowhead

[3] https://github.com/linksmart/thing-directory

C. WAE

The WAE[4] is an official component of the Eclipse Arrowhead Project. The tool enables seamless integration between Arrowhead services and WTs [5]. The WoT is an effective solution to the interoperability problem in the IoT environment. However, not all applications and devices can easily integrate with WTs due to the WoT interface requirements defined by the W3C [8]. Furthermore, applications are unaware of those devices' location or even existence. The Arrowhead Service Registry solves both of those problems since it exposes the interfaces and locations of the WTs as Arrowhead Thing Mirrors, which enables *any* REST user to discover and consume those IoT-based devices through the Arrowhead ecosystem. The WAE automatically discovers new WTs from a TDD and registers them as Arrowhead services through the Arrowhead Service Registry. In order to perform such a task, WAE periodically retrieves a list with all WTs stored in the TDD. Then, the tool makes a request in the Arrowhead Service Registry with the WTs metadata and detects if a WT needs to be updated, removed or registered. If any difference is detected, the WAE performs the appropriate modification in Arrowhead through the Service Registry API. WAE also enables the discovery and conversion of Arrowhead Services into WTs, though we did not explore this feature in this project.

III. Architecture

As remarked in Section I, we propose an architecture and toolchain to enable Industry 4.0 scenarios. We do this by integrating WoT-enabled ecosystems through the Eclipse AHF. More in details, we focus on monitoring and actuation scenarios, wherein the monitoring subsystem may be unaware of the existence of one or more compatible actuation sides and vice versa. The aim of our toolchain is then to enable the creation of a closed control loop by bringing together heterogeneous entities controlled by different third-parties. We assume that, for any communication channel with the outside, each party is supporting the W3C WoT standard. Hence, every endpoint can be expressed through a set of properties, actions, or events, as discussed in Section II-B. A similar assumption can be taken for any IoT industrial standard that guarantees syntactical and semantic compatibility such as OMA Lightweight M2M, OSLC, etc. Our contribution is then twofold: (i) we use the Eclipse Arrowhead Core Systems to establish a common ground and deploy all the useful hooks to the tools and systems within the local cloud; and (ii) we develop with minimal human effort a control logic that establishes the relationship between the monitoring party and the actuation party.

Figure 2 shows a high-level vision of the proposed architecture, which is composed of three main blocks interconnected through the AHF: (i) an SHM subsystem, (ii) a power control subsystem, and (iii) a control logic. The SHM subsystem is able to monitor the inertial and physical condition of a building in order to assess its health status or to forecast potential threats to the structure. The power control subsystem can monitor the power consumption of the industrial equipment, and turn it on and off remotely and on demand. Major technical details about them will be given in Sections III-A and III-B. Recall our premises, we act onto this local cloud in two steps. The first step is enabling systems within the local cloud to be announced and discovered. This is done using the WAE, which is described in Section II-C. We implement an Arrowhead-compliant control logic that is able to trigger actions in the actuation parties once given conditions within the monitoring parties are met. The control-logic component does not need to explicitly know the single endpoints, which are provided by the Arrowhead Orchestrator, therefore the only implementation effort lies in the logic itself.

A. SHM subsystem

In Figure 3, we show the architecture of the SHM subsystem, which is a complete monitoring environment divided into various layers from edge to cloud. The Monitoring layer is the closest to the physical structure and includes the network infrastructure and sensors that produce the data streams for the components above. The Edge-WoT layer represents a stratum of abstraction on top of the sensing devices designed to make data access homogeneous. In our system, we relied on the W3C WoT standard to design a set of WTs to represent our sensors and monitored structures. This approach has proven to be very functional because it allows to get rid of the heterogeneity of the underlying protocols and data formats and, at the same time, to implement filtering and pre-processing operations directly at the edge. Above these layers there is the MODRON cloud platform that includes a set of high-level services divided into two blocks: Data Management and Data Analytics [9]. The Data Management block includes several microservices that offer storage, aggregation, and visualization functionalities for the collected data. In addition, one of the services exposes an advanced interface for managing WTs (and thus physical devices) remotely. Finally, the Data Analytics block gives meaning to the data: condition assessment and damage prediction techniques are performed.

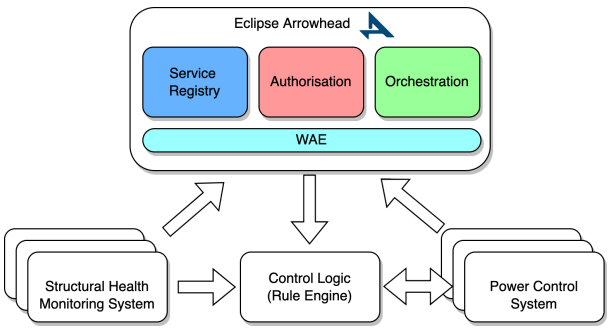

Fig. 2. High-level architecture of our proposed solution.

[4]https://github.com/arrowhead-f/application-skeleton-wot

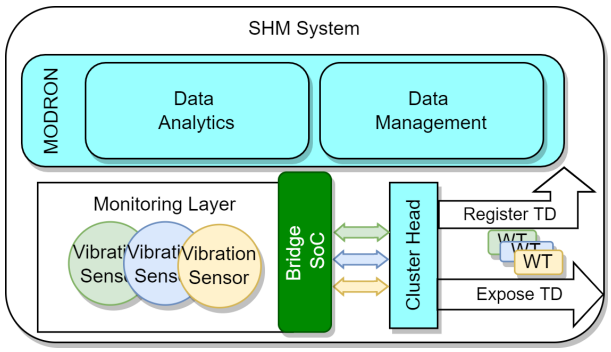

Fig. 3. Internal architecture of the SHM system used as the monitoring subsystem in the SHM loop.

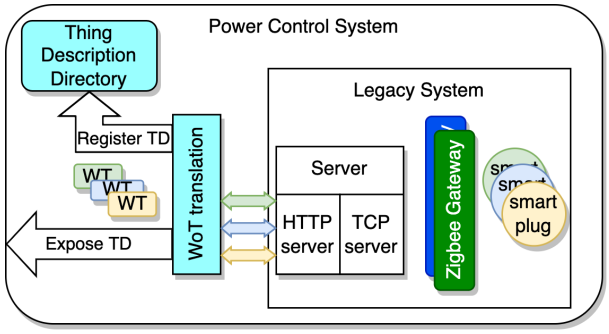

Fig. 4. Internal architecture of the power control system used as the actuation subsystem in the SHM loop.

B. Power control subsystem

In Figure 4, we show the architecture of the power control subsystem, which is used as actuation subsystem in the frame of the SHM loop. It is composed of a legacy system, a module for WoT translation and a TDD. The legacy system performs power monitoring and power control through a set of custom power plugs, which implement the behavior of smart circuit breakers. Power monitoring allows for real-time measurement of active and reactive power, voltage, current, whereas power control allows to remotely power an appliance on or off through a latching relay. The power plugs are designed for smart grid or smart home applications, and have a Bill Of Material (BOM) cost below 15$. The power plugs communicate with a ZigBee IP gateway whose BOM cost is 15$ as well. The gateway maintains a TCP connection with a server, exchanging sensor data and actuation commands. The server can run on a general-purpose computer and can connect to multiple gateways simultaneously. All the components of the legacy system use AES-128 for encryption, to secure exchanged data that can be sensitive and confidential. Finally, the server exposes an HTTP server to the user, allowing for data displaying and manual actuation through HTTP calls.

The WoT translation module (WTM) is responsible of making the legacy system accessible as a WT. It is implemented as a WoT servient and communicates with the legacy system's server via HTTP. It abstracts the legacy system by exposing one WT for each power plug. Each WT has four interaction affordances, i.e., three actions and one observable property. The actions are used to power the appliance on and off. More specifically, it is possible to power off an appliance in a hard or a soft way. The hard way powers off the appliance immediately. Instead, the soft way allows the WT to monitor power consumption of the appliance and to power it off only when power consumption is below a threshold, i.e., the appliance is in an idle state. Besides, these actions update the only exposed property, which is a boolean that tells whether the appliance is powered on or off. At startup, the WTM first connects to the HTTP server, retrieving the list of gateways and active power plugs. Then, it creates and exposes the TDs containing the required set of properties and actions. Finally, it registers the resulting TDs to a TDD. We chose LinkSmart for the implementation of the TDD. Whenever the WTM receives a request for an affordance of the exposed TDs, it will issue the proper HTTP call to the HTTP server.

IV. Implementation/Design of the SHM control loop

In this section, we describe the design and implementation of a control logic for an Industry 4.0 use case. This logic realizes a closed-loop interaction between the SHM platform, on the one hand, and the power control system, on the other.

```
{
  "triggers": [{
    "conditions": {
      "AND": [{
          "selector": "building01.floor02
              .room02.accl.*",
          "property": "Value",
          "operator": "gte",
          "values": [42]
        }, {
          "selector": "building01.floor02
              .*.plug.*",
          "property": "PoweredOn",
          "operator": "eq",
          "values": [true]
        }
      ]
    },
    "effects": [{
      "selector": "building01.floor02.*.
          plug.*",
      "affordanceType": "action",
      "affordanceName": "SoftPowerOff",
      "affordancePayload": {}
    }]
  }]
}
```

Listing 1. Example of a condition-effect rule in our system.

A. Syntax Language

We designed a simple syntax to describe rules useful for managing monitoring and prevention scenarios. This JSON descriptor allows us to define triggers having multiple conditions and effects. A condition describes the requirements needed to activate the trigger: the selector is a string, based on dot notation and wildcards, which allows identifying WTs located in a specific area of a structure; the `property` key is used to identify the property affordance to be observed, whereas `operator` and `values` are the parameters that define how the observed value needs to be evaluated. The conditions of a trigger can also be grouped through logical AND/OR operators. This way, it is possible to achieve a higher level of expressiveness critical for our use cases. The `effects` are executed one after the other when the trigger conditions occur. The `selector` behaves in the same way as the one used in the `conditions`, while properties like `affordanceType`, `affordanceName` and `affordancePayload` serve to identify and act on a specific interaction affordance that will modify the interal state of the WT. E.g., in the Listing 1, the specified conditions select all the accelerometers WTs that are located in room number 2 on floor 2 of building 1 and all the power plugs on the same floor. After that, the system starts observing these properties triggering the `effects` if the accelerometer values are greater (or equal) than 42 and the power plugs are on.

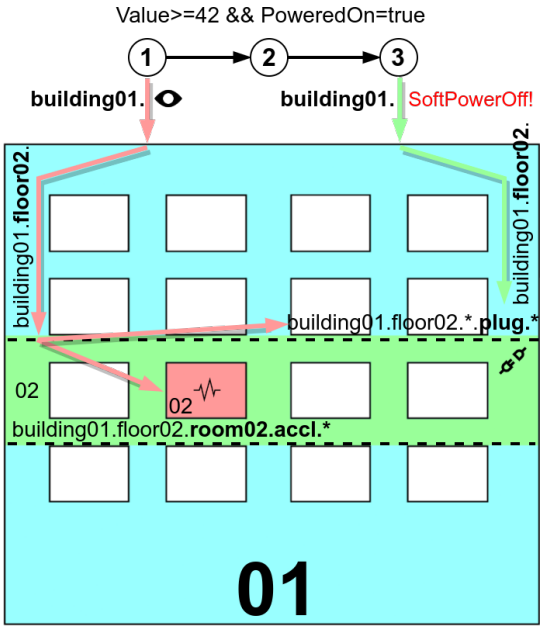

Fig. 5. Query Language

B. WoT Arrowhead Drawbridge

We refer to the proposed control logic as WoT Arrowhead Drawbridge (WAD). Our logic indeed bridges between two different ecosystems by triggering actions on one of them (i.e., the power control subsystem) if specific conditions are met in the other (i.e., the SHM subsystem). From an architectural point of view, the WAD is an Arrowhead consumer running within an Arrowhead local cloud.

When it first starts, the WAD is configured with a number of settings that flexibly allow it to adapt to the specific context and needs. The first type of setting are two Arrowhead Mirror Groups indicating respectively the group of vibration sensors to monitor and the group of power plugs to control. With this setting, it is possible to select the area of the production plant that needs to be monitored and controlled (e.g., the whole building, a floor, a room). The other setting are three threshold values – low, medium, and high – for the vibration samples, which are used by the WAD to trigger actions on the power control subsystem, as explained below.

The WAD observes the `Value` property of all the vibration WTs in order to periodically retrieve vibration samples. Besides, it observes the `PoweredOn` property of all the power WTs, with the aim to detect whether factory machines are powered on or off. The actual control logic of the WAD consists of condition-effect rules. These are defined using the JSON syntax presented above and can be dynamically updated through a RESTful API that the WAD exposes. The scenario that we implemented works as follows:

- If samples from all the vibration sensors in a room exceed the medium threshold and all the factory machines on the floor are powered on, the WAD assumes that there is a pre-alarm situation. For safety reasons, the WAD softly powers off all the factory machines on the floor. The WAD does so by invoking the `SoftPowerOff` action that is exposed by power WTs on the floor. This action also sets the `PoweredOn` property to ”false” when the machine is actually powered off. This condition-effect rule is defined in Listing 1 and depicted in Figure 5.
- If vibration samples from all the sensors in a room are above the high threshold (e.g., due to a major earthquake) and all the factory machines on the floor are powered on, the WAD immediately powers off those factory machines. This is done by invoking the `HardPowerOff` action, which also sets the `PoweredOn` property to ”false”.
- If instead vibration samples from a room are below the low threshold and all the factory machines on the floor are powered off, it means that the critical condition has terminated. Then, the WAD powers on all the factory machines on the floor by invoking the `PowerOn` action, which also sets the `PoweredOn` property back to ”true”.

V. PoC and Validation

In this section, we describe a PoC deployment that resembles the use case described in Section IV. The structure of the PoC is shown in Figure 6 and described in the following. The SHM subsystem consists of a high-rise grid structure with 12 accelerometers deployed on several points of the structure. The sensors are publishing bursts of data on a Cluster Head, which is deployed as a Raspberry Pi 3, hosting the WoT proxy of each of the sensors. MODRON runs on a Qotom mini-PC (Quad-core Intel Celeron CPU at 1.99GHz, 8GB of RAM, and 58GB of disk, running Ubuntu). MODRON acts as a

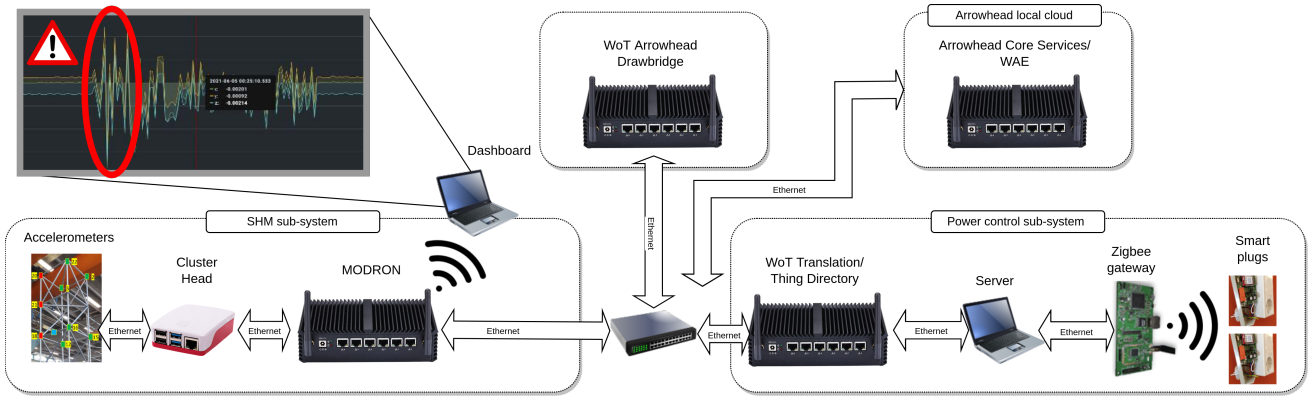

Fig. 6. Actual deployment of the use case in all its components, with specific focus on the physical appliances used.

Webserver for the Grafana dashboard that is visualized through a laptop. The power control subsystem instead is deployed as follows. Two custom power plugs, described in [10], are connected to a custom zigbee gateway powered by an ARM Cortex M4 processor with hardware encryption. The server controlling the power plugs and running the HTTP-server backend is a Sony Vaio PCG-71311M laptop with an Intel i5-450M CPU at 2.4GHz, 4GB of RAM, and 256GB of disk, running CentOS 6.3 and Linux kernel 2.6.32. A Qotom mini-PC represents a local cloud node, running both the WTM and the TDD. The WTM is developed using the Thingweb node-wot framework, which is the WoT implementation in Node.js. The TDD is the open-source LinkSmart Thing Directory[5]. The Arrowhead local cloud is again deployed on a Qotom mini-PC. We used the official Java Arrowhead Core Systems version 4.4.0[6] and we run the WAE in the same host as a separate module[7]. Finally, the control logic is deployed on a fourth Qotom mini-PC. We used the above PoC deployment to validate the integration of the various subsystem composing the use case. Moreover, we verified the correct execution of the WAD functions, both using synthetic inputs and with real data coming from the accelerometers, also exploiting the graphical interface to assess the behavior of the logic.

VI. Conclusions

In this paper we integrated two heterogeneous IoT ecosystems in an industrial scenario by leveraging both the WoT standard and the AHF. We first proposed a high-level architecture for integrating separate SoS that either are monitoring systems or actuation systems by introducing a rule-based control logic in between. Next, we implemented an exemplary control logic, the WAD, for a SHM use case where parties adopt the WoT standard and can discover each other by means of the AHF, and we defined the configuration syntax of the WAD. Finally, we deployed the proposed system in a real scenario where an SHM monitoring ecosystems causes a power plug actuation ecosystem to react promptly to emergencies, thus demonstrating feasibility of our approach. As a future work, we plan to carry out an extensive performance evaluation of the system, to assess its scalability and feasibility in realistic deployments.

[5]https://github.com/linksmart/thing-directory, last accessed Jan 2022
[6]https://github.com/eclipse-arrowhead/core-java-spring
[7]https://github.com/arrowhead-f/application-skeleton-wot

Acknowledgment

This work was partially supported by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence), and by the EU ECSEL Joint Undertaking under grant agreement No 826452 (Arrowhead Tools), supported by the European Union Horizon 2020 research and innovation programme.

References

[1] F. Tao, Q. Qi, L. Wang, and A. Nee, "Digital twins and cyber–physical systems toward smart manufacturing and industry 4.0: Correlation and comparison," *Engineering*, vol. 5, no. 4, pp. 653–661, 2019.
[2] A. Theorin, K. Bengtsson, J. Provost, M. Lieder, C. Johnsson, T. Lundholm, and B. Lennartson, "An event-driven manufacturing information system architecture for industry 4.0," *International journal of production research*, vol. 55, no. 5, pp. 1297–1311, 2017.
[3] 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)*, 2018, pp. 743–748.
[4] J. Delsing, *Iot automation: Arrowhead framework*. Crc Press, 2017.
[5] I. Zyrianoff, L. Gigli, F. Montori, C. Kamienski, and M. Di Felice, "Two-way integration of service-oriented systems-of-systems with the web of things," in *IECON 2021–47th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2021, pp. 1–6.
[6] 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)*. IEEE, 2018, pp. 755–760.
[7] C. Paniagua, J. Eliasson, and J. Delsing, "Efficient device-to-device service invocation using arrowhead orchestration," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 429–439, 2019.
[8] M. Kovatsch, R. Matsukura, M. Lagally, T. Kawaguchi, K. Toumura, and K. Kajimoto, "Web of things (wot) architecture," W3C Recommendation, Apr. 2020, https://www.w3.org/TR/wot-architecture/.
[9] C. Aguzzi, L. Gigli, L. Sciullo, A. Trotta, F. Zonzini, L. De Marchi, M. Di Felice, A. Marzani, and T. S. Cinotti, "Modron: A scalable and interoperable web of things platform for structural health monitoring," in *2021 IEEE 18th Annual Consumer Communications Networking Conference (CCNC)*, 2021, pp. 1–7.
[10] E. Spanò, L. Niccolini, S. Di Pascoli, and G. Iannaccone, "Last-meter smart grid embedded in an internet-of-things platform," *IEEE Transactions on Smart Grid*, vol. 6, no. 1, pp. 468–476, 2015.