Interoperability in Open IoT Platforms: WoT-FIWARE Comparison and Integration

Interoperability in Open IoT Platforms: WoT-FIWARE Comparison and Integration

Ivan Zyrianoff*, Alexandre Heideker †,Luca Sciullo*, Carlos Kamienski †, Marco Di Felice*,
* Department of Computer Science and Engineering, University of Bologna, Italy
† Federal University of ABC, Santo André, Brazil
{ivandimitry.ribeiro, marco.difelice3, luca.sciullo}@unibo.it, {cak, alexandre.heideker}@ufabc.edu.br

***Abstract*—The rapid and exponential growth of the Internet of Things (IoT) has been generating a new breed of technologies that introduce several different protocols and interfaces. The Web of Things (WoT) architecture stands out as an emerging and potential solution to improve interoperability across IoT platforms by describing well-defined software interfaces. However, few studies analyze and compare WoT to other interoperability solutions proposed in the IoT literature. In this paper, we attempt to bridge the gap by three main contributions. First, we qualitative compare the WoT approach with the well-known FIWARE-based interoperability solution.Second, based on the previous analysis, we design and implement a connector to bridge the WoT architecture to the FIWARE ecosystem. Third, we conduct a performance analysis emulating a real IoT-based environment to understand scalability, response time, and computer resource usage of the two interoperability solutions. The results reveal that conceptual design choices impact the applications' performance: the WoT architecture effectively enables interoperability across IoT Platforms, though it incorporates several characteristics that hinder the implementation of applications. On the other hand, the FIWARE IoT Agent solution is platform-specific. Hence new implementations are needed for each different IoT data model.**

I. Introduction

The Internet of Things (IoT) has been creating a whole new demand over system architectures, infrastructure, and platform deployment approaches to fulfill the requirement of a new breed of smart applications [1]. Several technologies, platforms, and devices emerge in this context, creating a fragmented and heterogeneous scenario.

The lack of interoperability is one of the main factors that hinder IoT adoption on a large scale [2]. Indeed, the heterogeneity present in IoT elevates its cost since there is the need to adapt or implement new features in already established solutions to deploy it in a new scenario. A McKinsey report quantifies in 40% the additional IoT value that can be unlocked when achieving full interoperability among heterogeneous IoT systems [3].

In this myriad of services and platforms, two potential solutions to the IoT interoperability issue have attracted widespread attention in the last years. One of the most promising solutions is the Web of Things (WoT) architecture proposed by the W3C consortium, which enables interoperability across IoT platforms and application domains [4]. The other is FIWARE [5], an open-source IoT platform funded by the European Commission, comprised of a series of generic enablers that exchange data between themselves, providing different services.

However, there are still open challenges hindering the dissemination of both solutions on a broad scale. The current FIWARE interoperability solution only supports a subset of the leading IoT protocols, and adding new protocols requires programming efforts. Similarly, the WoT does not interface with many IoT Platforms, hampering the technology dissemination. Lastly, the market presence of different IoT interoperability platforms that do not integrate with each other may exacerbate the fragmentation issue rather than solve it.

In this paper, we attempt to bridge this gap through three main contributions. First, we analyze how WoT and FIWARE handle interoperability regarding qualitative aspects as flexibility, implementation difficulty, and adaptability. Second, we design and implement a connector to bridge the WoT architecture to the FIWARE ecosystem. Finally, we conduct a performance analysis study emulating a precision agriculture IoT scenario from the SWAMP Project [6] to evaluate quantitative issues that have not emerged in our qualitative analysis, such as scalability, response time, and computer resource usage.

Our results show that both solutions are capable of processing thousands of sensors with limited computational resources. Notably, our analysis sheds light on some crucial issues on the strict WoT functional requirements defined in the W3C description [4], which hamper the development and performance of the WoT architecture on a large scale. Thus, our FIWARE-WoT interface introduces benefits for both worlds. It increases the availability of protocols supported by FIWARE and enables WoT to interact with a vast catalog of FIWARE-based software modules.

In the remainder of this paper, Section II presents the background and related work. A qualitative comparison between FIWARE and WoT composes Section III. Section IV provides a detailed view of research design and methods. The key results are presented in Section V, followed by discussing the lessons learned in Section VI. Finally, Section VII concludes and proposes relevant future work.

II. Background and Related Work

Interoperability is a significant concern for IoT systems and platforms [7], with several solutions proposed at device, network, or platform layers [2]. In this paper, we focus on the latter approach by comparing two of the most popular solutions available in the market, i.e., the FIWARE ecosystem and the W3C WoT standard. Performance evaluations have

been conducted individually for FIWARE [8], and WoT [9]. Also, there are specific studies that analyse the interoperability of WoT [10]. Similarly, several applications using both approaches have been described in the IoT literature (e.g., [11], [12]). However, we are not aware of existing studies that compare them in terms of interoperability support. In the remainder of this Section, we briefly present the main characteristics of the two approaches.

A. Interoperability in FIWARE

The FIWARE platform [5] is an open-source IoT framework fostered and funded by the European Commission under Horizon 2020 program. It comprises a series of software modules called Generic Enablers (GE) that perform functions needed in various IoT-based applications.

Applications in the FIWARE ecosystem adopt a standard NGSI (Next Generation Service Interface) data exchange model that enables communication between them. IoT Agents are components that handle IoT data heterogeneity in FIWARE, translating IoT-specific protocols into the NGSI context information protocol [13]. Additionally, IoT Agents map NGSI information as virtual representations of the IoT devices in JSON entities, stored and managed by Orion, a publish/subscribe context broker.Applications can consume and publish IoT data through Orion using NGSI REST-based web interfaces. Hence, interoperability is granted once applications are in the FIWARE ecosystem and use the NGSI data model.

IoT Agent applications are data model-specific, so each different data structure requires a new IoT Agent. There are currently agents for the following data models and protocols: LWM2M over CoAP, JSON or UltraLight over HTTP/MQTT, OPC-UA, Sigfox LoRaWAN [13]. Further, a NodeJS library to enable IoT Agent development is available to build custom agents to connect non-support data-structures/network protocols to the FIWARE ecosystem.

B. Interoperability in W3C WoT

The recent WoT architecture proposed by W3C enables interoperability across different IoT Platforms and application domains.The core of this proposal is the definition of a Web Thing (WT), which indicates any "*physical or a virtual entity whose metadata and interfaces are described by a Thing Description (TD)*" [4]. The latter denotes a sequence of standardized, machine-understandable metadata encoded in JSON-LD [1] that models the capabilities of an IoT device, such as:

Affordances: provide an abstract model of the WT interface in terms of properties (i.e., the state variables of the WT), actions (i.e., commands that can be invoked on the WT), and events (i.e., notifications sent by the WT).

Protocol Bindings: map the Affordances to the network strategies (e.g., the protocols) to communicate with the WT.

Security Configurations: define the control access mechanisms to the Affordances.

[1] json-ld.org

Data Schemas: describes the information model and payload structure passed between WTs and consumers during interactions.

The TD is the primary mechanism to enable interoperability across different IoT platforms and devices. Finally, a runtime software named *Servient* implements the software object described by the TD. The Servient allows to host and *expose* a WT (i.e., to make the TD available over a network) and to interact with a remote WT by *consuming* the TD. According to W3C, [4], a WT functionality should be available in all available protocols. Thus, Servients bind multiple protocols and data models to enable interactions with different platforms.

III. WoT vs. FIWARE: A qualitative comparison

Both FIWARE and WoT handle the heterogeneity in IoT environments through a common philosophy: map IoT devices as virtual entities with well-defined data structures that other applications can consume to interact with IoT devices. In FIWARE, the context broker manages context information, while in the W3C WoT, the Servient manages the virtual entities as TDs. FIWARE defines a standard data exchange interface (NGSI) in its ecosystem and utilizes Agents to map IoT devices to NGSI entities. In the W3C WoT, each device must be associated with a WT to integrate into the WoT context. Figure 1 illustrates the interoperability approaches of WoT (A) and FIWARE (B). The architectural differences may also introduce qualitative differences on system deployments, reflected by the following aspects:

Implementation efforts: the FIWARE interoperability solution is an out-off-shelf application. Thus, programming efforts are only required if there is the need to implement a new IoT Agent for a previously unsupported data model or protocol. Opposite to that, there is not an out-of-the-shelf WoT application. However, there exist frameworks (e.g., the `node-wot` [14] tool) for assisting the development and the run-time execution of WTs.

Interfacing other applications: IoT Agents communicate only to FIWARE-based context brokers. However, FIWARE provides a vast catalog of generic and specific enablers that easily interface to any application with its ecosystem, third-party applications that do not communicate via NGSI standard require a connector to be bridged to FIWARE. A similar issue emerges in the WoT context: WT can be consumed through a Servient or by processing the WoT-specific format that is often not compatible with other IoT Platforms.

Flexibility: Both solutions require the development efforts for dealing with unknown protocols. However, the W3C WoT architecture descriptions provides clear guidelines that assist the implementation of new solutions.

Adaptability - i.e., adapt to new situations minimizing the need of a new deployment: IoT devices can be created, read, updated, and deleted to IoT Agent at run-time using its API. There is no specification in the WoT architecture of a similar interface. Although it is possible to implement such a feature in a Servient, it requires programming efforts.

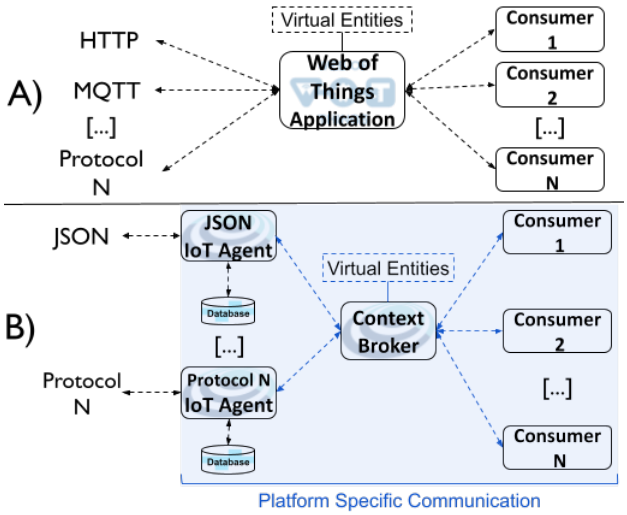

Fig. 1. WoT and FIWARE architectural definitions

Based on the aforementioned comparison, we can conclude that both solutions could gain from integrating one with the other. Although FIWARE already has an interoperability solution, it might extend the supported protocols and data formats, especially towards IoT devices that the IoT Agents do not already support. For WoT, integrating with FIWARE would vastly expand the support towards applications/platforms working with the NGSI model.

IV. WoT-FIWARE Software Adapter

We developed a generic application that bridges the WoT and the FIWARE ecosystems by translating the WoT data to the NGSI format. We opted to develop a standalone WoT mash-up application instead of implementing a direct connection from a WoT Servient to FIWARE due to two main reasons: 1) Generality: developing the Adapter as an application enables the communication between FIWARE and WoT for any system, not only for our scenario. 2) WoT best practices: the WoT Architecture [4] does not have a way of actively sending data in a specific network protocol to another software module. Instead, the mash-up application subscribes to a TD event, which transfers the data via a WoT-specific interface when triggered.

We developed the WoT-FIWARE Adapter in JavaScript, using NodeJS on top of the node-wot framework [14]. Our Adapter subscribes to the `ngsiOutput` event in the TD and maps the WT as an Orion entity. The WT triggers this event whenever there is an update in one of its properties. The Adapter iterates through such properties and encapsulates them in a JSON object represented in NGSI. We virtualized the WoT-FIWARE Adapter as a Docker container, available as an open-source project[2].

Figure 2 illustrates the WoT connection to the FIWARE ecosystem dataflow. The complete steps depicted in Figure 2 are: 1) an IoT device sends a message using a WoT supported

[2]https://github.com/UniBO-PRISMLab/WoT-FIWARE-adapter

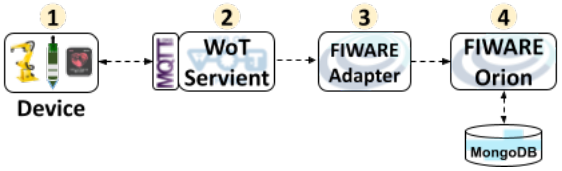

Fig. 2. Web of Things connection to the FIWARE ecosystem dataflow

protocol (e.g., MQTT) to the WoT Servient and then to the WT associated with the device; 2) the WT processes the IoT message and updates the TD properties related to that device. In turn, it triggers the `ngsiOutput` event, which notifies the WoT-FIWARE Adapter; 3) The Adapter maps the received WT to a corresponding FIWARE Orion entity. If this entity does not exist in Orion, the WoT-FIWARE Adapter creates it; 4) Orion receives the message and stores the entity information in MongoDB.

V. Quantitative Analysis: Experimental Design

In the following, we present a performance evaluation of the two interoperability solutions on a real-world IoT deployment. The goal of the evaluation is twofold: (*i*) to validate the operations of the WoT-FIWARE Adapter detailed in Section IV; (*ii*) to investigate further the performance trade-offs, scalability, and requirements of IoT Agent and WoT solutions. The quantitative comparison is influenced by the current implementations of the interoperability solutions.

A. Evaluation Scenario

We consider a performance analysis scenario based on a real IoT environment using the SWAMP Platform [6] for smart irrigation. In detail, sensor probes obtain soil data and transmit it to the SWAMP FIWARE-based Platform through LoRaWAN, where a set of mathematical and data-driven models are processed to generate an irrigation prescription map [15]. Figure 4 illustrates the complete dataflow from an infrastructural point-of-view, with real pictures of a SWAMP Pilot located in a Brazilian agriculture frontier [8].

Although the SWAMP reference sensor probe communication technology is LoRaWAN, some off-the-shelf soil sensors use the basic LoRa modulation. Those probes transmit data to a simple LoRa gateway that sends the sensor payload directly to the platform in a raw MQTT structure, thus bypassing ChirpStack - LoRaWAN server responsible for handling networking, authorization, and authentication issues. Soil probes sense the soil and transmit data to the LoRa Gateway every 10 minutes, structuring the payload according to the UltraLight2.0 protocol - a lightweight text-based protocol for constrained devices and communications where bandwidth and device memory may be limited [16].

Figure 3 depicts the core SWAMP Platform dataflow, including the two LoRa-based transmission methods. In the *Interoperability Solution* block, we tested two alternatives: 1) WoT software layer: composed of a Servient and the WoT-FIWARE Adapter that enables the communication with the

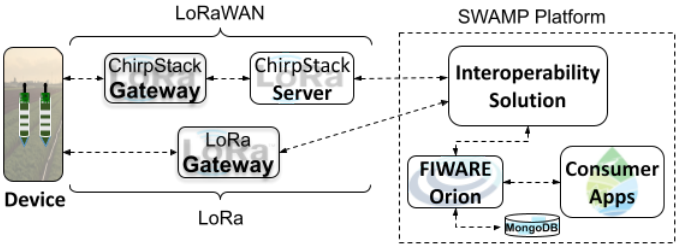

Fig. 3. Dataflow of the SWAMP IoT-based Platform

FIWARE Orion. To this aim, we developed WTs for each SWAMP soil probes; the WTs are fed by the UL data and from the ChirpStack Server; 2) Native FIWARE environment: We used the UL IoT Agent in the experiments with basic LoRa modulation and the SWAMP LoRaWAN IoT Agent [8] for the LoRaWAN experiments. Currently, there is an official version of the LoRaWAN IoT Agent, but it is not fully integrated into the ChirpStack Server.

B. Testing Environment Design and Configuration

We used SenSE (Sensor Simulation Environment) as the synthetic IoT sensor workload generator that can abstract real devices and model complex scenarios [17].As the performance analysis focuses primarily on the interoperability solution, we emulated all the data flow modules prior to the interoperability application. Thus, we relied on SenSE to emulate the data sent by those modules.

In the experiments, SenSE produces synthetic data and sends it to the interoperability solution. The corresponding soil probe entity is updated in Orion, which sends a notification with the data to the Consumer. The software components used in the experiments were deployed as Docker containers. The Consumer represents a generic application that consumes sensor data - e.g., data visualization tool, mobile app, or a third-party application. It is implemented as a simple data sink.

The number of sensors and the sensor message periodicity - each emulated sensor sends a message each 10 min - does not vary during the experiment. We utilized two Virtual Machines (VM) to perform the experiments. In VM #1, we deployed the modules that enable the performance analysis - SenSE and Consumer - and in VM #2, we deployed the applications under test: the interoperability solution, FIWARE Orion, and MongoDB. We configured both VMs as the standard Amazon AWS t2.medium instance configuration (2vCPU - 4GB of RAM).

C. Performance Analysis

We conducted 18 experiments, varying the levels of three factors - workload, protocol, and interoperability solution - as depicted by Table I. We evaluated WoT and FIWARE IoT Agent as interoperability solutions and Ultralight2.0 (UL) and LoRaWAN as protocols - mixed protocol as appears in Table I refers to experiments where both UL and LoRaWAN protocols were used simultaneously. Also, the workload was varied from low (1,000 sensors), medium (5,000 sensors), and high (10,000 sensors). SenSE emulates SWAMP sensor probes generating one packet every 10 minutes. Each experiment was replicated 30 times, and asymptotic confidence intervals were computed at the level of 99%.

TABLE I
EXPERIMENT FACTORS AND LEVELS

Factor	Level
Interoperability Solution	Web of Things - FIWARE IoT Agent
Protocol	Ultralight2.0 - LoRaWAN - Mixed
Number of Sensors	1000 - 5000 - 10000

We focused on the following metrics in the analysis: end-to-end delay (i.e. the average time taken since a sensor data point is generated until the Consumer application receives it), percentage of delivered messages, and system metrics (i.e. CPU and RAM usage per Docker container of the evaluated modules, collected every five seconds.).

VI. RESULTS

Figure 5 summarizes the key results of the performance evaluation, depicting the total experiment delay for the IoT Agent and the WoT software layer in low, medium, and high workloads for the three different types of traffic: LoRaWAN messages, UL messages, and mixed traffic - half of the sensors sending UL messages and the other half sending LoRaWAN messages. The y-axis is expressed on a logarithmic scale.

When comparing the IoT Agent and the WoT software layer's performance, it is important to stress that we utilized two different IoT Agent implementations. Thus, when observing Figure 5, we can conclude that the WoT (plus Connector) delay is similar to the official FIWARE IoT Agent. Nevertheless, it is worst than the SWAMP LoRaWAN implementation. In the experiments with mixed traffic, the WoT interoperability solution improved its performance - regarding delay - compared to its performance in experiments using LoRaWAN traffic. However, the IoT Agent performed better because both applications divided the processing between them, acting as a workload balancing.

Analyzing the delay for high workload, we conclude that neither application can keep up with 10,000 sensors, considering the computer resources allocated, since the experiments overall had delays from 18s to 51s. All messages were delivered in the IoT Agent experiments, and most of them were delivered in the WoT experiment. An issue unraveled by the experiments was the loss of messages in the WoT software layer experiments. This loss is reported in Table II, which shows the delivered message rate for all the experiments and reveals packet loss events under high workloads.

The computer resources usage are shown in Figure 6 - CPU usage - and Figure 7 - memory usage. We can observe that MongoDB is the application with the highest demand for CPU, caused by Orion. Each Orion entity is stored in MongoDB, and if an attribute is updated, Orion will also update that attribute in MongoDB, thus increasing the processing demand for the database. Regarding memory usage, there is a significant

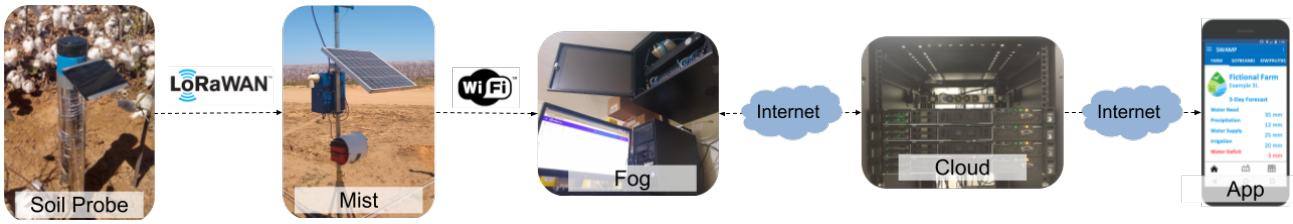

Fig. 4. SWAMP dataflow from infrastructure point-of-view

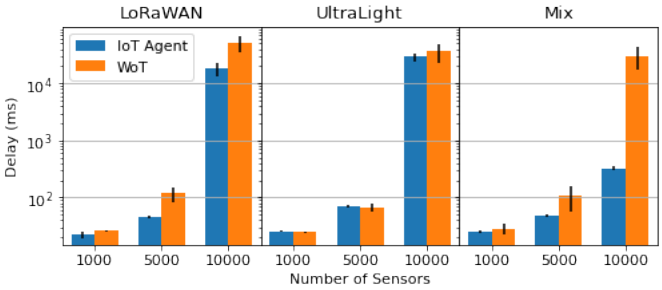

Fig. 5. Experimental delay

TABLE II
DELIVERED MESSAGES

Interoperability App.	Traffic	Workload	Delivered Messages
Web of Things	LoRaWAN	High	84.04 ± 6.10%
Web of Things	UltraLight	High	90.13 ± 3.57%
Web of Things	Mix	Medium	98.32 ± 1.14%
Web of Things	Mix	High	89.81 ± 2.80%

difference between the WoT solution and the IoT Agents, which we believe is caused by the soil probe TD. A TD is verbose and individual to each different soil probe. Also, the `node-wot` implementation does not use - by default - a database. Thus all information needed is stored in the RAM. The depletion of RAM in the WoT experiments is the cause of losing messages.

VII. DISCUSSION

The higher overall delay of the WoT software layer is expected since the latter is composed of two modules (WT + the Adapter); this solution may introduce more processing and networking steps that are not needed in the IoT Agent implementation. However, this is not true when comparing the WoT software layer with the official IoT Agent - i.e., in experiments solely with UL traffic - in this case, the delay is similar.

A significant implementation difference between the IoT Agent and the WoT solution is that Servient does not use a database. That difference is reflected in the RAM usage, which was higher for WoT since the Servient allocated the TDs in memory. Although we could have used a database in our WoT Servient implementation, this is not the default. Hence its implementation would have increased the project complexity.

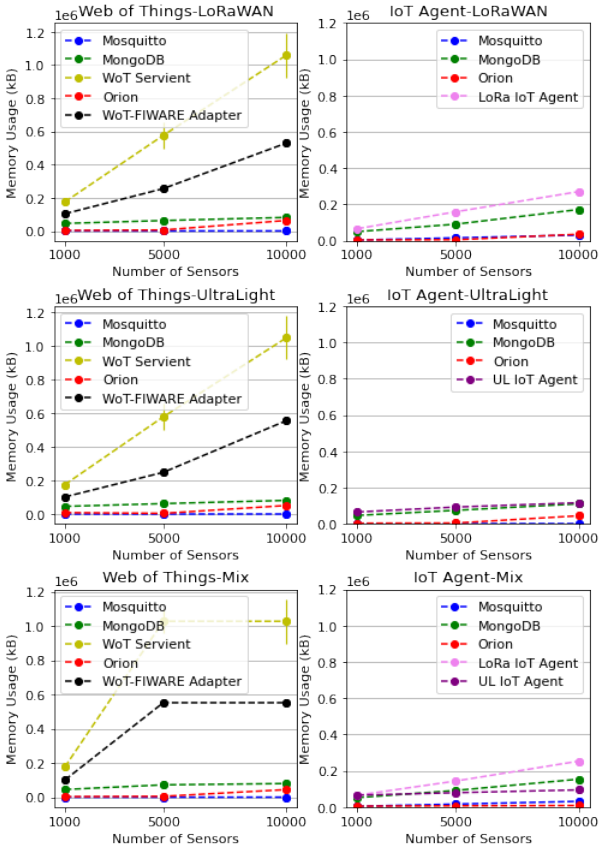

Fig. 6. Experimental CPU Usage

Moreover, the WoT requirement of not having protocol/platform-specific endpoints prevents integration with other IoT Platforms that have well-defined interfaces and protocols. A direct consequence of this decision is implementing a specific adapter for bridging the generic WoT interface to specific ones, such as the WoT-FIWARE Adapter. Although the evaluation analysis demonstrated the effectiveness and validated the correct operations of our software, the need for an adapter may introduce a possible system failure point or a bottleneck. Similar concerns about the usage of the adapter can be raised from a software architectural perspective. Indeed, the main goal of the W3C WoT initiative is to enable interoperability across IoT Platforms. Nonetheless, in our scenario, interoperability is not guaranteed by WoT but by the WoT-FIWARE Adapter.

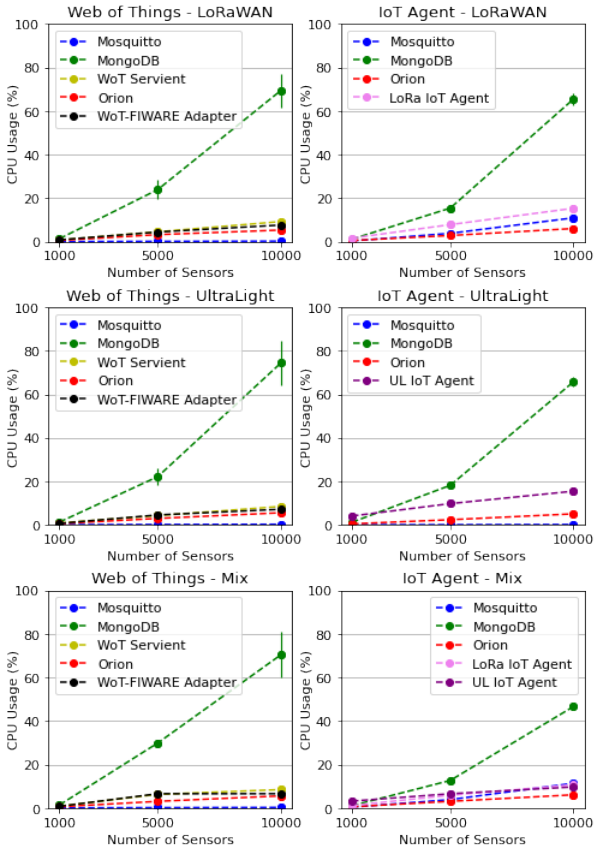

Fig. 7. Experimental Memory Usage

The same behavior occurred in [18] in which the authors developed a middleware application to integrate the WoT within the ArrowHead ecosystem. Thus, the conceptual requirements of WoT impacted its performance and required additional programming efforts compared to the IoT Agent solution: WoT enables interoperability only if all applications that communicate with it adopt the TD standard interfaces.

VIII. Conclusions and Future Works

The lack of interoperability in the IoT environment is currently a barrier to its large-scale adoption. This paper presents and compares interoperability solutions between two emerging IoT solutions - FIWARE and W3C WoT. Further, we propose, implement, and evaluated a connector that bridges both technologies. Additionally, we evaluated WoT and FIWARE interoperability applications in a real-world IoT deployment related to smart farming. Our results show that both solutions can handle large IoT systems with limited available computer resources. Our evaluation enlightens crucial trade-offs. FIWARE lacks adaptability and flexibility but is an off-the-shelf solution. WoT is flexible and customizable, though it requires coding applications from the ground up. We are planning to extend the study in twofold research directions. Further, we intend to evaluate the WoT-FIWARE connector's effectiveness on other IoT scenarios, such as Structural Health Monitoring (SHM).

Acknowledgments

This work has been funded by INAIL within the BRIC/2018, ID=11 framework, project MAC4PRO ("Smart maintenance of industrial plants and civil structures via innovative monitoring technologies and prognostic approaches"). We also acknowledge the SWAMP Project has been jointly funded by MCTIC/RNP in Brazil and the European Commission in Europe, under the EUB-02-2017 IoT Pilots call.

References

[1] L. Atzori, A. Iera, and G. Morabito, "Understanding the internet of things: definition, potentials, and societal role of a fast evolving paradigm," *Ad Hoc Networks*, vol. 56, pp. 122–140, 2017.

[2] M. Noura, M. Atiquzzaman, and M. Gaedke, "Interoperability in internet of things: Taxonomies and open challenges," *Mobile Networks and Applications*, vol. 24, no. 3, pp. 796–809, 2019.

[3] J. Manyika, M. Chui, P. Bisson, J. Woetzel, R. Dobbs, J. Bughin, and D. Aharon, "The internet of things: Mapping the value beyond the hype," 2015.

[4] 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/.

[5] FIWARE Foundation, "Fiware: The open source platform for our smart digital future." www.fiware.org, Accessed Mar. 10, 2021.

[6] C. Kamienski, J.-P. Soininen, M. Taumberger, R. Dantas, A. Toscano, T. Salmon Cinotti, R. Filev Maia, and A. Torre Neto, "Smart water management platform: Iot-based precision irrigation for agriculture," *Sensors*, vol. 19, no. 2, 2019.

[7] A. Bröring, S. Schmid, C. Schindhelm, A. Khelil, S. Käbisch, D. Kramer, D. Le Phuoc, J. Mitic, D. Anicic, and E. Teniente, "Enabling iot ecosystems through platform interoperability," *IEEE Software*, vol. 34, no. 1, pp. 54–61, 2017.

[8] I. Zyrianoff, A. Heideker, D. Silva, J. Kleinschmidt, J.-P. Soininen, T. Salmon Cinotti, and C. Kamienski, "Architecting and deploying iot smart applications: A performance–oriented approach," *Sensors*, vol. 20, no. 1, p. 84, 2020.

[9] C. Aguzzi, L. Gigli, L. Sciullo, A. Trotta, and M. Di Felice, "From cloud to edge: Seamless software migration at the era of the web of things," *IEEE Access*, vol. 8, pp. 228118–228135, 2020.

[10] C.-M. Chituc, "Towards seamless communication in the web of things: Are standards sufficient to ensure interoperability?," in *2020 13th International Conference on Communications (COMM)*, pp. 427–431, IEEE, 2020.

[11] B. Klotz, S. K. Datta, D. Wilms, R. Troncy, and C. Bonnet, "A car as a semantic web thing: Motivation and demonstration," in *2018 Global Internet of Things Summit (GIoTS)*, pp. 1–6, 2018.

[12] L. Carnevale, A. Celesti, M. Di Pietro, and A. Galletta, "How to conceive future mobility services in smart cities according to the fiware frontiercities experience," *IEEE Cloud Computing*, vol. 5, no. 5, pp. 25–36, 2018.

[13] FIWARE Foundation, "Iot agents." https://fiware-academy.readthedocs.io/en/latest/iot-agents/idas/index.html, Accessed Mar. 9, 2021.

[14] W3C Working Group, "Eclipse thingweb node-wot." https://github.com/eclipse/thingweb.node-wot, Accessed Mar. 9, 2021.

[15] R. Togneri, C. Kamienski, R. Dantas, R. Prati, A. Toscano, J.-P. Soininen, and T. S. Conic, "Advancing iot-based smart irrigation," *IEEE Internet of Things Magazine*, vol. 2, no. 4, pp. 20–25, 2019.

[16] FIWARE Foundation, "Fiware iot agent ultralight." https://fiware-iotagent-ul.readthedocs.io/en/latest/usermanual, Accessed Mar. 9, 2021.

[17] I. Zyrianoff, F. Borelli, G. Biondi, A. Heideker, and C. Kamienski, "Scalability of real-time iot-based applications for smart cities," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 00688–00693, 2018.

[18] L. Sciullo, F. Montori, A. Trotta, M. Di Felice, and T. Salmon Cinotti, "Discovering web things as services within the arrowhead framework," in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, vol. 1, pp. 571–576, 2020.