MIINT: Middleware for IIoT Platforms Integration

1st Riccardo Venanzi
DISI - dept. of Computer Science and Engineering
University of Bologna
Bologna, Italy
riccardo.venanzi@unibo.it

2nd Alberto Cavalucci
DISI - dept. of Computer Science and Engineering
University of Bologna
Bologna, Italy
alberto.cavalucci2@unibo.it

3rd Luca Foschini
DISI - dept. of Computer Science and Engineering
University of Bologna
Bologna, Italy
luca.foschini@unibo.it

4th Paolo Bellavista
DISI - dept. of Computer Science and Engineering
University of Bologna
Bologna, Italy
paolo.bellavista@unibo.it

***Abstract*—In the last recent years Internet of Things has extended its adoption to the industrial manufacturing field. The digitalization of industry brings the new concept of Industry 4.0 to light. In fact, the enormous value for the companies generated from the adoption of Industrial IoT and Edge Computing has led to a vertiginous increments of IIoT platform demands. Therefore, world-wide big IT players and Foundations start to introduce their own flagship IIoT platforms into market. This multitude of platforms presents similar common features, but with different APIs. These platforms are hardly interoperable, and they are frequently bounded to their own vertical solution stack. To overcome these limitations, in this paper, we propose MIINT, a Middleware for IIoT platforms INTegration. MIINT groups the common functionalities of IIoT frameworks, and it integrates different platforms by providing standard access APIs. To prove the feasibility of MIINT, in this paper, we show an integration use case of Azure IoT and EdgeX Foundry IIoT platforms. Moreover, we also thoroughly assess MIINT by executing on the edge a field data reading functionality in the two considered IIoT platforms, by showing their advantages and limitations.**

***Index Terms*—IIoT Middleware, IIoT Platforms, Industry 4.0, Edge Computing, Azure IoT, EdgeX Foundry**

I. Introduction

Nowadays, the world is more connected than ever and Internet of Things (IoT) has become a solid paradigm that we commonly use in everyday life. Moreover, IoT is a cornerstone for the digitalization of the industry and manufacturing fields, thanks to the advent of Industrial IoT (IIoT) and the new concept of Industry 4.0 [1], [2]. This two new concepts open the doors to new research topics and new IT solutions for the manufacturing field [3].

Cloud computing is another crucial enabler of the new scenario and various proposals tackled in the last years the challenging issue of integrating IoT device with remote clouds. Indeed, initial efforts, typically called IoT cloud solutions, were based on a two-layered architecture in which the industrial devices directly communicate with the Cloud [4]. These solutions had the goal to exploit the large amount of resource provided by Cloud to process data directly coming from industrial appliances, and to analyze those data with Big Data or Machine Learning applications in order to create value for the company [5]. This approach resulted to be not so efficient and some limitations emerged. The industrial appliances generate a huge amount of data, and to transfer all those data to the Cloud, requires very big bandwidth. In addition, the latency introduced by remote Cloud does not meet the time requirements of timely-constrained decision-making processes. Furthermore, the companies were not confident to send all their sensitive data to remote Cloud. To overcome these limitations, more recent solutions proposed Edge computing as a middleware layer between IIoT devices and Cloud [6]. The interposition of Edge nodes enables the collection, the exchange, and the analysis of data on premises. That, in its turn, facilitates monitoring practices, near real time decision making process, and predictive maintenance [7]. In addition, Edge may be leveraged to enhance data security, and relegation, as sensitive data are kept within the company, and only few safe data are uploaded to the Cloud. All these features bring a significantly increment of efficiency along with relevant economic benefits and costs reductions. In this second solutions' branch our work is placed.

More recently, the explosive diffusion of Edge-based solutions for Industry 4.0 has led to the birth of many IIoT platforms on the market. IIoT platforms provide services and functionalities to interact with industrial nodes, manage them, upload data to the Cloud, and develop and deploy applications on the edge node directly. The world-wide big IT players and Foundations approach this novel field of Industry 4.0 by introducing into market their own flagship IIoT platforms. This multiplicity of platforms, if on one side provides a large range of choice, on the other hand, introduces a enormous redundancy. All these IoT platforms provide very overlapping functionalities, indeed, for example, all the IIoT platforms enable to deploy a new edge node, provide mostly the same field device connectors, offer very similar Cloud data export services, etc.. All these very similar functionalities are provided with very different APIs. The IIoT providers offer their platform tightly coupled with their develop environment, and

Cloud Services, so a company is often forced to adopt the whole solution stack, and a vendor lock-in frequently occurs. In addition, IIoT platforms do not integrate to each other very easily, and a integration with third-party services is not always possible.

To address the above limitations, we propose MIINT, a Middleware for IIoT platforms INTegration. MIINT aims to overcome the above limitations by grouping the common functionalities provided by IIoT frameworks, and by providing standard APIs to interact with different platforms. More in detail, MIINT groups all the common features typically required by every IIoT environment and it provides a software layer that exposes standard APIs to interact with every IIoT platform and to manage the whole fleet of devices of a company, even connected with different vendors' solutions. In this way, the proposed middleware faces the IIoT platforms' heterogeneity, and enables different vendor's solution integration. To prove that, in the presented work, we integrate two very different and largely adopted IIoT platforms with MIINT. Among the wide plethora of IIoT platforms, in this research we choose a proprietary platform, Azure IoT, and a open source platform, EdgeX Foundry [8], [9]. In addition, we develop and test one of main common feature an IIoT framework has to provide, field data reading functionality. This is the pivotal feature for every monitoring application. More specifically, we develop that feature with both the above platforms and we show and report the test performance evaluation along with other emerged results. Finally, we share the developed code with the community to give the opportunity to our colleagues to reproduce and validate presented results. An open-source version of MIINT, our proposed middleware is available for the community at the link: https://gitlab.com/riccardo.venanzi/middleware-for-iiot-platforms-integration.

II. Background and Related Works

In this section, we provide some background about the reference IoT platforms used in this research and we also address similar and related works in literature.

A. Background

The rapid growth of the Internet of Things in recent years has given the birth to countless examples of smart devices, cloud services and applications. As their complexity increases, so does the need for IoT platforms. Among the wide plethora of IoT platforms, in this research we refer to two specific ones, a proprietary platform, Azure IoT, and a open source one, EdgeX Foundry [8], [9]. We have chosen these two specific platforms as a integration study case to prove the value of our research work. These two platforms have a particular relevance because they are very widely used on the market and represent a proprietary and open source solution. Let us introduce the two selected platforms in order to provide a general overview.

1) Azure IoT: It is the Microsoft's IoT platform. It provides cloud services for the connection, monitoring and control of companies' IoT assets. The general architecture is complex and offers a large number of available features. In this overview we provide a brief description about the most relevant and interesting components. Azure IoT Hub is the main service of any IoT solution developed within the Microsoft environment. It is a cloud-hosted, managed connector that serves as a central message hub for two-way communication between applications and associated devices. Another pivotal entity in the Azure IoT is Azure IoT Edge. Azure IoT Edge is a service that allows to deploy cloud workloads, Azure services, third-party services, or custom business logic component, on the edge nodes. Azure IoT Edge is the component that runs directly at the OT level and it is composed by four main entities, Software Modules, Edge Runtime, Edge Agent, and Edge Hub. Modules are the smallest computational unit managed by the framework and they contain the business logic. Edge Runtime is the execution environment installed on the edge device. The Edge Agent is responsible to create Modules' instances, to monitor them, to guarantee their continuous execution, and to notify their status to IoT Hub. Finally, Edge Hub manages messages between modules, as a message bus, and also the messages exchanged from device-to-cloud (D2C) and cloud-to-device (C2D).

2) EdgeX Foundry: EdgeX Foundry is an open source, vendor-neutral and completely operating system agnostic platform composed by microservices. This framework is designed to work at the edge level, it has dedicated microservices to directly communicate with the industrial field appliances. The core of EdgeX Foundry is composed by four microservices, namely Core Services. Those services are Core Data, Core Command, Core Metadata, and Configuration & Registry. Core Data provides storage support until the upper services (application services) consume the data. Core Command exposes to application services the actions that can be performed on a connected device. Core Metadata stores information about connected devices like: device type, data type, provided actions, etc.. Finally, Configuration & Registry has the pivotal role of providing the endpoints to the other services in order to enable their direct communication. Device Services are responsible to connect field devices with Core Services. The Device Services transform the read data into framework compliant data format. They also perform all the commands received from Command Service. Finally, the Application Services are deployed on top of Core Services, they are custom or third-party services and contain the business logic of applications.

B. Related Works

In this paper, we present a IoT middleware that aims to overcome the IoT platforms integration problems by grouping the common core platforms' functionalities and providing a common standard APIs. In addition, the middleware provides a unique entry point to manage all the companies' assets independently by the IoT platform, or platforms, currently adopted by the company. This research was born by the necessity of standardization due to vast number of IoT platforms on the market. There are several works in literature that address IoT platforms, but as best of our knowledge, our proposal is quite novel. For example, in [12], Al-Jaroodi et al. present another service-oriented IoT middleware, Man4Ware, but it is focused on integration of different IoT services. Another similar solution is presented by Ngu et al. in their survey [13]. They lay the focus on different devices integration, enabling technology and challenges. A middleware solution for Industry 4.0 scenario is presented by Glock et al. in [14]. It is a custom IoT middleware that act as a IoT platform and it is meant to be deployed at the same level of other vendors' IoT solutions. In [15], Agarwal et al. provide a study of some IoT middlewares on the market with the goal to address the IoT application developer. The middleware presented by Steinmetz and his colleagues provide an interesting solution to integrate different devices based on a ontology [16]. Cilia is a service-oriented middleware for smart manufacturing proposed by Lalanda et al. [17]. It faces data flexibility in industry field, and it aims to manage data workflow from the field to the Cloud, by reducing human intervention. All of those works are very interesting and well done, but they address different limitations from ours.

III. MIINT Middleware: Architecture, Components, and APIs

In this section, we describe our proposed middleware MIINT along with its architecture and internal components. The middleware groups the common functionalities of each integrated IoT platform making its usage totally transparent for the user. To do that, MIINT provides software APIs that abstract the common functionalities of the specific IoT platform by hiding the single implementation details. The APIs exposed by MIINT allow the user to deploy common services without having any interaction with the specific IoT framework.

A. Logical Architecture

MIINT is based on a microservices architecture. It is meant to be placed on top of the integrated IoT platforms and it exploits microservices architecture modularity to integrates the various underlying IIoT platforms. More in detail, MIINT requires a microservice for each IIoT platform that it is meant to integrate. In this way each platform is managed independently. Furthermore, MIINT provides the opportunity to deploy just the required functionalities of each single platform, without the constraint to integrate the whole framework. In this work we only focused on the integration of Azure IoT and EdgeX Foundry, but to integrate other platforms is easy by adding a service that takes care of specific IoT platform functionalities. The Fig.1 depicts the logical architecture of the MIINT. The two services, AzureIoT Service, and EdgeX Service manage the edge nodes by hiding their implementation details. Middleware Service is a software module that acts as a bridge between the different IoT platforms services and the user. Middleware Service is responsible of exposing the common APIs to interact with the IoT platforms services. More in details, each IoT platform is controlled by its service, while Middleware Service groups and exposes their common functionalities. Finally, MIINT integrates the Eureka Server. This server provides a Registry and a Discovery Service.

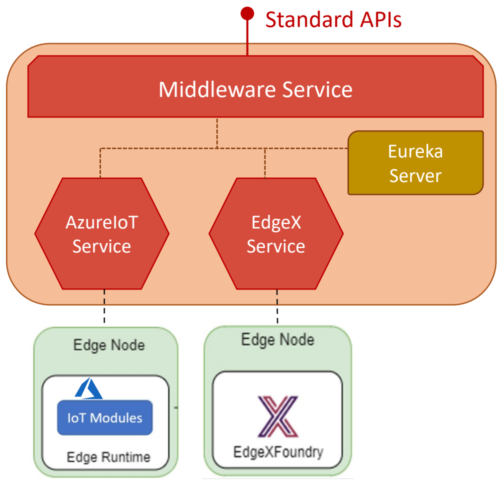

Fig. 1: MIINT Middleware Logical Architecture

B. Middleware Service

Middleware Service is the component of the middleware that provides IoT platform-independent standard APIs. This service allows the companies to manage their production processes trough a unified interface that groups the most common features of IoT platforms. Specifically, the Middleware Service exploits the methods/features that IoT platform related Service exposes to interact with the specific framework in a totally agnostic manner. So far, we have developed the following features:

- **Update Configuration:** POST request to define the required parameters to establish a connection with the edge node. The body of this request change according with the target platform. This method allows to edit a existing configuration;
- **Get Configuration:** GET request to obtain the configuration currently in use;
- **Create Edge Device:** POST request to create and define resources on a edge node. The body content of the request changes according with the target platform.
- **Get Edge Device:** GET request to obtain the current state of the resources on the edge node. This request requires the resource name as a parameter;
- **Delete Edge Device:** DELETE request to eliminate the resource on the edge node. This request requires the resource name as a parameter;
- **Deploy Storage Service:** POST request to deploy the storage functionality by saving the data of target monitored machinery. This API requires a body in JSON format whose content changes depending by the target provider.

1) Eureka Server: Eureka Server is the Service Discovery of the middleware. It is responsible of decoupling services from their respective physical addresses. The discovery mechanism is updated every time a service instance starts or stops execution. At start-up time all services register to Eureka Server. When a client invokes an API on Middleware Service to perform an action on a edge node, Middleware Service queries Eureka Server for the specific IIoT platform service instance that manages that edge node. Eureka provides service's endpoint to Middleware Service, which then can communicate directly with the service. Each service must periodically send a heartbeat to prevent the cancellation of its registration which is kept in non-persistent memory. Eureka server also provides a load balancer that works with round robin algorithm.

C. AzureIoT Service

AzureIoT Service has the role to translate the standard APIs provided by Middleware Service in specific Azure requests. AzureIoT Service interacts with Azure IoT Hub via REST API, and it offers all the required features to manage the Azure Edge Node life cycle. In this way, final users are no longer forced to interface with the Azure cloud portal to manage their IoT systems. The AzureIoT Service provides and implements the following functionalities:

- **Create/Delete** a edge device entity. These operations include the installation/elimination of the Azure Edge Runtime;
- **Deployment of a module**, even custom, on the edge device;
- **Deployment of a functionality** as set of modules that work together for providing a service.

D. EdgeX Service

EdgeX Service is the microservice responsible of translating the API exposed by Middleware Service in requests for EdgeX Foundry. EdgeX Service handles node devices through: Device Profiles and Device Services. In EdgeX Foundry each connected device is associated with a profile. Device Profile describes the various resources available on the device and the operations that it can perform. EdgeX Service implements and includes all the required features for correctly managing the edge node:

- Retrieval of all Device Profiles available (through Core Metadata);
- Querying for a specific Device Profile by its identification name;
- Upload a custom Device Profile on the platform;
- Deleting a Device Profile from the platform by its identification name.

The edge device directly communicate with its relative Device Service that acts as device abstraction for the framework. This class of services is responsible for connecting devices to the platform, transforming the values coming from the field into a data structures that can be used by Core Services. EdgeX Service implements all the required APIs to control this functionalities:

- Fetching all devices available within the framework, or of a specific one by name;
- Registration/Elimination of a device;
- Upload a custom Device Profile on the platform;
- Deleting a Device Profile from the platform by its identification name;

A substantial difference is emerged between the two reference frameworks, Azure IoT forces the user to pass through IoT Hub to manage the edge node, while EdgeX Foundry has the direct control over the edge nodes.

IV. Experimental Results

In this section, we present the experiments performed on the two platform services we have integrated with MIINT. To test those services, we have developed and deployed a key functionality to any IIoT platform: field data reading functionality. This functionality is the core of every Industry 4.0 monitoring scenario, and it responsible of reading data from the field appliance. This functionality can be developed to read data with many protocols. We chose Modbus, the standard de-facto protocol for industrial Programmable Logic Controllers (PLCs).

This section is divided in two subsections. The first one addresses the environment in which the two IoT platforms have been evaluated and the tools used for the analysis. The latter presents the tests performed on two different deployment configurations and the emerged results.

A. Tools and Testing Environments

The goal of our tests is to study and analyze the behaviour of the platforms in a typical Industry 4.0 data reading scenario. The two IIoT platforms read data from the field appliance by exploiting the above mentioned field data reading functionality. In order to have a the most comparable and truthful results, we prepared the testing environments to be as similar as possible. To do that, we decided to connect the EdgeX Foundry platform, typically deployed as a standalone software at the edge, to the public Azure cloud by developing an additional component that exports the data read to Azure IoT Hub. Fig.2 depicts the environment. It is composed by three core elements: an HMI simulator, an Edge node and an IoT Hub. To simulate a realistic machine-to-machine traffic we used an HMI simulator with 190 registers. HMI simulator uses Modus TCP as communication protocol. The HMI simulator and the Edge node run on two different VMs with the same hardware and software configuration. They have a quad-core processor, 2 GBs of RAM, and Ubuntu 18.04 LTS as OS. The hardware resources are intentionally constrained to simulate a traditional edge node installed on a real plant. The edge nodes run different modules according with the relative platform. Azure node has 3 components: a Modbus connector to read data from the field device, an ETLmodule that pre-process data, and an Azure Blob Storage to save data. Similarly, in EdgeX we have a Device Service to read data from Modbus simulator, the Core Data that stores data, and the Azure Export Service that exports data to IoT Hub. Both edge nodes send

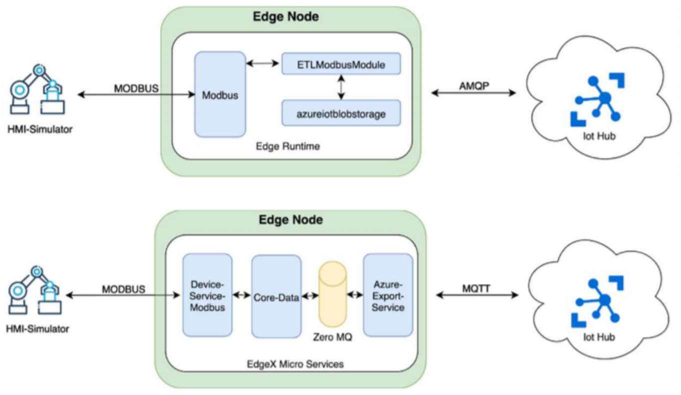

Fig. 2: Deployment environments

the data to an Azure IoT Hub, but they differ on the protocol used: the Azure node uses AMQP, while EdgeX node MQTT. We have used Wireshark to analyze and evaluate the packets traffic among the components. Furthermore, we used cAdvisor, a container performance monitor developed by Google, to keep track of the CPU and MEM usage [18]. We also used Prometheus and Grafana to store data in time series and to provide data visualization.

B. *Tests and Experimental Results*

This subsection reports how the two IIoT platforms perform on running the field data reading functionality with two different reading frequencies, 4Hz, and 8Hz (it means a reading period of 250ms and 125ms respectively). For these two configurations, we report a chart for the data exchanged between edge nodes and IoT Hub, and two qualitative charts for edge node CPU and MEM usage. More in detail, the CPU and MEM charts depict the maximum CPU/MEM utilization of the specific edge node while it is executing reading operations. This specification is needed because each edge node internally runs all the components as microservices. The charts show the total of all these components' consumption. The detailed charts of the broken-down components' CPU and MEM utilization are available at https://gitlab.com/riccardo.venanzi/middleware-for-iiot-platforms-integration.

The first test targets to read 30 registers from Modbus device with a frequency of 4Hz (blue bars of Fig.3, and Fig.4). In this case, both the platforms showed a normal and expected increment of CPU and MEM usage, referred to their resources' utilization in idle state, due to the communication. The differences of behaviour are due to the different internal composition of modules, but no substantial or unexpected result has emerged from these charts in this test case. What it has emerged, was the big difference of size of the data transmitted to the IoT hub, Fig.5. As we can see from the charts in Fig.5, Azure transferred roughly 26 MB of data overall against 380 KB of the EdgeX counterpart. Considering the fact that the format obtained from the Modbus registers is almost the same, we have deeper investigated the Modbus communication modules. It came out that the configuration files of the two modules differ for a parameter that enables multiple registers readings, it is present only in Azure IoT. Since there is no equivalent configuration parameter on the EdgeX Modbus Device Service, each register has to be singularly read. It might be a crucial limitation in a scenario in which the scope is to read multiple registers at once. In the second test, yellow bars of Fig.3, and Fig.4, we increased the reading frequency to 8Hz for a total of 8 values per second for each register monitored. An accordingly increase of resource usage was expected, in fact the MEM usage of EdgeX node had a remarkable increment, while the CPU usage roughly tripled its value. While, its amount of data exported almost doubled in value, exchanging roughly 600KB, Fig.6. On the other hand, Azure IoT platform showed a very different and unexpected results. In this case, the data transferred drastically dropped from 25 MB of the previous test, to 1.1 MB (Fig.5, and Fig.6). This result has led to deeper investigation on values and data stored locally on the edge node. It turned out that only 5 values were stored, showing that Azure Modbus connector was not capable of keeping up with reading at frequency lower than 4Hz. From the tests performed, it has emerged that there is not an absolute right choice about the IoT framework to use, but it is strictly related to the context. More precisely, in case there is the need of multiple registers reading, Azure IoT platform results to be better than EdgeX Foundry. On the other hand, EdgeX Foundry is a better choice than Azure IoT in case of high frequency readings scenario.

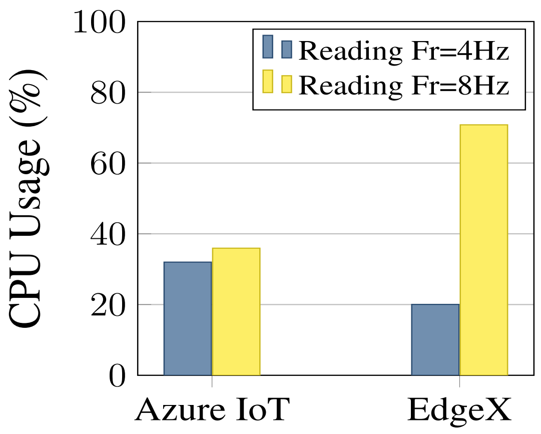

Fig. 3: CPU usage

V. Conclusions and Ongoing Work

The paper presented MIINT, a service oriented middleware for IIoT platforms integration. MIINT aims to overcome the large heterogeneity among the various IIoT platforms in the market. MIINT enables integration by grouping the most common functionalities of the IIoT platforms and then it provides standard APIs to interact with different vendors' solutions. To prove the effectiveness of our solution and to assess it in real settings, we have integrated two widely used IIoT frameworks, Azure IoT and EdgeX Foundry. In addition, in this research

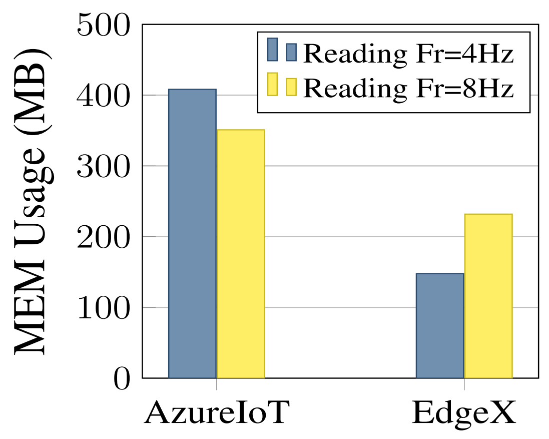

Fig. 4: MEM usage

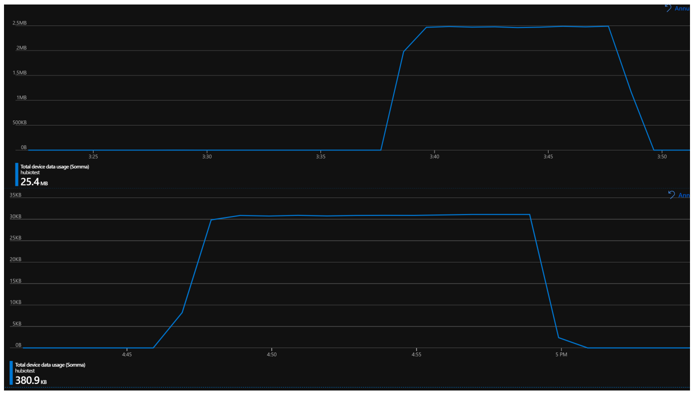
Total device data usage (Somma)
hubiotest
25.4 MB
Total device data usage (Somma)
hubiotest
380.9 KB

Fig. 5: Azure IoT and EdgeX Data exchanged (Fr=4Hz)

we developed and deployed a field data reading functionality. We tested the performance of the two platforms by exploiting this functionality to read from a MODBUS simulator. We also shared the code to allow our colleagues to reproduce and validate our research. Finally, in our ongoing effort we are extending the common standardized features exposed by MIINT middleware and integrating more IIoT platforms, such as Siemens MindSphere.

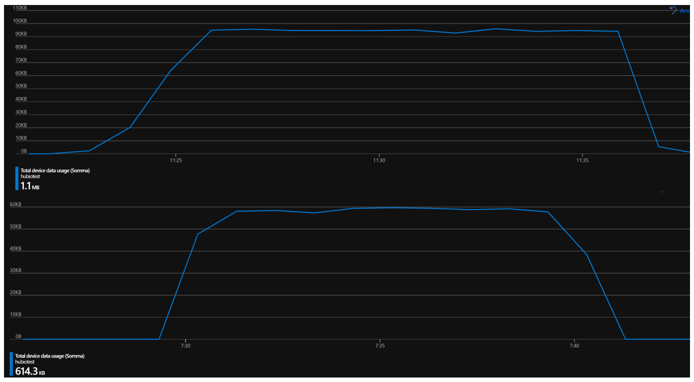
Total device data usage (Somma)
hubiotest
1.1 MB
Total device data usage (Somma)
hubiotest
614.3 KB

Fig. 6: Azure IoT and EdgeX Data exchanged (Fr=8Hz)

Acknowledgment

The authors would sincerely thank our graduated M.Sc. student, Andrea Sturiale, for the amazing work he has done for his dissertation, and on which this research is based. We wish him a brilliant career.

References

[1] Okano, Marcelo T. "IOT and industry 4.0: the industrial new revolution." International Conference on Management and Information Systems September. Vol. 25. 2017.

[2] Kagermann, Henning, Wolf-Dieter Lukas, and Wolfgang Wahlster. "Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution." VDI nachrichten 13.1 (2011): 2-3.

[3] Hugh Boyes, Bil Hallaq, Joe Cunningham, Tim Watson, The industrial internet of things (IIoT): An analysis framework, Computers in Industry, Volume 101, 2018, Pages 1-12, ISSN 0166-3615, https://doi.org/10.1016/j.compind.2018.04.015.

[4] Kim, Jin Ho. "A review of cyber-physical system research relevant to the emerging IT trends: industry 4.0, IoT, big data, and cloud computing." Journal of industrial integration and management 2.03 (2017): 1750011.

[5] Shohin Aheleroff, Xun Xu, Yuqian Lu, Mauricio Aristizabal, Juan Pablo Velásquez, Benjamin Joa, Yesid Valencia, IoT-enabled smart appliances under industry 4.0: A case study, Advanced Engineering Informatics, Volume 43, 2020, 101043, ISSN 1474-0346, https://doi.org/10.1016/j.aei.2020.101043.

[6] Sittón-Candanedo, Inés, et al. "Edge computing architectures in industry 4.0: A general survey and comparison." International Workshop on Soft Computing Models in Industrial and Environmental Applications. Springer, Cham, 2019.

[7] S. Trinks and C. Felden, "Edge Computing architecture to support Real Time Analytic applications : A State-of-the-art within the application area of Smart Factory and Industry 4.0," 2018 IEEE International Conference on Big Data (Big Data), 2018, pp. 2930-2939, doi: 10.1109/BigData.2018.8622649.

[8] https://azure.microsoft.com/en-us/overview/iot/

[9] https://www.edgexfoundry.org/

[10] IoT Cloud Platform Market by Offering (Platform and Service), Deployment Mode (Public Cloud, Private Cloud, and Hybrid), Organization Size, Application Area (Building & Home Automation and Connected Healthcare), and Region - Global Forecast to 2025, MarketsAndMarkets, 2020.

[11] Shanong Liu, Number of publicly known Internet of Things (IoT) platforms worldwide from 2015 to 2019, Statista, 2021.

[12] Jameela Al-Jaroodi, Nader Mohamed, and Imad Jawhar. 2018. A service-oriented middleware framework for manufacturing industry 4.0. SIGBED Rev. 15, 5 (October 2018), 29–36. DOI:https://doi.org/10.1145/3292384.3292389

[13] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal and Q. Z. Sheng, "IoT Middleware: A Survey on Issues and Enabling Technologies," in IEEE Internet of Things Journal, vol. 4, no. 1, pp. 1-20, Feb. 2017, doi: 10.1109/JIOT.2016.2615180.

[14] T. Glock et al., "Service-Based Industry 4.0 Middleware for Partly Automated Collaborative Work of Cranes," 2019 8th International Conference on Industrial Technology and Management (ICITM), 2019, pp. 229-235, doi: 10.1109/ICITM.2019.8710661.

[15] Agarwal, Preeti, and Mansaf Alam. "Investigating IoT middleware platforms for smart application development." Smart Cities—Opportunities and Challenges. Springer, Singapore, 2020. 231-244.

[16] Charles Steinmetz, Achim Rettberg, Fabíola Gonçalves C. Ribeiro, Greyce Schroeder, Michel S. Soares, Carlos E. Pereira, Using Ontology and Standard Middleware for integrating IoT based in the Industry 4.0, IFAC-PapersOnLine, Volume 51, Issue 10, 2018, Pages 169-174, ISSN 2405-8963, https://doi.org/10.1016/j.ifacol.2018.06.256.

[17] P. Lalanda, D. Morand and S. Chollet, "Autonomic Mediation Middleware for Smart Manufacturing," in IEEE Internet Computing, vol. 21, no. 1, pp. 32-39, Jan.-Feb. 2017, doi: 10.1109/MIC.2017.18.

[18] cAdvisor (Container Advisor) - https://github.com/google/cadvisor.