ARCHIVIO ISTITUZIONALE DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

Discovering Web Things as Services within the Arrowhead Framework

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Sciullo L., Montori F., Trotta A., Di Felice M., Cinotti T.S. (2020). Discovering Web Things as Services within the Arrowhead Framework. Institute of Electrical and Electronics Engineers Inc. [10.1109/ICPS48405.2020.9274694].

Availability:
This version is available at: https://hdl.handle.net/11585/801018 since: 2021-04-09

Published:
DOI: http://doi.org/10.1109/ICPS48405.2020.9274694

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

24 December 2024

This is the post peer-review accepted manuscript of:

L. Sciullo, F. Montori, A. Trotta, M. Di Felice and T. Salmon Cinotti, "Discovering Web Things as Services within the Arrowhead Framework," 2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS), Tampere, Finland, 2020, pp. 571-576, doi: 10.1109/ICPS48405.2020.9274694.

The published version is available online at:
`https://doi.org/10.1109/ICPS48405.2020.9274694`

© 2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works

Discovering Web Things as Services within the Arrowhead Framework

Luca Sciullo*, Federico Montori*, Angelo Trotta*, Marco Di Felice*†, Tullio Salmon Cinotti*†

* Department of Computer Science and Engineering, University of Bologna, Italy
†Advanced Research Center on Electronic Systems "Ercole de Castro", Italy
Email: {luca.sciullo, federico.montori2, angelo.trotta5, marco.difelice3, tullio.salmoncinotti}@unibo.it

***Abstract*—The IoT is spreading heavily around us and many new standard at different architectural layers proliferate. This creates a heavy interoperability gap that many architectural proposals and new standards have tried to cope with over the years. In particular, we take into account the Arrowhead interoperability Framework, a powerful and promising paradigm for interconnecting local clouds and we demonstrate the possibility to integrate it with the emerging W3C Web of Things standard. We study the usage of the Arrowhead Framework as a potential candidate for discovery operation and inclusion of WoT ecosystems within Service Oriented Architectures (SOA), in order to support the integration of legacy and proprietary systems. To this aim, a three-layered architecture is proposed and validated experimentally on a proof-of-concept use case, by highlighting the advantages of the proposed integration.**

***Index Terms*—Arrowhead framework, Web of Things, interoperability, platform integration.**

I. Introduction

The Internet of Things (IoT) is a paradigm that is revolutionizing heavily the world we live in. The number of connected devices already overtook the world population, and new integration paradigms are proposed on a monthly basis. As a matter of fact, interoperability is one of the toughest challenges that has been raised in the last decade. Many interoperability platforms have been proposed for different uses cases [Di Martino et al.(2018)Di Martino, Rak, Ficco, Esposito, Maisto, and Nacchia]. However, to fully unleash the power of joining different sources of information, sometimes the challenge shifts to connecting such clouds together. In fact, not rarely IoT clouds tend to behave as closed islands with little or no interoperability with the outside world. The Arrowhead Framework [Delsing(2017)] is an interoperability Service Oriented Architecture (SOA), proposed within the Arrowhead European project, which interconnects several IoT-based local clouds. It has been proposed for industrial automation, however, it fits many more scenarios by now and has been used extensively to interconnect worlds of different domains.

In this paper we analyze a possible integration between the Arrowhead Framework and the W3C Web of Things (WoT) paradigm, which is a standardization effort for the IoT undertaken by the World Wide Web Consortium (W3C) over the last years [W3C Working Group(2019a)]. In particular, the integration is shown through a proposed three-layered architecture in which a standalone WoT ecosystem is integrated within an Arrowhead local cloud. We chose the WoT because it is, nowadays, the most promising standardization effort within the application layer of the IoT, however there can be many cases in which legacy, obsolete or even proprietary systems need an ad-hoc integration with such world. For such reason, we envision the Arrowhead Framework as the candidate architecture for filling such gaps, as well as maintaining properties such as late binding and loose coupling between services. Indeed, while the conceptual similarity between the two paradigm allows an agile integration, the potential of compliant applications gets significantly boosted.

More in detail, the contributions of this study can be summarized by the following three points:

- We propose a three-layered architecture thanks to which clients compatible with both the Arrowhead Framework and the W3C WoT will be able to interact with the IoT devices.
- We design an essential middleware component, defined as WAE, that acts as a discovery bridge for the WoT layer.
- We validate the proposed architecture providing performance evaluation in a real world scenario in which a multitude of Web Things is instantiated and published to the main service broker.

The paper is then structured as follows: Section II provides the basics of the Arrowhead Framework and the W3C Web of Things, as well as the related works in literature, Section III defines the high-level architecture and its components, while Section IV focuses on the interaction of the components of the architecture, Section V shows details about the implementation, Section VI provides the details about our validation experiment and, finally, Section VII concludes the paper.

II. Background

A. Arrowhead Framework

Over the last decades, ecosystems revolving around IoT in its various facets have shown the common trend of shifting from monolithic or ad-hoc deployments to architectures in which each entity is responsible for producing or consuming services, as in any Service-Oriented Architecture. The Arrowhead Framework (AHF)[1] is the result of an effort of more than 80 European partners [Delsing(2017)] and has been used

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

extensively in several other EU initiatives such as Productive 4.0[2], Far-Edge[3] and Arrowhead Tools[4].

The Framework, designed for supporting IoT automation scenarios at any application level, is based on the key guidelines that characterize a SOA: late binding, loose coupling, and lookup [Breivold and Larsson(2007)]. More in detail, each System of Systems (SoS) based on the AHF is deployed in connected local clouds, each of them managing their internal services and communicating with each other in a non-hierarchical structure, therefore separating responsibilities while still guarantee interoperability. Each local cloud hosts several Systems, defined as the software components that interact with each other and shape the application workflow. Systems can expose a number of Services as well as consume other services in the network, they are indeed defined as Service Providers or Service Consumers (clearly any system can be both). The interaction between systems and services within each local cloud is given by the "Core Systems" (CS) – one instance is deployed per local cloud – that support and orchestrate the exchange of information. They are divided into "Mandatory" CS, which have to be deployed within a local cloud to make it Arrowhead-compatible, and "Support CS" [Varga et al.(2017)Varga, Blomstedt, Ferreira, Eliasson, Johansson, Delsing, and de Soria]. Mandatory CS are described in detail below:

- The **Service Registry** system is responsible for the registration of each service within the local cloud. It acts as a repository, against which other systems can perform a service lookup – i.e. a discovery operation – in order to obtain the information and the endpoint of the service they are looking for. In the last version of the CS (4.1.3 at the time of writing), the service lookup is performed through HTTP REST calls.
- The **Authorization** system is responsible for the correct interaction between producers and consumers according to their rights. It manages the correct authentication of providers and consumers as well as their authorization for consuming or producing resources based on a set of rules that can be added or modified by the cloud manager.
- The **Orchestration** system is responsible for coordinating the interactions between systems freeing the consumers from the burden of establishing their preferences at design time. With the Orchestration system, the Service Provider that is best suitable for the consumer's request can be chosen dynamically based on a list of orchestration rules about the type of service requested. This can potentially handle cases of faults and perform load balancing.

Support CS are not mandatory and can be included in any local cloud where needed. Examples of available Support CS are: QoS Manager, Translator System, Event Handler and Configuration Manager. Furthermore, the Gatekeeper System and the Gateway System, which are still Support CS, are devoted to the inter-cloud communication, mediating the exchange of, respectively, lookup requests and chunks of data [Varga and Hegedus(2015)]. The latest version of all the AHF components (4.1.3 at the time of writing) can be found in [Arrowhead Consortia()].

[2]https://productive40.eu/

[3]http://faredge.eu/#/

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

B. Web of Things

The chaotic world of the Internet of Things is characterized by tens of different technologies, protocols, and architectures for interconnecting Smart Things to the Internet. Because of its fragmented nature, one of the biggest challenge of the IoT landscape is constituted by the lack of interoperability. For this reason, starting from 2015, several universities and companies in the WoT ecosystem seamlessly joined the W3C working group for the definition of a Web of Things (WoT) standard, whose goal is claimed to counter the fragmentation of the IoT, by defining a reference architecture, the communication patterns and the interfaces of the Things; the rationale is to enable the interoperability among IoT systems, regardless of the underlying stack implementation and of the networking technologies being used. In particular, the W3C WoT Architecture is based on four main blocks:

- **Web Thing**: a Web Thing, referred also to as simply Thing, is defined as whatever entity can be semantically described. More precisely, and reporting the working group's words: a Thing is "an abstraction of a physical or a virtual entity whose metadata and interfaces are described by a WoT Thing Description, whereas a virtual entity is the composition of one or more Things." [W3C Working Group(2019a)]. A Thing can be a device, a logical component of a device, a local hardware component, or even a logical entity such as a location (e.g., room or building).
- **WoT Thing Description (TD)**: the Thing Description is the structured data that defines a Thing, like its metadata, together with its Interaction Affordances and its links to other Things. By default it is serialized by using the JSON-LD language and follows the *Property, Action, and Event* paradigm, i.e., it maps each state variable of the Thing to a Property, each commands that can be invoked on the Thing to an action, and each notification fired by the Thing to an event.
- **WoT Scripting API**: despite this is an optional block, the Scripting APIs provide a standard way for writing Applications that define the Thing Behaviour. Thanks also to a scripting runtime system (*Wot Runtime*), the TAs are portable, i.e. they can be easily re-used and moved on different Things.
- **WoT Binding Templates**: the Binding Templates are a collection of metadata that describe the strategies of communication offered by a Thing, like for instance: Machine-to-Machine (M2M) through CoAP with TLS security mechanism and CBOR as Content Type.

C. Related Works

The Web of Things paradigm presents nowadays the challenge of implementing a discovery operation of Web Things that supports matching and lookup as well as orchestration, for which several proposals have been presented in the literature [Zhou et al.(2016)Zhou, De, Wang, and Moessner]. In fact, Thing Discovery has been repeatedly claimed as a desirable feature to cope with several problems, for instance, the mobility of Things [Kamilaris and Ali(2016)], although, in the very last months, progresses have been made within the W3C as a new discovery paradigm – we will use it in the form of a Thing Directory – is starting to build within the standard. It is still quite embryonic, in fact, many unofficial works have been produced in the meantime. A recent paper [Sciullo et al.(2020)Sciullo, Gigli, Trotta, and Di Felice] proposed an ecosystem for managing resources and applications in the context of the W3C WoT, including a proof-of-concept discovery mechanism. Among the other discovery implementations for generic WoT approaches, we cite the Dyser framework [Romer et al.(2010)Romer, Ostermaier, Mattern, Fahrmair, and Kellerer] and the WOTSF system [Younan et al.(2016)Younan, Khattab, and Bahgat], although they only focus on the lookup part, while no matching mechanism is implemented.

In industrial and generic IoT scenarios, a strong integration capability is required. The Arrowhead Framework has been applied in a wide variety of cases and, in many of them, it has been integrated with other IoT frameworks. In [Lam and Haugen(2019)] the authors survey the potentials of semantics in the IoT and propose the integration of the Semantic Web of Things (SWoT), proposed in [Noura et al.(2019)Noura, Gyrard, Heil, and Gaedke], with the Arrowhead Framework. In other cases, protocol translation via a proxy has been the predominant solution: the work in [Palm et al.(2019)Palm, Paniagua, Bodin, and Schelén] is on the conversion of message payloads and headers the work in [Campos-Rebelo et al.(2019)Campos-Rebelo, Moutinho, Paiva, and Maló] is focused on the XSLT rule-based data transformation and the work in [D'Elia et al.(2016)D'Elia, Viola, Montori, Azzoni, and Maiero] envisions the integration with a Kura-based architecture for the implementation of a real world electromobility scenario.

III. Architectural Design

In this section we define in detail the architectural structure of our proposal for integrating Web Things within the Arrowhead Framework. Despite the great potential of WoT paradigm, since the process of making an already existing service W3C WoT-compliant requires some effort, there can still be cases - especially those involving old legacy systems - in which a service could be interested to benefit from capabilities offered by Web Things, without joining the WoT ecosystem. In particular, our proposal is specifically designed for such components that may either be unable to understand the same language and use the same protocols as the WoT ecosystem does, or be unaware of the location of the Web Things that need to be queried. More in detail, we envision an ecosystem in which a set of Web Things, which are devoted to collect data through sensors, expose their data to potential consumers that are external to their ecosystem, in particular:

- A consumer that is able to communicate using the WoT standard and the same communication protocol the Web Things are using, however it does not know how to reach the Web Things endpoint.
- A consumer that has no information about the Web Things endpoint and communicates only using another legacy protocol (say, HTTP).

In order for the whole ecosystem or, as it is defined in the Arrowhead official documentation model[5], the System of Systems (SoS) to be able to cope with such cases, we propose an architecture in which each Web Thing is also a Service Provider of an Arrowhead local cloud, therefore exposing sensor data as a service. The Arrowhead Framework allows indeed each service to be discoverable through its main component: the Service Registry. As anticipated in Section II-A, the Service Registry acts as the main service broker in a SOA: for each service in the local cloud that advertises its endpoint through a publish call, it keeps in memory a service record, encoded in JSON, that includes a set of service metadata. The record includes the type of service, its endpoint and the protocol used, although other metadata can be added upon need. In a typical and simple scenario, a Service Provider first publishes its service record, then a Service Consumer willing to consume such service performs a service lookup call against the Service Registry and obtains information about the endpoint and the protocol of the desired service. Once this information is held by the Service Consumer, the communication with the Service Registry is no longer needed and the Service Provider and the Service Consumer can communicate directly.

Note that in an Arrowhead service consumption the Orchestration module also has its part: it gets queried by the Service Consumer for a service of a defined type and it searches the Service Registry for the most suitable service record, based on a set of rules. The use of the Orchestration service is out of the scope of this paper, therefore we intentionally simplify the interaction bypassing the Orchestration and only demonstrating the architectural integration.

IV. Service Interactions

In order to support different types of external consumers interacting with the WoT ecosystem, we propose the layered architecture in Figure 1. The architecture consists of three conceptual layers: the Physical layer, the WoT Layer and the Arrowhead Layer. Entities on each layer can communicate directly with other entities belonging to the same layer as they are assumed to use the same application protocol. For different layers, instead, entities either have an abstraction or a inter-layer communication channel, as will be explained later. On the bottom-left corner we depicted the physical

[5]https://www.arrowhead.eu/arrowheadframework/this-is-it/documentation-model/

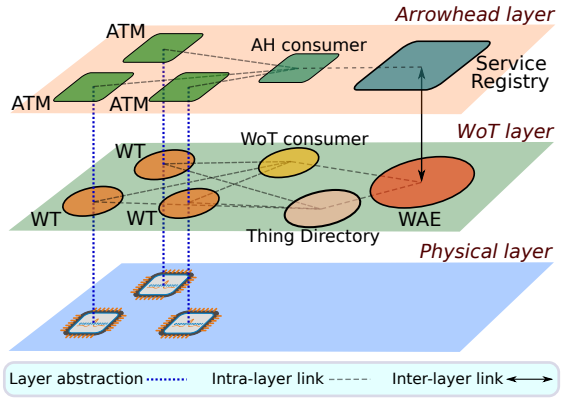


Fig. 1. The System Architecture

sensors, the only entity of the physical layer. Sensors can be of any type as long as they produce a numerical observation from the real world. Each sensor gets abstracted onto a Web Thing, according to the WoT paradigm. Each Web Thing is then registered onto a special Thing Directory, a standard registry for the WoT ecosystem, as specified in [W3C Working Group(2019a)]. The central component of the WoT Layer, the WoT Arrowhead Enabler (WAE), can be classified as a WoT Mashup Application, a concept recently outlined in [Sciullo et al.(2020)Sciullo, Gigli, Trotta, and Di Felice]. In detail, it periodically queries the Thing Directory to detect new Web Things right after they spawn (i.e. the binding with the actual sensor is generated). As new Web Things are detected, the WAE performs a publish operation for each of them against the Service Registry in the Arrowhead Layer to publish Web Things as new Arrowhead services. The communication between the WAE and the SR is the sole case of inter-layer communication channel, in which a component (in this case the WAE) acts as a proxy able to use two different communication protocols. Furthermore, each Web Thing is extended onto the Arrowhead Layer by a new module, called Arrowhead Thing Mirror (ATM). The ATM exposes the Web Thing service endpoint as an HTTP Web Service in the Arrowhead local cloud. Note that a Web Thing and its relative ATM can run on the same piece of software as well as in separate components connected by a custom communication link. The ATM plays, to some extent, the role of an Arrowhead service adapter, however, it does not perform publish operation, as they are handled by the WAE.

The record published by the WAE exposes by default the endpoint and the metadata of the ATM related to the Thing, while the JSON-LD description of the Thing at the WoT Layer is converted to a string and encapsulated in the newly created "TD" subfield of the "metadata" JSON field of the service record. This way, a consumer can interact with the Web Thing in two ways, depending on its communication capabilities:

- An HTTP-enabled Consumer queries the Service Registry, selects the service that provides the type of data needed and gets the endpoint of the service, which corresponds to the endpoint of the related ATM. The Consumer then performs the consume calls against the service offered by the ATM which, in turn, queries the Web Thing and retrieves the data point. Response data travels then backwards to the Consumer.
- A WoT-enabled Consumer queries the WAE, which retrieves the list services from the Service Registry. As the consumer is only able to interact with WoT-enabled systems, the WAE decapsulates the related TD from the field "metadata" of the service record and sends it back to the consumer. The latter is then able to select a Web Thing among the received ones and query it directly; the Web Thing endpoint is enclosed in the TD itself, thus the actual endpoint, which belongs to the ATM, is ignored.

The whole interaction for the two types of consumers is shown in detail through the sequence diagram in Figure 2. In particular, it shows mainly two patterns:

1) Web Things' publication on SR: when a Web Thing is generated, it automatically instantiates an ATM to be able to fulfill a request coming from an AH Consumer. At the same time, the Web Thing publishes itself on the Thing Directory, according to the draft of the W3C standard proposal. The WAE is in charge of keep checking if new Web Things appear on the Thing Directory, and in case to publish themselves on the SR. This can be achieved in two ways, depending on the WAE implementation: either the WAE polls the Thing Directory or the WAE is implemented as a Web Thing, so it can subscribe to Thing Directory's events. The SR is listening from queries of services' consumers and replies with all the Services is aware of, including the Arrowhead ones that however are not shown in this diagram.

2) Web Things' consuming: once the services related to Web Things become available on the SR, they can be queried by consumers. Depending on the kind of consumer, there are different interactions. The easiest case is the one involving an AH Consumer, since it has only to query the SR, to get the list of services and to directly interact with them through their ATM. A Wot Consumer instead requires more steps in order to be able to retrieve the Thing Description of the Web Thing that should be consumed. First, it has to send the query to the WAE, since it might not be able to speak directly to the SR. Hence, the WAE forwards the request to the SR and waits for the list of services that match the query originally coming from the WoT Consumer. Finally, WAE returns the list to the WoT Consumer, that is now able to *consume* the Web Thing and to interact with it, for instance by invoking one of its actions.

V. Implementation

We here briefly describe the implementation of the main components of our architecture. The Service Registry is a JAVA server that exposes some REST APIs. In particular, we used the API already available as open source project [Arrowhead Consortia()]. All Web Things involved in the scenario have been implemented and instantiated by using *node-wot* [W3C Working Group(2019b)], the official W3C framework for the WoT. The WAE component has been designed as a Web Thing - for being able to natively speak

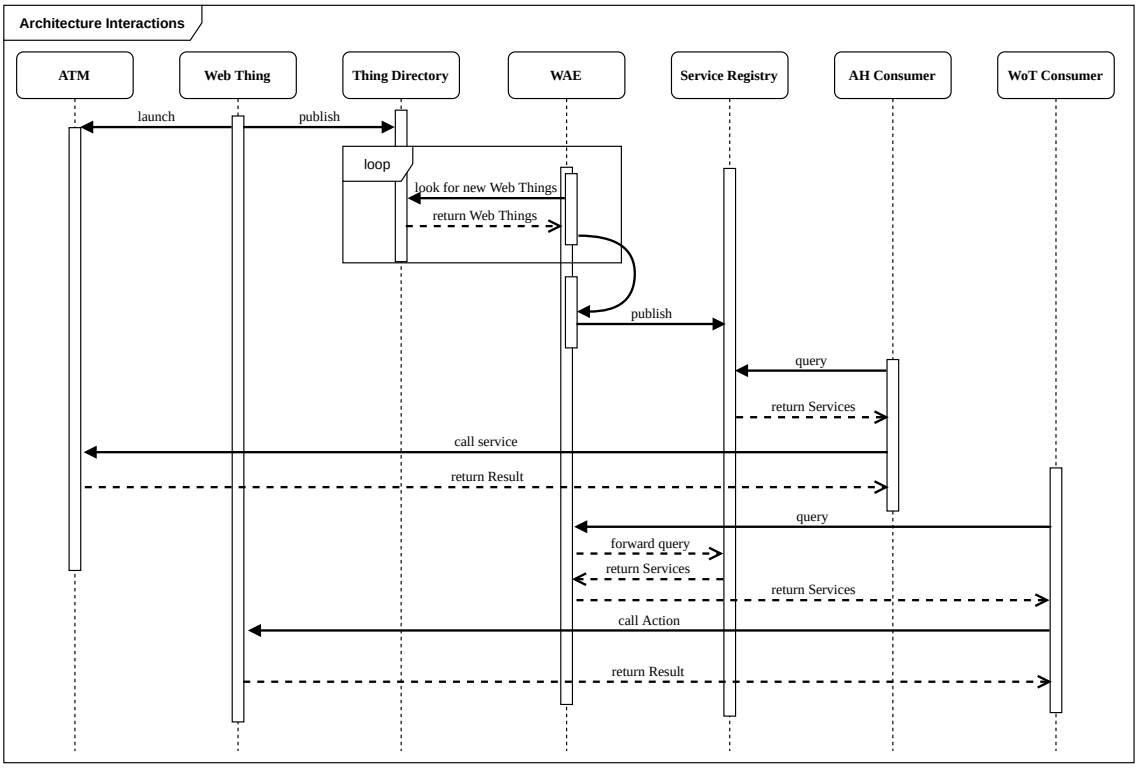


Fig. 2. Sequence diagram presenting the interactions of all the components of our three-layer architecture

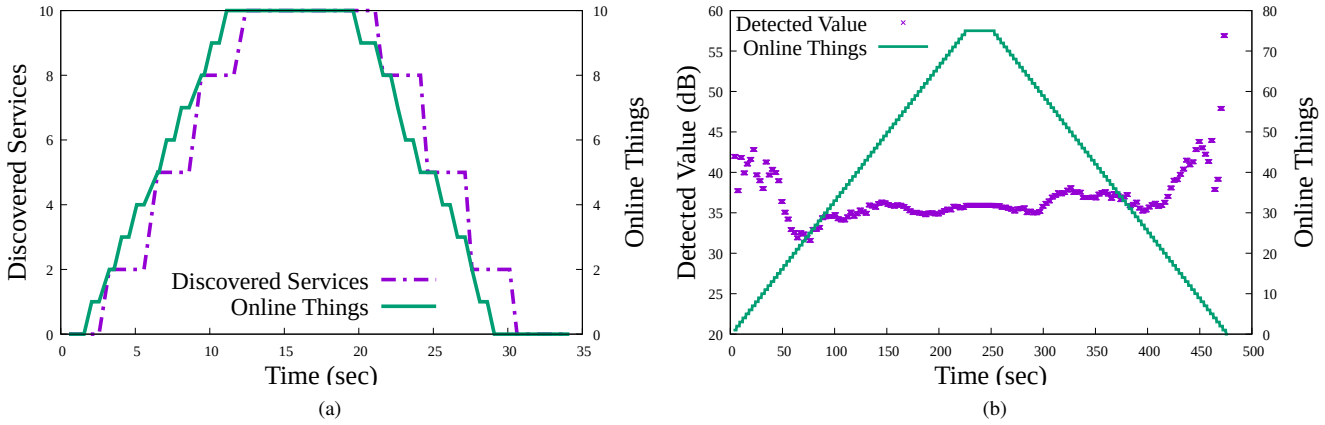
TABLE I
LIST OF PROPERTIES, ACTIONS, AND EVENTS OF THE WAE THING.

Name	Type	Description
listOfWebThings	Property	List of all the Web Things the WAE is aware of.
startCrawling	Action	Start to look for new Web Things that are published on the TD.
query	Action	Forward the query of a Wot Consumer to the WAE.
newWebThing	Event	This event is fired when a new Web Thing has been discovered on the TD by the WAE.

to other W3C WoT entities - and as an HTTP client - in order to use the SR's APIs. As shown in table I, following the paradigm *Properties, Action, Events* as explained in II-B, the WAE Web Thing exposes the *listOfWebThings* Property for listing all the already known Web Things it has published. Additionally, it exposes also the *startCrawling* and the *query* actions. The first is automatically invoked once the WAE has been deployed to look for new Web Things that have been published on the Thing Directory. The second one is invoked by a WoT Consumer in order to query the SR and to get the list of services that match its request. Finally, a generic event *newWebThing* is fired each time new Web Things have been discovered by the WAE. Both the HTTP client of WAE and the AH Consumer have been customized for our need by taking advantage of the already existing open source NodeJS Arrowhead Client [Arrowhead Consortia()]. Each WoT Consumer is a *Mashup Application*, i.e., a javascript application that uses *node-wot* framework as a library and simply consumes multiple Things to interact with them in order to collect data and manipulate it for its needs. Lastly, the ATM is an *ExpressJS* web server that maps each Web Thing Affordance to a REST API and that uses the *node-wot* as a library behind the scenes in order to interact with the Web Thing it represents.

VI. VALIDATION

In order to validate our proposal, we created a proof-of-concept scenario where we deployed the components of the architecture described in section III. The goal of our validation is dual: first, we want to prove the functionalities of the Arrowhead discovery in conjunction with the WoT ecosystem. Secondly, we want to show the benefits of such discovery method for a generic Consumer that is agnostic of the real nature of the services used for its application. In particular, we instantiated a WoT Arrowhead Enabler (WAE) which is in charge of discovering new Web Things and of publishing them on the Arrowhead SR to make them available to all the possible consumers. Each new Web Thing is launched after λ seconds from the previous one, and then published as soon as it has been discovered by the WAE. Additionally, after a pre-defined $TIME_{MAX}$ interval of time, Web Things start disconnecting every λ seconds and so they are unregistered from the SR. This means that, depending on the WAE's Update Frequency (WUP), there could be some delay before Web Things become available/unavailable on the SR. Figure 3(a) shows the total number of services made available on the



(a) (b)

Fig. 3. Figure 3(a) shows the Online Things vs Discovered Services, while Figure 3(b) shows the mean detected value over all the sensors.

SR and the number of Web Things online. In this case, since $\lambda = 1$ second and $WUP = 3$ seconds, the plot shows the misalignment between the online Web Things and the ones registered as Arrowhead services in the Arrowhead SR. Both Web Things and their correspondent Arrowhead services increase until $TIME_{MAX}$ is reached and then they start decreasing accordingly respectively to λ and WUP frequencies. While in the first case we focused on validating the Thing Discovery, in the second case we set up a testbed for validating a Consumer entity. In particular, we instantiated an Arrowhead consumer that retrieves the values from some services and elaborate them. More in details, the services are WoT services that return the value of acoustic sensors, with an error estimated of $\pm 5dB$, while the application's goal is to detect the walk of a human inside a room. The application first queries the Arrowhead SR for looking for all the available sensors, then it retrieves the detected values and computes the average of the value obtained. Depending on the use case, a threshold can be set for identifying a particular feature. Figure 3(b) shows the behaviour of the application over the time, with the number of Web Thing Services that changes over the time. For this testbed, the parameters are set to: $\lambda = 3$ seconds, $WUP = 1$ second, and $TIME_{MAX} = 250$ seconds. It is clear that the more WoT Things services contribute to the detection phase, and the more the average gets closer to 35 dB, that is a reasonable sound level for the human walking.

VII. Conclusion

In this paper we have presented the integration of a W3C Web of Things (WoT) ecosystem within the Arrowhead interoperability Framework, in order to support a much larger number of applications and make possible the interaction between the state-of-the-art WoT-enabled systems and services and legacy ecosystems. This is an important achievement as it also presents an alternative form of discovery for the W3C WoT that includes hybrid interactions with other components and, thus, maintaining the concepts of loose coupling and late binding even outside the single frameworks. The integration presented in this paper is not the only possible way of making such systems interact, in fact, we have shown the inclusion of a WoT system within the Arrowhead Framework, however, the dual integration could still be possible. This is object of study currently and will be further analyzed as future work. Also, we plan to extend the experimental validation by considering real-world condition monitoring use-cases where the integration of heterogeneous things/services is of paramount importance.

Acknowledgements

This research is funded by ECSEL, the *Electronic Components and Systems for European Leadership Joint Undertaking* under grant agreement No 826452 (Arrowhead Tools), supported by the European Union Horizon 2020 research and innovation programme and by the member states.

References

[Di Martino et al.(2018)Di Martino, Rak, Ficco, Esposito, Maisto, and Nacchia] B Di Martino, M Rak, M Ficco, A Esposito, SA Maisto, and S Nacchia. Internet of things reference architectures, security and interoperability: A survey. *Internet of Things*, 1:99–112, 2018.

[Delsing(2017)] Jerker Delsing. *Iot automation: Arrowhead framework*. CRC Press, 2017.

[W3C Working Group(2019a)] W3C Working Group. WoT Reference Architecture (Proposed Recommendation 30 January 2020), 2019a. URL http://www.w3.org/TR/wot-architecture/.

[Breivold and Larsson(2007)] Hongyu Pei Breivold and Magnus Larsson. Component-based and service-oriented software engineering: Key concepts and principles. In *33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007)*, pages 13–20. IEEE, 2007.

[Varga et al.(2017)Varga, Blomstedt, Ferreira, Eliasson, Johansson, Delsing, and de Soria] Pal Varga, Fredrik Blomstedt, Luis Lino Ferreira, Jens Eliasson, Mats Johansson, Jerker Delsing, and Iker Martínez de Soria. Making system of systems interoperable–the core components of the arrowhead framework. *Journal of Network and Computer Applications*, 81:85–95, 2017.

[Varga and Hegedus(2015)] Pál Varga and Csaba Hegedus. Service interaction through gateways for inter-cloud collaboration within the arrowhead framework. *5th IEEE WirelessVitae, Hyderabad, India*, 2015.

[Arrowhead Consortia()] Arrowhead Consortia. Arrowhead Framework official repository. URL https://github.com/arrowhead-f.

[Zhou et al.(2016)Zhou, De, Wang, and Moessner] Yuchao Zhou, Suparna De, Wei Wang, and Klaus Moessner. Search techniques for the web of things: A taxonomy and survey. *Sensors*, 16(5), 2016.

[Kamilaris and Ali(2016)] Andreas Kamilaris and Muhammad Intizar Ali. Do "web of things platforms" truly follow the web of things? In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 496–501. IEEE, 2016.

[Sciullo et al.(2020)Sciullo, Gigli, Trotta, and Di Felice] Luca Sciullo, Lorenzo Gigli, Angelo Trotta, and Marco Di Felice. Wot store: Managing resources and applications on the web of things. *Internet of Things*, page 100164, 2020.

[Romer et al.(2010)Romer, Ostermaier, Mattern, Fahrmair, and Kellerer] Kay Romer, Benedikt Ostermaier, Friedemann Mattern, Michael Fahrmair, and Wolfgang Kellerer. Real-time search for real-world entities: A survey. *Proceedings of the IEEE*, 98(11):1887–1902, 2010.

[Younan et al.(2016)Younan, Khattab, and Bahgat] Mina Younan, Sherif Khattab, and Reem Bahgat. Wotsf: A framework for searching in the web of things. In *Proceedings of the 10th International Conference on Informatics and Systems*, pages 278–285, 2016.

[Lam and Haugen(2019)] An Ngoc Lam and Øystein Haugen. Applying semantics into service-oriented iot framework. In *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, volume 1, pages 206–213. IEEE, 2019.

[Noura et al.(2019)Noura, Gyrard, Heil, and Gaedke] Mahda Noura, Amelie Gyrard, Sebastian Heil, and Martin Gaedke. Automatic knowledge extraction to build semantic web of things applications. *IEEE Internet of Things Journal*, 6(5):8447–8454, 2019.

[Palm et al.(2019)Palm, Paniagua, Bodin, and Schelén] Emanuel Palm, Cristina Paniagua, Ulf Bodin, and Olov Schelén. Syntactic translation of message payloads between at least partially equivalent encodings. In *2019 IEEE International Conference on Industrial Technology (ICIT)*, volume 88, 2019.

[Campos-Rebelo et al.(2019)Campos-Rebelo, Moutinho, Paiva, and Maló] Rogerio Campos-Rebelo, Filipe Moutinho, Luís Paiva, and Pedro Maló. Annotation rules for xml schemas with grouped semantic annotations. In *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 5469–5474. IEEE, 2019.

[D'Elia et al.(2016)D'Elia, Viola, Montori, Azzoni, and Maiero] Alfredo D'Elia, Fabio Viola, Federico Montori, Paolo Azzoni, and Matteo Maiero. Electro mobility automation through the arrowhead framework. In *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 5246–5252. IEEE, 2016.

[W3C Working Group(2019b)] W3C Working Group. Eclipse Thingweb node-wot. Source repository, 2019b. URL https://github.com/eclipse/thingweb.node-wot.