Modelling IoT Application Requirements for Benchmarking IoT Middleware Platforms

Shalmoly Mondal
Swinburne University of Technology
Melbourne, Australia
shalmolymondal@swin.edu.au

Alireza Hassani
Deakin University
Melbourne, Australia
ali.hassani@deakin.edu.au

Prem Prakash Jayaraman
Swinburne University of Technology
Melbourne, Australia
pjayaraman@swin.edu.au

Pari Delir Haghighi
Monash University
Melbourne, Australia
pari.delir.haghighi@monash.edu

Dimitrios Georgakopoulos
Swinburne University of Technology
Melbourne, Australia
dgeorgakopoulos@swin.edu.au

ABSTRACT

The significant advances in the Internet of Things (IoT) have led to IoT applications being widely used in various scenarios ranging from smart city, smart farming, to Industrial IoT (IIoT) solutions. With the explosion of IoT application development, IoT middleware platforms are increasingly being used for hosting such IoT applications. This has given rise to the need for developing benchmarking solutions to analyze and test the performance of different middleware platforms that host these IoT applications. To develop such benchmarks, there are a number of key components that are needed. One of these components is an IoT dataset. To generate such datasets, representing IoT application requirements in a general and formal way is important. In this paper, we propose a framework to model the IoT Applications Requirements and enable Data Generation(ARDG-IoT). The framework supports a formal way to capture IoT application requirements and use these requirements to generate IoT data that can be used to create benchmarks for different IoT middleware platforms. ARDG-IoT consists of our proposed model, IoTSySML, which captures the application requirements, and an IoT data simulator tool, which is used to generate IoT data. We present an evaluation of the framework using a real world Industrial IoT application case study.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; • **Software Engineering → Object Oriented Language**; UML.

KEYWORDS

IoT Middleware, Benchmarking, IoT Application Requirements, Requirements Engineering, Requirements Modelling

ACM Reference Format:
Shalmoly Mondal, Alireza Hassani, Prem Prakash Jayaraman, Pari Delir Haghighi, and Dimitrios Georgakopoulos. 2021. Modelling IoT Application Requirements for Benchmarking IoT Middleware Platforms. In *The 23rd International Conference on Information Integration and Web Intelligence (iiWAS2021), November 29-December 1, 2021, Linz, Austria.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3487664.3487742

1 INTRODUCTION

IoT middleware platforms are increasingly being used to host IoT applications to ease the application development process. The current research trend shows that the market for IoT middleware is expected to reach a value of USD 22.36 billion by 2025 [24]. IoT middleware platforms such as Microsoft Azure, FIWARE [3], Cumulocity[5], to name a few, are now hosting a greater number of IoT applications providing the capability to manage IoT devices, store, and process the data generated from these devices. The performance of IoT applications depends on the middleware platform and ability to understand its capabilities in hosting the IoT application. Benchmarking has been traditionally employed to establish a standard way to assess the performance of systems that can be used to help make suitable choices. While benchmarking in areas like databases, big data, and stream processing are well established, the differences in characteristics of IoT applications from those of conventional database and similar systems makes developing benchmarking solutions for IoT systems difficult. As an example, factors like the heterogeneity of IoT devices, complex network structures and protocols, the need to support various data analytics requirements (from simple statistics to complex machine learning), etc makes it a challenging task to develop solutions that provides a mechanism to conduct benchmarks of IoT middleware platforms.

1.1 Motivating Scenario

Consider the following motivation scenario of a manufacturing or Industrial IoT (IIoT) solution [13] which aims to improve the productivity of the factory workers in a meat processing plant using data captured from wearable IoT devices. The performance of each individual worker is evaluated using activity recognition. The factory workers are equipped with wearable watch-like IoT devices called MetaWear that record accelerometer and gyroscope data. The motion data is analyzed using machine learning algorithms hosted in the cloud based IoT platform and classifies the activities of each worker.

This scenario has several challenges. The first challenge reflects the complexity of IoT ecosystem in terms of the heterogeneity of the different components it encompasses (devices, network, communication protocols). Additionally, such IoT applications have requirements which are different from those of conventional database systems. Furthermore, data generated by IoT devices is characterized by high velocity and dynamicity, which makes it different from conventional datasets (which is more static). Hence, getting access to such datasets or generating similar data to support benchmarking and analyzing the performance of different IoT middleware is a challenge and impedes IoT application development and deployment.

Being able to generate IoT data pertinent to the application requirements is a key to be able to conduct performance benchmark of IoT middleware platform. A first step towards this is to be able to capture and represent the IoT application requirements which then provides a standard way to generate IoT data for the given application. This enables repeatable data to be generated while conducting performance benchmarks of IoT middleware. There is limited work in the literature that focuses on capturing the diverse IoT application requirements and use this to generate IoT data relevant to the IoT application. To address this gap, we propose IoTSySML, a system modelling language-based approach to elicit, capture and represent the requirements of IoT applications. We have also developed ARDG-IoT, a framework that can be used to generate data considering the captured application requirements captured by IoTSySML. The goal is to describe the application requirements and generate IoT data which can later be used to create benchmarks for different IoT middleware platforms. We evaluate the proposed methodology using our motivating scenario.

Our main contributions in this paper are:
(i). Identifying requirements of IoT applications and proposing a model-based approach to represent these requirements. This study will be significant for other related research in better understanding of common needs and requirements of IoT applications.
(ii). A framework which captures the application requirements to generate data that can be used for benchmarking across different middleware platforms
(iii). Experimental implementation and evaluations to validate the proposed framework and show that IoT data is being generated according to the application requirements. We implement our proposed framework using an Industrial IoT solution.

With our proposed framework, we are providing an easy-to-use approach where IoT application developers who want to host their IoT application on middleware platforms, can use the IoTSySML model to specify their requirements for generating their datasets, and run it across multiple platforms for conducting benchmarks. The rest of the paper is organized as follows: section 2 introduces the architecture and design of the ARDG-IoT framework, section 3 illustrates its implementation. Section 4 presents the experimental evaluation of the framework and reports the results. Section 5 discusses the related work, and section 6 concludes the paper with a roadmap for future work.

2 ARDG-IOT FRAMEWORK

In this section we describe our proposed framework - ARDG-IoT. Figure 1 illustrates the key elements of the ARDG-IoT framework. It consists of the proposed IoTSySML model to capture and represent the IoT application requirements, an application description generator, a data generation tool, and a monitoring engine that monitors the performance of the IoT middleware platform hosting the IoT application. In the subsequent subsections, we describe each of these components.

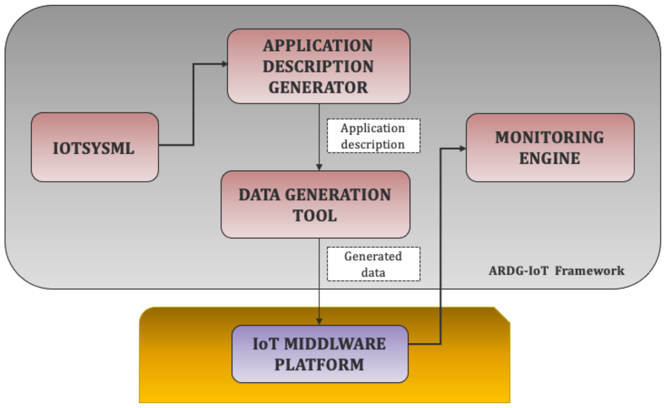


Figure 1: Architecture of the ARDG-IoT framework

2.1 IoTSySML: Modelling IoT Application Requirements

IoT applications can be applied to various domains like smart traffic [11], smart manufacturing[13], smart parking[15], smart agriculture[17], and many more. IoT applications have requirements which are versatile and different from the traditional web applications. Hence, to capture these diverse set of requirements, it is essential to first study and identify the characteristics of IoT applications in various domains. Currently, there is a lack of systematic approaches for determining and identifying the different IoT applications requirements. As a first step in that direction, we propose IoTSySML, to capture the IoT application requirements. The model has been developed using SysML[1] which is a domain specific and platform independent graphical language especially designed for the field of Model Based System Engineering (MBSE)[14]. SysML uses *Stereotypes*, *Tagged Values* and *Constraints*[1] to provide extensions to existing concepts.

We have followed the below steps to develop the model for capturing the requirements of IoT applications:
(1). An in-depth analysis of a wide range of existing IoT applications across various domains.
(2). Identifying the attributes for each of these requirements (e.g., Sense Rate, Data Interval, Sampling Frequency for IoT sensor device requirements)
(3). Identifying the relationship and dependencies between the entities (association/composition)
(4) Determining the multiplicity (one-to-many, many-many)

[1]https://www.uml-diagrams.org/profile-diagrams.html

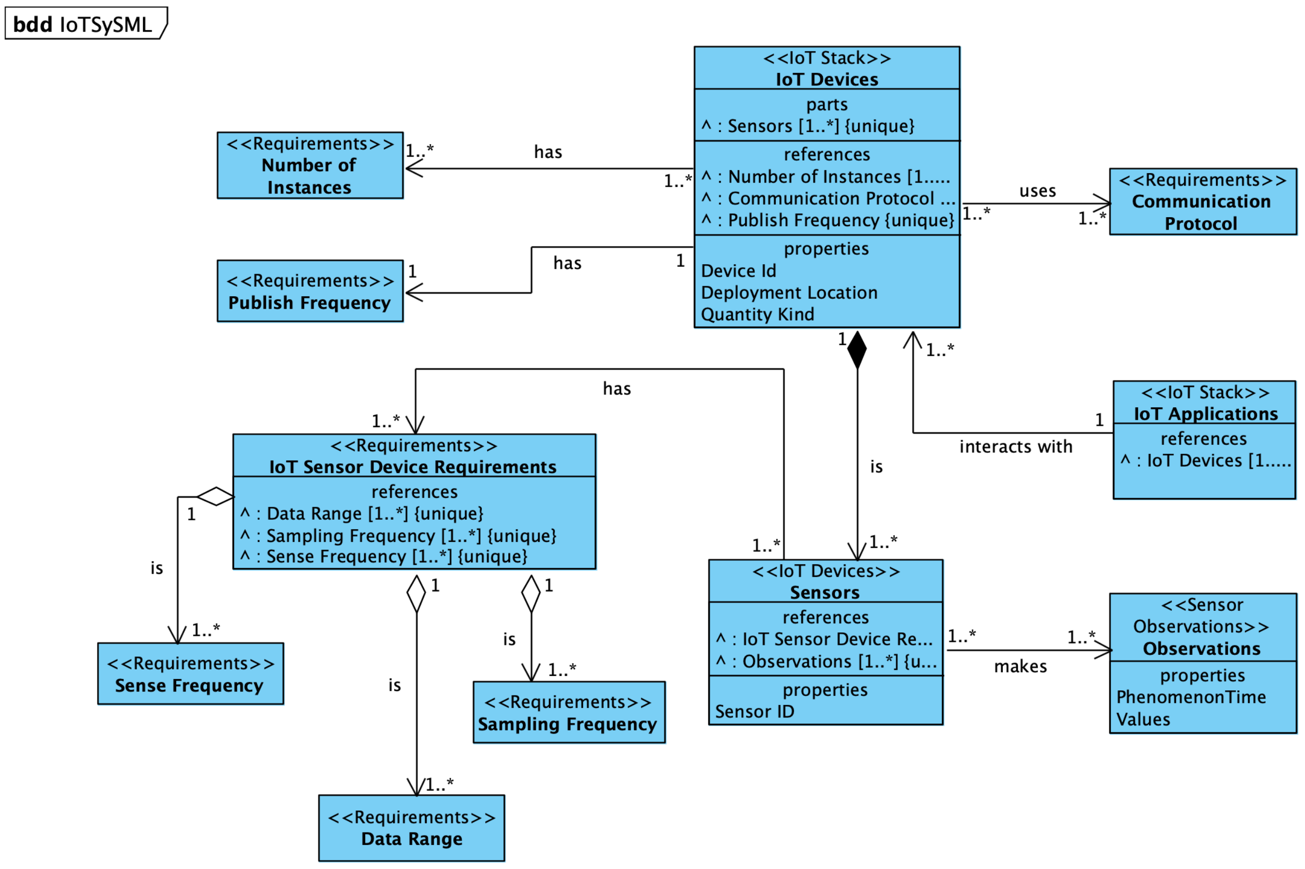


Figure 2: IoTSySML:A Model for Capturing IoT Application Requirements

We model the IoT device requirements of IoT applications from a benchmarking perspective. Our model has been developed after an extensive analysis of the literature of IoT applications across various domains[11][13][17][15] and research on the field of requirements engineering[19][9][10][18][20]. IoTSySML is also based on concepts from IoTLite[22] and SOSA Ontology[23]. IoTSySML is a lightweight extension of SySML and uses UML profiling to capture the IoT application requirements. The model has been conceived using the SySML Block Definition Diagram (BDD) illustrated in Figure 2. Each concept has been encapsulated using a 'block'. Each block is a stereotyped (customized) extension of a UML Class and has a set of properties and defined relation with other blocks. The 'IoT Application' block is stereotyped as <<*IoTStack*>> and interacts with the 'IoT Devices' block. We show the model element property like multiplicity property for the blocks. Multiplicity properties can be 0 to many, one to many, many to many, etc. For example, IoT applications share a many-to-may relationship with IoT devices shown in the figure. Similarly, IoT devices can have multiple sensors which is shown by one-to-many relationship between the 'IoT devices' and 'Sensors' block. We also show a composition relationship between 'IoT Devices' and 'Sensors' block. Composition relationships are denoted by a solid line with a solid diamond at the end of a composition. IoT devices leverage different 'Communication Protocols' to send the data to IoT middleware platforms. They also have a 'Publish Frequency' requirement which denotes the rate at which the sensed data is required to send the data to the edge devices or cloud services. Different frequencies for publishing data should be supported. Sensors may send multiple values in one second or one value in four hours.

IoT devices also have 'Number of Instances' which indicate the number of sensors for which data has to be generated. Sensors are stereotyped as << *IoTDevices* >> and can make 'Observations' having attributes 'Values' and 'Phenomenon Time' which are stereotyped as 'Sensor Observations'. The property 'Values' is the data generated by the sensors, for example, acceleration values generated by accelerometer or temperature values generated by temperature sensors. The property 'Phenomenon Time' is the exact time instant in which the data is being generated by the sensors. 'IoT Devices' block has static attributes like device id, deployment location, and quantity kind. *Device id* is the unique identification for each device,*deployment location* is the geographic locations where the devices are deployed. We are considering the location static for this research. The *quantity kind* determines the type of value that is being sensed. For example, the quantity kind is acceleration for our motivating scenario. The 'IoT Sensor Device Requirements' block stereotyped <<*Requirements*>> consists of *Sense Frequency, Data*

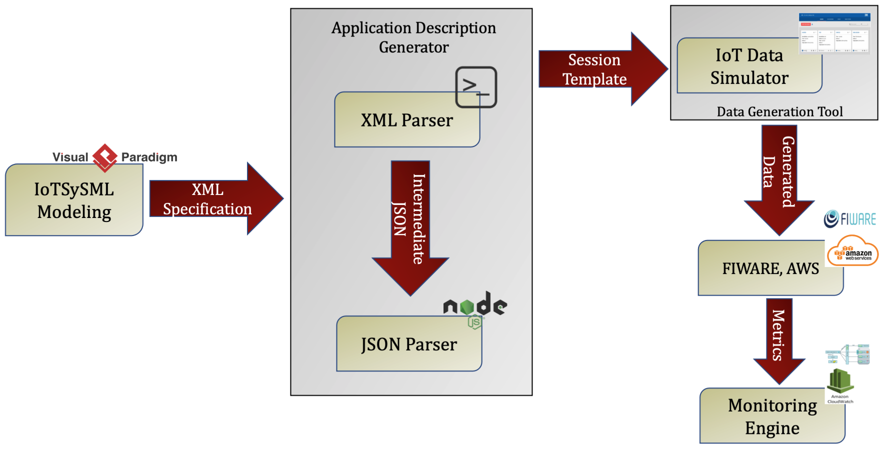


Figure 3: ARDG-IoT Framework Implementation.

Range, and *Sampling Frequency*. They have a shared aggregation relationship. Sense frequency[10] describes how often the data is being generated. For example, a temperature sensor may generate data every second whereas health vitals from patients can be generated every couple of minutes. Sampling frequency[16] refers to the number of data points required per second. Sampling Frequency can be calculated as 1/(SF)SenseFrequency. Some applications like stock market analysis might require higher sampling frequency whereas other applications might not require a higher sampling frequency. Data range defines the range in which the sensor data points would lie. For example, in the example of our motivating scenario, if the value of acceleration along the x-axis lies in the data range [-0.2, 0.6], it denotes an activity of hand movement, e.g., lifting of hand.

2.2 Application Description Generator

The application description generator component in Figure 1 is in charge of taking the IoT application requirements from IoTSySML and converting it into an input template for the data generation tool. The application requirements captured by IoTSySML needs to be parsed by this component to a template which can be provided as an input to the IoT data simulator.

2.3 Data Generation Tool

This component is responsible for generating IoT data according to the captured requirements. IoT applications might have varying requirements for data generation. Some applications might require data to be generated more frequently than others. For example, a traffic monitoring application[11] might require data to be generated every few seconds to get the real-time location and speed information of vehicles. A smart farming application[17] on the other hand, might require data from soil sensors every few minutes. For our motivating scenario, to detect high-level activities of meat processing workers like working with meat or waiting for meat, low-level sensor data (e.g., accelerometer data or gyroscope data) should be generated which can represent activities like the direction of the hand movement. The generated data can then be reasoned into activities like grabbing a piece of meat or cutting large chunks of meat into smaller pieces. To compute such activities of the workers, the analysis algorithm might require data every minute. To provide data to the application at this frequency, the data should be generated at the required rate which is taken care of by this component.

2.4 Monitoring Engine

This component is responsible for providing real-time information of the generated data being received on the cloud. It monitors the incoming number of messages, incoming number of requests, and the number of successful requests. It also keeps a record of the obsolete message requests, failed requests, etc. Some of the tools that can be used for this purpose is CLAMS[8], which may be deployed across multiple IoT middleware platforms.

3 ARDG-IOT FRAMEWORK - IMPLEMENTATION

Figure 3 shows the workflow and the technology stack used for implementing the proposed framework. We start by modelling the IoT application use-case scenario. Based on the proposed generic IoTSySML model in Figure 2, we have developed an instance of the use-case. We have used Visual Paradigm tool[6] since it supports the generation of the XML specification document, which needs to be converted into a format understandable by the IoT data simulator.

This is done by the application description generator which consists of two components: the XML parser and the JSON parser. The XML parser first receives the IoT application requirements from IoTSySML as an XML specification document and parses it to intermediate JSON using a PowerShell script, which is accepted as the input of the JSON Parser. The JSON Parser now reads the JSON file and generates a template that interacts with the IoT data simulator and contains all the details needed to create sessions to start generating the data. This has been implemented using Node.js and JavaScript. The outcome of the application description

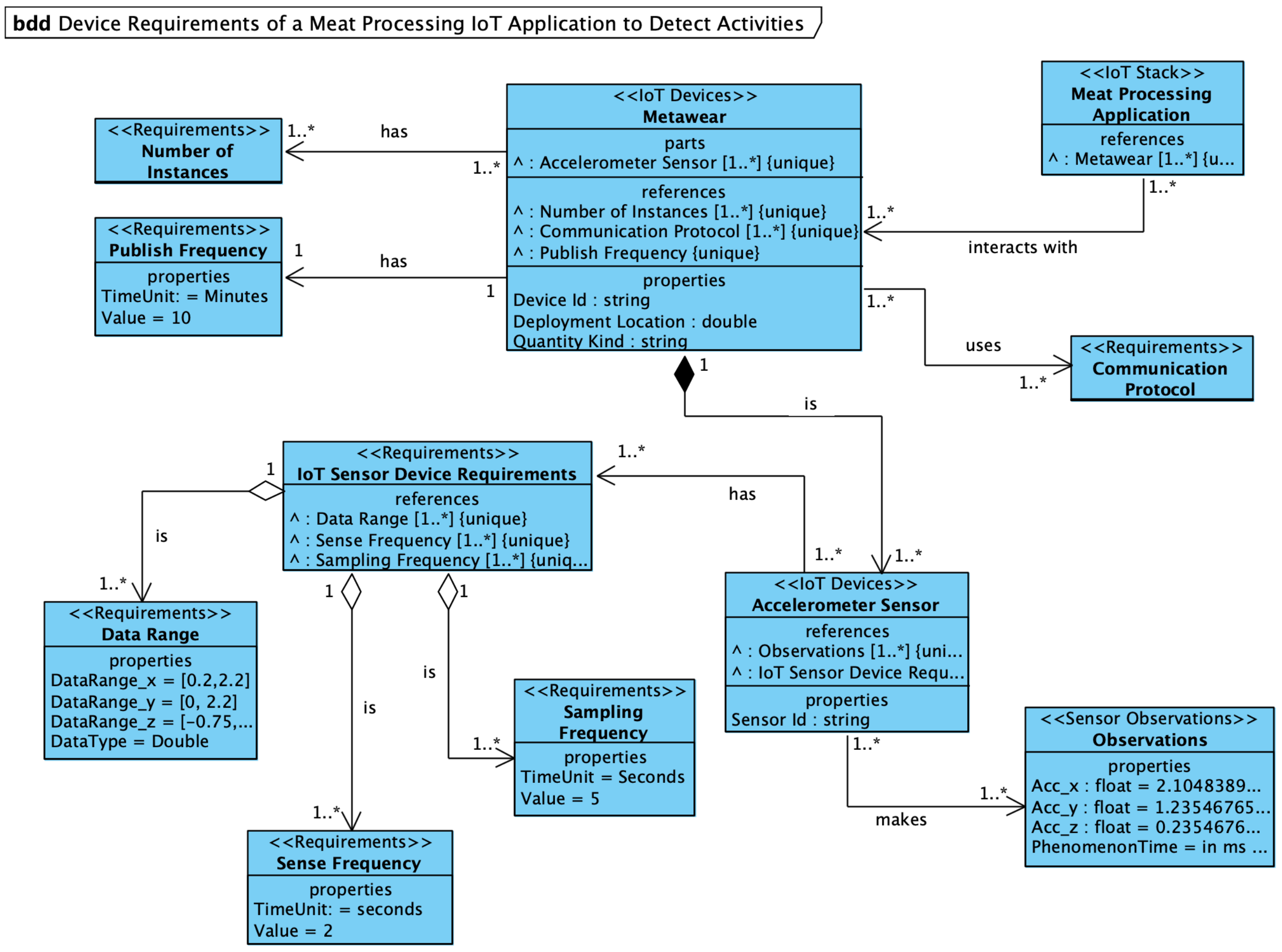

bdd Device Requirements of a Meat Processing IoT Application to Detect Activities
<<Requirements>> Number of Instances
<<Requirements>> Publish Frequency
properties
TimeUnit: = Minutes
Value = 10
<<IoT Devices>> Metawear
parts
^ : Accelerometer Sensor [1..*] {unique}
references
^ : Number of Instances [1..*] {unique}
^ : Communication Protocol [1..*] {unique}
^ : Publish Frequency {unique}
properties
Device Id : string
Deployment Location : double
Quantity Kind : string
<<IoT Stack>> Meat Processing Application
references
^ : Metawear [1..*] {u...
interacts with
uses
<<Requirements>> Communication Protocol
has
is
<<Requirements>> IoT Sensor Device Requirements
references
^ : Data Range [1..*] {unique}
^ : Sense Frequency [1..*] {unique}
^ : Sampling Frequency [1..*] {uniq...
<<IoT Devices>> Accelerometer Sensor
references
^ : Observations [1..*] {uni...
^ : IoT Sensor Device Requ...
properties
Sensor Id : string
<<Requirements>> Data Range
properties
DataRange_x = [0.2,2.2]
DataRange_y = [0, 2.2]
DataRange_z = [−0.75,...
DataType = Double
<<Requirements>> Sampling Frequency
properties
TimeUnit = Seconds
Value = 5
<<Sensor Observations>> Observations
properties
Acc_x : float = 2.1048389...
Acc_y : float = 1.23546765...
Acc_z : float = 0.2354676...
PhenomenonTime = in ms ...
makes
<<Requirements>> Sense Frequency
properties
TimeUnit: = seconds
Value = 2
1..*
1

Figure 4: Block Definition Diagram of the IoT Application Requirements of the Meat Processing Application.

generator is a session template which can be uploaded to the data generation tool to generate IoT data. The IoT-data-simulator[2] tool was used to generate data relevant for our IoT application. It is an open-source tool and provides two options to generate data: we can either create a schema to define the data structure of the data to be generated or we can opt for a schema-less data generation which can be implemented using JavaScript. There are some basic concepts that are required to use the tool. The core entity of the IoT data simulator is a session. The simulator relies on these sessions to generate the data. The second core concept is data definition which is used to define a schema to represent the data structure we are working with. We first create a session and then define schema using data definitions. We can also add multiple devices and send the data to different target systems. The communication protocols currently supported are AMQP, MQTT, REST, Web sockets and HTTPS. The IoT data simulator is written in java, hence, it was convenient to generate the session template using JavaScript and Node.js. The IoT middleware platforms that have been used to send the generated data are FIWARE[3] and Amazon Web Service (AWS). We establish a connection to FIWARE platform through REST API and to AWS through MQTT protocol.

Additionally, we have automated the session creation process of the IoT data simulator which is currently done manually. Earlier, every time we wanted to generate data with a given set of application requirements, we needed to create a session, define schemes, data definition, etc. As a part of the implementation, to speed up things and avoid the manual process, we have implemented a component which receives the application requirements from the IoTSySML model, generates a template, which can directly be uploaded to the tool to create the sessions. Furthermore, to eliminate the need to upload the generated template to create sessions, we have enabled session creation through HTTP requests. We used Node.js.Axios to send asynchronous HTTP requests to the REST endpoints. So, currently, we can send HTTP requests from the browser to create the sessions automatically.

4 EXPERIMENTAL EVALUATION

This section presents the evaluation of the proposed ARDG-IoT framework. Our evaluation consists of two parts. First is a case study based evaluation, where we show that our proposed IoTSySML model can capture the requirements for our motivation scenario. Second, we demonstrate how an IoT application developer can use the proposed framework to conduct performance analysis on different IoT middleware platforms.

4.1 Case Study Based Evaluation

To evaluate IoTSySML, we discuss the application requirements of our motivating scenario discussed in section 1.1. The application monitors the productivity of workers by detecting their activity in a meat processing plant. We have captured the application requirements using the proposed IoTSySML. Figure 4 illustrates a SysML Block Definition Diagram (BDD) of the captured application device requirements.

In our scenario, the IoT application interacts with IoT devices which are wearable watch like devices called Metawear with accelerometer and gyroscope sensors. We assume we have 10 workers wearing the metawear devices on both wrists. This information is captured by the 'Number of Instances' block in Figure 4. We provide this captured information to the IoT data simulator, which starts generating data for twenty devices. In this use-case, we have considered accelerometer sensor data for detecting the movement of hands. We specified a set of requirements to detect activities of workers in a meat processing plant. Static requirement of devices: device id, location and quantity kind which has been set to specific values. The IoT sensor device requirement, 'Sense Frequency', has been set to a value of 2 seconds, which means that the accelerometer sensor is required to generate data every 2 seconds. The 'Data Range' for the acceleration values has been specified along the x, y, and z axis. Finally, we specify the required 'Sampling Frequency'. The 'Accelerometer Sensor' makes 'Observations' which are acceleration values along the x, y, and z axes and the time instant in which the accelerator data is generated in milliseconds precision. These acceleration values that are generated for the application represent data relevant to some activity performed by the workers.

4.2 Evaluating ARDG-IoT

In this section we discuss the designed experiments to validate our proposed framework and its ability to generate relevant IoT data for conducing benchmarks based on IoT application requirements captured by the IoTSySML model. Let us assume, an IoT application developer of the above case study(section 4.1) wants to conduct a data ingestion and query performance analysis on different IoT platforms. To achieve this, we have conducted two sets of experiments on two IoT middleware platforms. We have chosen one open-source and one commercial platform for our experiment.

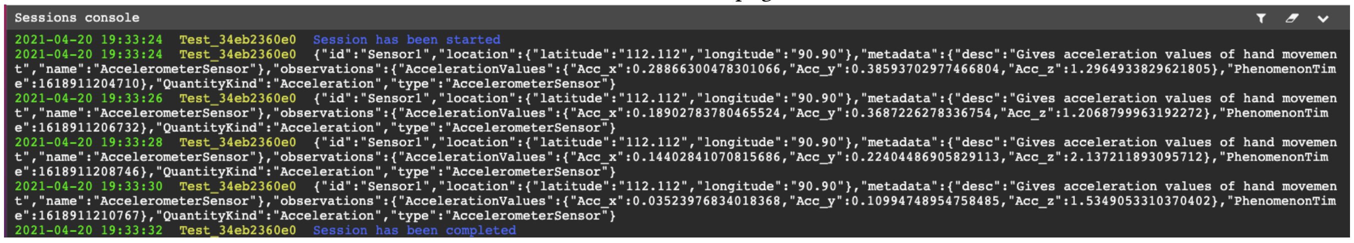
4.2.1 Experimental Set up. The experiments were conducted on two different IoT middleware platforms: FIWARE and AWS. Data was generated using the IoT data simulator. FIWARE and IoT data simulator were deployed using Docker on a system with 1.8 GHz Dual-Core Intel Core i5 processor, 16GB RAM, running MAC OS (Operating System).

The first set of experiments are run on AWS to evaluate the data ingestion performance. We leveraged the "IoT core" managed cloud service provided by AWS IoT to register our devices for data to be ingested into the middleware platform. We first register our IoT devices on AWS Cloud and establish a secure connection with the IoT data simulator over port 8883. We then run our data generation component to generate data and ingest the data into the middleware platform. The proposed IoTSySML allows users to select their preferred communication protocol for data ingestion. We used the MQTT communication protocol to connect to AWS IoT. MQTT uses a high throughput pub/sub message broker and ensures secure transmission from the IoT devices. The two basic operations associated with MQTT mechanism is publish and subscribe. The IoT devices publish their data to a particular topic. Anyone who wants to view or access the data needs to be subscribed to that particular topic. IoT core was used to subscribe to our MQTT topic "test/devices". The data in this experiment has been securely sent using X.509 digital certificates. The certificates were used to establish a secure MQTT connection with AWS IoT. Authorization and access have been granted by AWS IAM Roles and Policies.

Similarly, the experiment is run on the FIWARE platform. We have used the Generic Enabler (GE) Context Broker (CB) to connect to the IoT data simulator. It is one of the key and mandatory modules of FIWARE and implements a Publish/Subscribe paradigm. We have used HTTP to connect the IoT data simulator to FIWARE. The CB is implemented through Orion, which is the open-source reference implementation of the CB Generic Enabler and is based on the OMI NGSI standard [4]. The FIWARE version of the OMA NGSI interface is a RESTful API via HTTP. The CB receives the data generated by the IoT data simulator through the Cygnus connector. The Orion Context Broker relies on MongoDB technology to stores the generated sensor data and requests to this service is using NGSI. The CB is run by making HTTP requests to the exposed ports. After the data is generated, it is ingested into the FIWARE platform. We have repeated the experiment on both middleware platforms for different data ingestion rates and calculated the ingestion delay for each data ingestion rate.

Now that we have the generated dataset, users (an IoT application developer) might want to run queries and test the query execution performance. We have performed the second set of experiments on FIWARE and AWS to show that the proposed framework can be used to conduct this performance analysis and provide them with the results. We have considered two types of queries for this experiment: simple and complex queries. For the query performance evaluation on AWS, we used the AWS Service "IoT Rules' to trigger queries based on an event. The outcome of the query was stored on DynamoDB database. For the experiments on FIWARE, once the data was received on FIWARE, we implemented a FIWARE event listener using Node.js which evaluated the response time of a query which is triggered on detection of any event. We connected to the MongoDb server to fetch the outcome of the queries. We carried out the experiment with queries of different complexities and calculated the query response times for each of them. We have repeated the experiments ten times on both platforms and have taken the aggregation of the results. The configurations and metrics are discussed in the next section. The results are illustrated in section 4.3.

4.2.2 Performance Analysis Configuration and Metrics: We performed the experiment with different configurations to test the session console.png



```
Sessions console
2021-04-20 19:33:24  Test_34eb2360e0  Session has been started
2021-04-20 19:33:24  Test_34eb2360e0  {"id":"Sensor1","location":{"latitude":"112.112","longitude":"90.90"},"metadata":{"desc":"Gives acceleration values of hand movemen
t","name":"AccelerometerSensor"},"observations":{"AccelerationValues":{"Acc_x":0.28866300478301066,"Acc_y":0.38593702977466804,"Acc_z":1.2964933829621805},"PhenomenonTim
e":1618911204710},"QuantityKind":"Acceleration","type":"AccelerometerSensor"}
2021-04-20 19:33:26  Test_34eb2360e0  {"id":"Sensor1","location":{"latitude":"112.112","longitude":"90.90"},"metadata":{"desc":"Gives acceleration values of hand movemen
t","name":"AccelerometerSensor"},"observations":{"AccelerationValues":{"Acc_x":0.18902783780465524,"Acc_y":0.3687226278336754,"Acc_z":1.2068799963192272},"PhenomenonTim
e":1618911206732},"QuantityKind":"Acceleration","type":"AccelerometerSensor"}
2021-04-20 19:33:28  Test_34eb2360e0  {"id":"Sensor1","location":{"latitude":"112.112","longitude":"90.90"},"metadata":{"desc":"Gives acceleration values of hand movemen
t","name":"AccelerometerSensor"},"observations":{"AccelerationValues":{"Acc_x":0.14402841070815686,"Acc_y":0.2240448690582913,"Acc_z":2.137211893095712},"PhenomenonTim
e":1618911208746},"QuantityKind":"Acceleration","type":"AccelerometerSensor"}
2021-04-20 19:33:30  Test_34eb2360e0  {"id":"Sensor1","location":{"latitude":"112.112","longitude":"90.90"},"metadata":{"desc":"Gives acceleration values of hand movemen
t","name":"AccelerometerSensor"},"observations":{"AccelerationValues":{"Acc_x":0.03523976834018368,"Acc_y":0.10994748954758485,"Acc_z":1.5349053310370402},"PhenomenonTim
e":1618911210767},"QuantityKind":"Acceleration","type":"AccelerometerSensor"}
2021-04-20 19:33:32  Test_34eb2360e0  Session has been completed
```

Figure 5: Session Console of the IoT data simulator showing the generated IoT data

data ingestion and query execution performance of AWS and FIWARE.

Data ingestion rate is the rate at which sensor data being inserted to a system per second.

Query Complexity is the number of conditions being used in the queries. We define 'Simple Queries' as queries with one filter condition while 'Complex Queries' as those with multiple filter conditions'.

We have also identified the following metrics which have been used by the proposed framework for conducting performance analysis. The monitoring engine component of our framework has the capability to measure the following metrics.

Ingestion Delay: Ingestion delay is the time difference between the time when the data is being generated and the time at which it is being inserted into the platform.

Query Response Time: By query response time we mean the time taken to execute a query after it is triggered on detection of an event.

4.3 Experimental Results

In this section, we report the results of the conducted experiments to evaluate the framework. The first result illustrates the frameworks ability to generate data based on the captured application requirement.

Figure 5 illustrates the generated data using the IoT data simulator. We can see that the IoT data simulator generates a reading every 2 seconds, which validates our 'Sense Frequency' requirement captured by IoTSySML. It also shows the generated timestamp and the acceleration values. The accelerometer sensor generates data as a three valued vector along the x(lateral), y(longitudinal), and z(vertical) axes.

Figure 6 plots the acceleration values and illustrates that the acceleration values generated are according to the data range requirement specified in our proposed model.

4.3.1 *Data Ingestion.*

Figure 7 illustrates the data ingestion performance. To illustrate that ARDG-IoT can work with different configurations and IoT middleware platforms, we report the performance in varying conditions. We compare the delay in the data ingestion of the two platforms FIWARE and AWS as we vary the data ingestion rate. The graph indicates that as the data ingestion rate increases, there is a slight increase in the ingestion delay of FIWARE, whereas the increase in the ingestion delay of AWS with the increase in the ingestion rate is more noticeable.

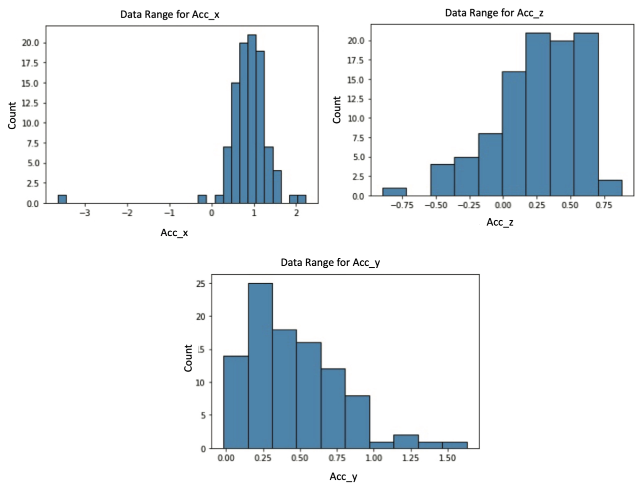


Figure 6: Plot showing the generated acceleration values along the x, y and z axes

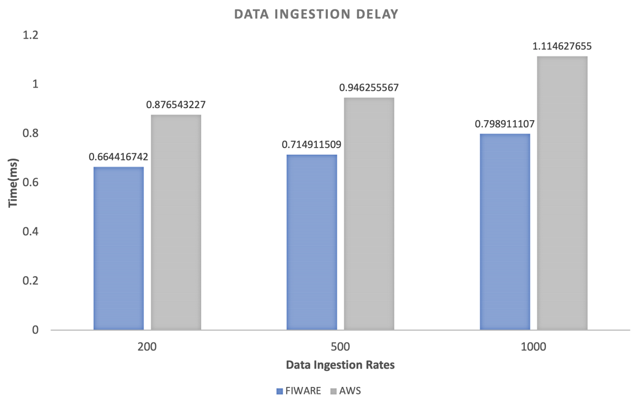


Figure 7: Data ingestion delay of AWS and FIWARE with varying data ingestion rates

4.3.2 *Query Execution Performance.*

Figure 8 shows the query response time of AWS and FIWARE as we vary the complexity of the query.

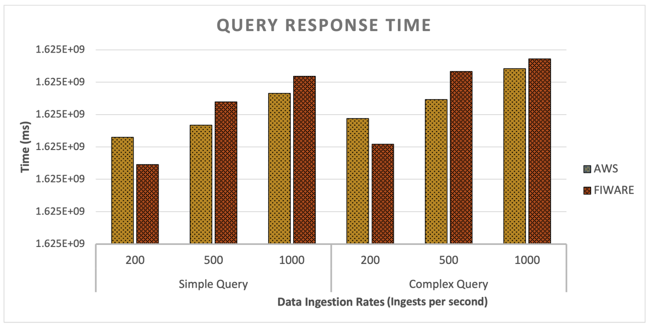


Figure 8: Query response time of AWS and FIWARE with varying data ingestion rates and complexity of queries

We have performed the experiment with three data ingestion rates. The graph shows that for a data ingestion rate of 200 data per second, the query response time of AWS is greater than FIWARE, whereas for ingestion rates 500 and 1000, the response time of FIWARE is more than AWS. Overall, the graph shows that, with the increase in the complexity of the query (as in increase the number of conditions in the query), there is a slight spike in the query response time for all the ingestion rates. We have taken the aggregated values for comparing the query response times.

4.3.3 Analysis. Using the proposed framework, we have described the IoT application requirements using a system engineering approach. The focus has been to provide a formal way of representing the IoT application requirements using our proposed model, IoT-SySML. To demonstrate that IoTSySML can capture the IoT application requirements correctly, we have used the proof-of-concept implementation of our framework and generated the data with the application requirements of a real-world IIoT application scenario as the input.

For the second part of the validation, we have run a set of experiments to illustrate the data generation and ingestion capability of our proposed framework. The evaluation results showed that ARDG-IoT can generate IoT data by taking the IoT application requirements as input description. We have ingested the generated data to two different platforms using different communication protocols. We have also illustrated that our framework can evaluate the performance of different platforms based on a set of queries. Once users have specified their application requirements, the proposed framework can be used to run a performance analysis on different platforms and provide them with the results. For example, in section 4.3, Figure 7 shows spikes in data ingestion delay for AWS platform as we increase the data ingestion rates while for FIWARE platform there is a subtle increase. Hence, the proposed framework can be used to detect this type of occurrences and would allow application developers to run such experiments easily and investigate if any issues exist.

During the experiment, we observed other factors which affected the data ingestion of a middleware platform like the underlying structure of the dataset and the communication protocol being used. We can potentially say that the data ingestion depends on the database component being used by the different middleware platforms to store the data and the communication protocol being used. The format of generated dataset varies for each middleware platform. Hence, the complexity of data ingestion of each IoT middleware platform depends on the data structure of the dataset and the data model that is supported by the platform. For example, ingesting the data in FIWARE was more complex to AWS because of the type of data model supported by FIWARE, where each entity should have an "id" and "type" property to be properly ingested into the platform. This is common to all NGSI entities, as well as defining terms such as "Property" and "Relationship" to describe the dataset.

5 RELATED WORK

Significant work has been done using UML modelling techniques by Reggio et al.[19] for IoT systems. The paper proposed IotReq, a UML based domain modelling method to specify the requirements for IoT based systems. IotReq consists of a domain model and a goal paradigm approach for requirement specification. Rohjans et al.[20] have used IntelliGrid methodology for developing requirements for smart grid architectures. The methodology describes user requirements as use case descriptions. Similarly, another study [9]introduces a requirement engineering technique based on use case modelling for requirement specification of smart spaces. The proposed technique identifies the possible actions in a smart space, where each action is represented by use-cases. Software engineering techniques that have been explored for requirements gathering other than UML include ERD and simple block diagrams. [18] modelled IoT healthcare systems using rich pictures (cartoon-like drawings based on informal rules) and use cases. Costa et al.[10] have used MBSE approach using SySML to specify the requirements of IoT systems. The paper proposes a modelling language, which is conceived as a SysML Profile. The use of constraints for IoT systems have been well depicted.

The existing works [7] on benchmarking IoT middleware platforms have focused on publish/subscribe type of middleware. Two middleware platforms have been compared based on a set of qualitative and quantitative metrics. Additionally, the middleware platforms have been considered as a black box without considering the internal implementation. Salhofer et al. [21] have evaluated the FIWARE platform from an application deployment point of view. However, no performance and load test has been conducted. The Transaction Processing Council (TPC), which has developed industry-standard benchmarks for databases and big data systems, has introduced TPCx-IoT, a benchmark targeted for IoT gateways. Cruz et al.[12] give a performance evaluation study of a few existing middleware platforms. Qualitative and Quantitative metrics have been proposed to evaluate the performance of the IoT Middleware platforms. The proposed metrics were tested with five middleware platforms, namely InatelPlat, Konker, Linksmart, Orion+STH and Sitewhere. It has been reported that depending on the metrics, the different middleware platforms performed differently. For example, in case of low throughput and packet size, Konker and Linksmart performed better, whereas when error percentage is the priority, Orion +SSH gives the best performance with less than 1% error rate. Sitewhere operated well with concurrent users.

In summary, relatively little effort has been given to identifying IoT application requirements from a benchmarking perspective.

Moreover, most of research on identifying the application requirements have emphasized on architectural requirements[20] of IoT systems. To the best of our best knowledge, there is no research at present which captures IoT requirements for generating data to support benchmarking. Furthermore, a framework that can capture IoT application requirements and use these to run various testing scenarios on IoT middleware platforms is missing.In our work, we identify the application requirements and use these to generate IoT data that can be used to compare the performance of different middleware platforms.

6 CONCLUSION AND FUTURE WORK

The ARDG-IoT framework presented in this paper is a step towards developing an integrated approach for conducting benchmarks and performance analysis of IoT middleware platforms hosting IoT applications. We have demonstrated that our framework can take different configurations of input from the user and can generate application specific IoT datasets that can be used to conduct benchmarking and performance analysis of IoT platforms. We validated the proposed framework using a real-world Industrial IoT application use case. The application requirements are captured using our proposed model, IoTSySML. We also demonstrated that the proposed framework could capture IoT application requirements and automatically translate these requirements into IoT data. Furthermore, our proposed framework provides the capability to work with different IoT middleware platforms which is exemplified by ingesting the generated data to different IoT middleware platforms. Hence, based on any application and the requirements they might have, the proposed framework can be used to select between different middleware platforms and give a view of how they would perform when we have the actual application running. Currently, we have used JSON format to send the data to the platforms and have considered fixed queries for performance analysis. For our future work, we aim to extend the proposed framework to include audio/video data stream or data sent in binary format. We also aim to modify the queries and take into consideration factors like the richness of queries.

REFERENCES

[1] 2019 (accessed March 18, 2021). *Systems Modeling Language (SysML).* https://sysml.org.

[2] 2020 (accessed April, 2021). *IoT Data Simulator.* https://github.com/IBA-Group-IT/IoT-data-simulator.

[3] (accessed July 31, 2020). *Fiware.* https://www.fiware.org.

[4] (accessed June 28, 2021). *NGSI.* https://knowage.readthedocs.io/en/6.1.1/user/NGSI/README/index.html.

[5] (accessed October 15, 2020). *Cumulocity IoT.* https://www.softwareag.cloud/site/product/cumulocity-iot.html.

[6] (accessed October 15, 2020). *Visual Paradigm.* https://www.visual-paradigm.com.

[7] Ana Aguiar and Ricardo Morla. 2019. Lessons learned and challenges on benchmarking publish-subscribe IoT platforms. In *Proceedings of the 2nd Workshop on Benchmarking Cyber-Physical Systems and Internet of Things.* 24–29.

[8] Khalid Alhamazani, Rajiv Ranjan, Karan Mitra, Prem Prakash Jayaraman, Zhiqiang Huang, Lizhe Wang, and Fethi Rabhi. 2014. Clams: Cross-layer multi-cloud application monitoring-as-a-service framework. In *2014 IEEE International Conference on Services Computing.* IEEE, 283–290.

[9] Muhammad Waqar Aziz, Adil Amjad Sheikh, and Emad A Felemban. [n.d.]. Requirement engineering technique for smart spaces. In *Proceedings of the International Conference on Internet of things and Cloud Computing.* 1–7.

[10] Bruno Costa, Paulo F Pires, and Flavia C Delicato. [n.d.]. Specifying Functional Requirements and QoS Parameters for IoT Systems. In *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech).* IEEE, 407–414.

[11] Néstor Cárdenas-Benítez, Raúl Aquino-Santos, Pedro Magaña-Espinoza, José Aguilar-Velazco, Arthur Edwards-Block, and Aldo J Sensors Medina Cass. 2016. Traffic congestion detection system through connected vehicles and big data. 16, 5 (2016), 599.

[12] Mauro AA da Cruz, Joel JPC Rodrigues, Arun Kumar Sangaiah, Jalal Al-Muhtadi, and Valery Korotaev. 2018. Performance evaluation of IoT middleware. *Journal of Network and Computer Applications* 109 (2018), 53–65.

[13] Abdur Rahim Mohammad Forkan, Federico Montori, Dimitrios Georgakopoulos, Prem Prakash Jayaraman, Ali Yavari, and Ahsan Morshed. 2019. An Industrial IoT Solution for Evaluating Workers' Performance Via Activity Recognition. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS).* IEEE, 1393–1403.

[14] Laura E Hart. 2015. Introduction to model-based system engineering (MBSE) and SysML. In *Delaware Valley INCOSE Chapter Meeting*, Vol. 30. Ramblewood Country Club Mount Laurel, New Jersey.

[15] Alireza Hassani, Pari Delir Haghighi, Prem Prakash Jayaraman, Arkady Zaslavsky, Sea Ling, and Alexey Medvedev. 2016. CDQL: A Generic Context Representation and Querying Approach for Internet of Things Applications. In *Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media.* 79–88.

[16] JC Hernandez, F Sanchez-Sutil, A Cano-Ortega, and CR Baier. 2020. Influence of Data Sampling Frequency on Household Consumption Load Profile Features: A Case Study in Spain. *Sensors* 20, 21 (2020), 6034.

[17] Andreas Kamilaris, Feng Gao, Francesc X Prenafeta-Boldu, and Muhammad Intizar Ali. [n.d.]. Agri-IoT: A semantic framework for Internet of Things-enabled smart farming applications. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT).* IEEE, 442–447.

[18] Nancy L Laplante, Phillip A Laplante, and Jeffrey M IEEE systems journal Voas. 2016. Stakeholder identification and use case representation for Internet-of-Things applications in healthcare. 12, 2 (2016), 1589–1597.

[19] Gianna Reggio. [n.d.]. A UML-based proposal for IoT system requirements specification. In *Proceedings of the 10th International Workshop on Modelling in Software Engineering.* 9–16.

[20] Sebastian Rohjans, Christian Dänekas, and Mathias Uslar. [n.d.]. Requirements for smart grid ICT-architectures. In *2012 3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe).* IEEE, 1–8.

[21] Peter Salhofer. 2018. Evaluating the FIWARE Platform. In *Proceedings of the 51st Hawaii International Conference on System Sciences.*

[22] W3C. 2021 (accessed March 18, 2021). *IoT-Lite Ontology.* https://www.w3.org/Submission/iot-lite.

[23] W3Cs. 2017 (accessed March 18, 2021). *Semantic Sensor Network Ontology.* https://www.w3.org/TR/vocab-ssn.

[24] Business Wire. 2020 (accessed Jan 18, 2020). *Global IoT Middleware Market (2020 to 2025) - Growth, Trends, and Forecasts.* https://www.businesswire.com/news/home/20201229005254/en/Global-IoT-Middleware-Market-2020-to-2025---Growth-Trends-and-Forecasts---ResearchAndMarkets.com/.