Internet of Things Multi-protocol Interoperability with Syntactic Translation Capability

Nedaa H. Ahmed[1]
Information Systems Department
Faculty of Computers and Information
Fayoum University, Fayoum, Egypt

Ahmed M. Sadek[2]
Computer Science Department
Faculty of Computers and Information
Fayoum University, Fayoum, Egypt

Haytham Al-Feel[3]
Computer Science Department
Community College
Imam AbdulRahman Bin Faisal University, Saudi Arabia

Rania A. AbulSeoud[4]
Electronic and Communication Department
Faculty of Engineering
Fayoum University, Fayoum, Egypt

***Abstract*—Because Internet of Things (IoT) systems contain different devices, infrastructures, and data formats; its success depends on the realization of full interoperability among these systems. Interoperability is a communication challenge that affects all the layers of the system. In this paper, a transparent translator to solve interoperability issues in two layers of an IoT system is proposed. The communication protocol layer is the first layer. In this layer, it is necessary to overcome the difference between the interaction patterns, such as request/response and publish/subscribe. The second layer includes the syntactic layer, which refers to data encoding. This type of interoperability is achieved through the semantic sensor network (SSN) ontology. Tests and evaluations of the proposed translator in comparison with a similar translator were performed using the constrained application protocol (CoAP), message queuing telemetry transport (MQTT) protocol, and hypertext transfer (HTTP) protocol, in addition to different data formats, such as JSON, CSV, and XML. The results reveal the efficiency of the proposed method in terms of application protocol interoperability. In addition, the suggested translator has the added feature that it supports different data encoding standards as compared to the other translator.**

Keywords—Internet of things (IoT); interoperability; multiprotocol translation; message payload translation; SSN ontology

I. Introduction

The IoT comprises a collection of different devices connected using different Internet protocols. Examples of these devices include the thermostats, air conditioners, and lightbulbs that can be found in smart homes. In addition, the IoT plays an important role in other domains, such as transportation, healthcare, industrial automation, smart cities, and agriculture. The IoT enables physical objects to perform actions and share data. Therefore, IoT intelligence is bestowed on these objects by using different technologies, such as cloud computing, embedded devices, sensor networks, and Internet protocols. Because of the diversity of IoT systems, many protocols have been developed and applied. Interoperability between the different systems represents an important factor in the success of an IoT; however, it remains a significant challenge. Interoperability problems can be found in different levels, such as at the device, messaging protocol, syntactic, and semantic levels.

Interoperability on the device level refers to the wide range of devices located in an IoT. These may be high- or low-end. Examples of high-end devices include Raspberry Pi and smartphones, which have ample resources and computational capabilities, whereas low-end devices include radio frequency identification tags, sensors, and devices with constrained resources [1]. These devices may support wired or wireless networking protocols, such as Ethernet, ZigBee, Bluetooth, ZWave, 3G/4G cellular technologies, and near-field communication. In addition, these protocols can be standard communication protocols or non-standard proprietary protocols, such as long range (LoRa) and SIGFOX [1]. Sometimes, the devices that need to share information use different network technologies, requiring that that the interoperability among these different devices and network technologies be resolved to enable their integration [1].

Interoperability on the messaging protocol level refers to the multiple application protocols that exist, such as the message queue telemetry transport (MQTT) protocol, constrained application protocol (COAP), and hypertext transfer protocol (HTTP), are used to provide communications. Each protocol has characteristics that support different types of IoT applications [2]. Nevertheless, the various IoT applications should be able to exchange messages independently of messaging protocols to allow a scalable IoT architecture and cross-domain applications. Thus, the success of the interoperability of messaging protocols is manifested in a system's ability to translate between these different messaging protocols.

Syntactic interoperability refers to the fact that the content types of the data sent through the communication protocols can be of different types. Some of the most frequently used data formats are extensible markup language (XML), JavaScript object notation (JSON), and comma-separated values (CSV). The syntactic interoperability problem arises when the sender encodes the message in a specific format and the receiver can decode received messages only in a different format. Thus, the encoding rules of the sender are incompatible with the decoding rules of the receiver, leading to mismatching of messages [1]. Therefore, this level of interoperability is important to allow a smooth transition of messages among different IoT systems.

The final type of interoperability discussed in this paper is semantic interoperability. The World Wide Web Consortium (W3C) defined it as ``enabling different agents, services, and applications to exchange information, data and knowledge in a meaningful way, on and off the web" [3]. In the traditional IoT scenario, raw sensor data from heterogeneous nodes are provided to the software agent [1]. These data contain no semantic annotations and an extensive effort is required to build intelligent applications. In addition, they may be represented in different units of measurements and have additional information [1], resulting in semantic interoperability problems. These semantic problems between data and information models render IoT applications unable to interoperate both automatically and dynamically because their descriptions and understanding of resources differ [1].

The current middlewares used to achieve application protocol interoperability have limitations, such as adding interoperability problems when working in conjunction with another middleware. Also, some proxies are used to solve the same level of interoperability but have issues like low bandwidth, low processing, and high cost of management. On the other hand, the current works for solving syntactic interoperability problems are very few. The existing solutions can convert between encodings with similar syntax only. Also, there is no current solution that achieves both application protocol and syntactic interoperability together.

In this study, a software architecture was designed to solve the interoperability problems related to both messaging protocols and syntactic levels. The main contributions of this paper are as follows.

1) An IoT translator that can achieve communication protocol and syntactic interoperability is proposed. Semantic interoperability will be addressed in a future paper.

2) The development of a multi-protocol translator that can translate messages among the CoAP, MQTT, and HTTP protocols is described.

3) The use of the semantic sensor network (SSN) ontology to allow conversion among XML, JSON, and CSV data formats to achieve syntactic interoperability is described. This will enable clients to obtain the data they need in any required format, even if they are stored in different formats on the server.

4) A translator based on a hub-and-spoke model, which supports scalability and modularity, is presented. The scalability and modularity will allow the translator to be extended to support more protocols easily, in addition to more data formats.

5) An evaluation is presented that shows the effectiveness of the proposed translator in comparison with the Arrowhead translator.

The remainder of the paper is organized as follows. In Section II, some related studies on solutions for interoperability problems in the IoT are reviewed. Sections III and IV discuss the software architecture and implementation of the proposed solution, respectively. The results and evaluations are presented in Section V. Finally, the conclusions and suggestions for future work are provided in Section VI.

II. Related Work

Different architectures and frameworks are used to solve IoT interoperability on the various levels.

To address syntactic interoperability, Palm et al. [4] presented a theoretical method for translating message payloads among different endpoints. First, this method constructs a syntax tree from the incoming message. Then, it converts the syntax tree into an equivalent syntax tree of the target encoding. The syntactic translation can convert only between an encoding standard and intersecting syntaxes.

To provide messaging protocol interoperability, Derhamy et al. [5] proposed a transparent protocol translator to allow interoperability between communication protocols. This translator depends on a service-oriented architecture (SOA), not on middleware. Thus, it supports low latency and operates transparently. It is also secured through the use of Arrowhead authorization and authentication [6]. Its architecture consists of two spokes and a central hub: the first spoke operates as a service provider spoke and the second as a service consumer spoke. The translator can support any number of protocols, each of which has only two spokes. The authors tested their architecture on the CoAP and HTTP protocols and determined that it was faster than the Californium proxy [7].

Lee et al. [8] proposed an IoT framework based on the software-defined network (SDN) that can intercept all packets from CoAP to MQTT and vice versa. They defined URL rules to specify the resource or the topic and distinguish between homogeneous (e.g., from MQTT client to MQTT client) and heterogeneous (e.g., from MQTT client to CoAP client) traffic. In the homogeneous scenario, the SDN ignores the traffic and these packets are operated as in the original scenario. In the heterogeneous scenario, the SDN switch delivers the packets to the SDN controller and redirects them to the cross proxy for translation. The advantage of this framework is that it causes no delay in a homogeneous scenario. However, the authors provided no evaluation results for their suggested framework.

Ponte [9] is an Eclipse IoT project that provides open APIs to create applications that support the CoAP, HTTP, and MQTT communication protocols. Ponte provided a centralized solution to enable clients using different communication protocols to communicate easily with each other. Data collected from the three different protocols are stored in SQL or NoSQL databases. Therefore, all the clients can access all resources, regardless of the communication protocol they use. In the same direction, Khaled and Helal [10] proposed the Atlas IoT framework to allow communication between clients using MQTT, CoAP, and HTTP. The proposed protocol translator can be deployed on either a cloud infrastructure or the IoT device itself. This framework depends on the IoT device description language [11] and was compared with Ponte

[9]. The results show that energy consumption is reduced in comparison with that of Ponte [9]. A disadvantage of this framework is that not all IoT devices can be part of this interoperable ecosystem: the device should have an Internet connection (e.g., Ethernet, cellular network, or Wi-Fi) and an operating system that supports multithreading to integrate with this ecosystem.

Desai et al. [2] proposed the Semantic Gateway as Service architecture to achieve messaging protocol interoperability among the CoAP, MQTT and XMPP protocols using a multi-protocol proxy. Semantic reasoning using the SSN ontology to solve interoperability was highlighted, but no description of implementation details or evaluation methodology were provided.

To address semantic level interoperability, Gyrard et al. [12] proposed the machine-to-machine measurement (M3) framework to develop semantic-based cross-domain IoT projects and reuse the optimum number of ontologies and rules. In their study, they focused on designing the Linked Open Vocabularies for the Internet of Things dataset to reference and classify semantic-based projects that are relevant to the IoT. In addition, they designed a sensor-based linked open rules dataset of domain rules to infer high-level abstractions from sensor data. David Perez et al. [13] developed an ontology for the smart city scenario, specifically for the SusCity project [14], to facilitate the management of the infrastructure. This ontology consists of several main classes, such as IoT infrastructures, devices, communication interfaces and links, and performance metrics. Evaluations proved the correctness of this ontology. One of its disadvantages is that an automatic ontology update mechanism is required. Gyrard and Serrano [15] presented a methodology called SEG 3.0, a name which comes from segmentation and Web 3.0, which depends on semantic Web technologies. They defined the characteristics and steps of this methodology and subsequently implemented a framework for applying it. The purpose of this framework is to achieve semantic interoperability among IoT projects. In addition, they investigated various use cases to show the correctness of the methodology and that it can be applied to other domains, such as smart cities. Kleine et al. [16] developed a Semantic Web of Things architecture including virtual sensors, smart service proxy, and semantic entities to measure the traffic density of a road. This architecture depends on smartphones located in vehicles moving in the network and represents the sensor data in resource description framework (RDF) form. In the future, the authors plan to enable selective privacy to identify the exact vehicles traveling on certain road sections. Additionally, Kamilaris et al. [17] proposed an eco-system for urban computing, using the concept of the Web of Things together with event processing, mobile computing, semantic Web, and big data analysis techniques to record the real information of smart cities for their residents. They conducted a case study in the city of Aarhus, Denmark. One of the disadvantages of the suggested ecosystem is its lack of privacy or security aspects.

Kamilaris et al. [18] proposed a semantic framework called Agr-IoT for smart farming applications. This framework uses semantic Web techniques to allow reasoning and to facilitate increased information collection and more accurate decision making. In addition, semantic Web techniques help achieve interoperability among different data sources, such as sensors, social media, governmental alerts, connected farms, and regulations. The ontologies used in the framework are the SSN ontology, complex event service ontology, and an agriculture ontology, called AgOnt.

In the healthcare domain, Zgheib et al. [19] presented an IoT system to detect the risk of bedsores, using SSN ontology to achieve interoperability between system components. This system is based on message-oriented middleware to process some constraints, such as the security, scalability, and privacy of medical information.

The above review shows that the methods that achieve messaging protocol and syntactic interoperability remain few in number. Therefore, the proposed architecture is focused on these two levels of interoperability.

III. Proposed Architecture

The proposed architecture is based on that presented in [5]. As shown in Fig. 1, it consists of three parts: the clients, translator, and servers. The clients are service consumers who send different requests to different servers. They can also receive data in any format they need, even if these data are stored in a different format on the server. The proposed translator is used to allow a client using a specific protocol to communicate with a server using a different protocol. It consists of a hub and multiple spokes. As this translator can translate among three different protocols, it has six spokes. It can support additional protocols by adding only the two spokes needed for each of these protocols. At each operation, only two spokes are used according to the request. The hub is a conceptual representation and is represented using an intermediate format and SSN ontology. The intermediate format is used to convert one protocol to another. Each protocol is represented by two spokes. The first spoke is a server spoke, which listens to different requests on a specific port, and the second is the client spoke, which creates a new request according to the information received from the intermediate format in the hub. Inside the hub, the SSN ontology is also located, which is used to describe sensors and their observations, the features of interest, the observed properties, and the actuators [20]. It is used only to achieve syntactic interoperability and is converted into different data formats. The final part of the architecture is the server, which contains the service providers that serve different clients. The proposed translator can be used with different clients and with servers acquired from different vendors.

The following scenarios clarify the architecture.

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

A. *Request from Representational State Transfer (REST) Client to REST Server*

In this scenario, the representational state transfer (REST) points to the HTTP and CoAP protocols. Therefore, this scenario illustrates the events that occur when a CoAP client sends a request to an HTTP server or an HTTP client sends a request to a CoAP server. As shown in Fig. 2, the server spoke, a CoAP server in this case, listens to the requests from a CoAP service consumer. A CoAP request is converted to the appropriate HTTP request using the intermediate format. Subsequently, the HTTP request is forwarded to the HTTP service provider.

If the syntactic interoperability service is required, the SSN ontology is used. In this case, if the request is PUT or POST, an INSERT or UPDATE statement modifies the SSN ontology. However, if the request is GET, there exist two situations. The first is where the service consumer requires the payload of a specific resource in the same format as that of the service provider. In this case, the service provider replies directly with the payload to the HTTP client spoke without needing to use the SSN ontology. When the HTTP client spoke receives the response, a CoAP response is generated by the CoAP server spoke using the intermediate format and is forwarded to the CoAP consumer. In the second situation, the service consumer requires the payload of a specific resource in a format different from that which exists in the service provider. In this case, the service provider replies with a "NOT ACCEPTABLE" code to the HTTP client spoke. If the CoAP server spoke receives a "NOT ACCEPTABLE" message, it generates a SELECT query to obtain the specified resource data in a standard format, converts this format to the required format, and finally responds to the CoAP client consumer with these data. The same steps are taken when the HTTP client requires a service from the CoAP server. The only difference is that the service consumer in this case is an HTTP client and the service provider is a CoAP server. An additional difference is that, inside the translator, the two spokes that should be used in this case are the HTTP server and the CoAP client spokes. The mapping between the HTTP and CoAP protocols is shown in Table I. However, it is important to note that the CoAP protocol contains an OBSERVE request that does not exist in the HTTP protocol. The CoAP OBSERVE request was used only with the MQTT protocol.

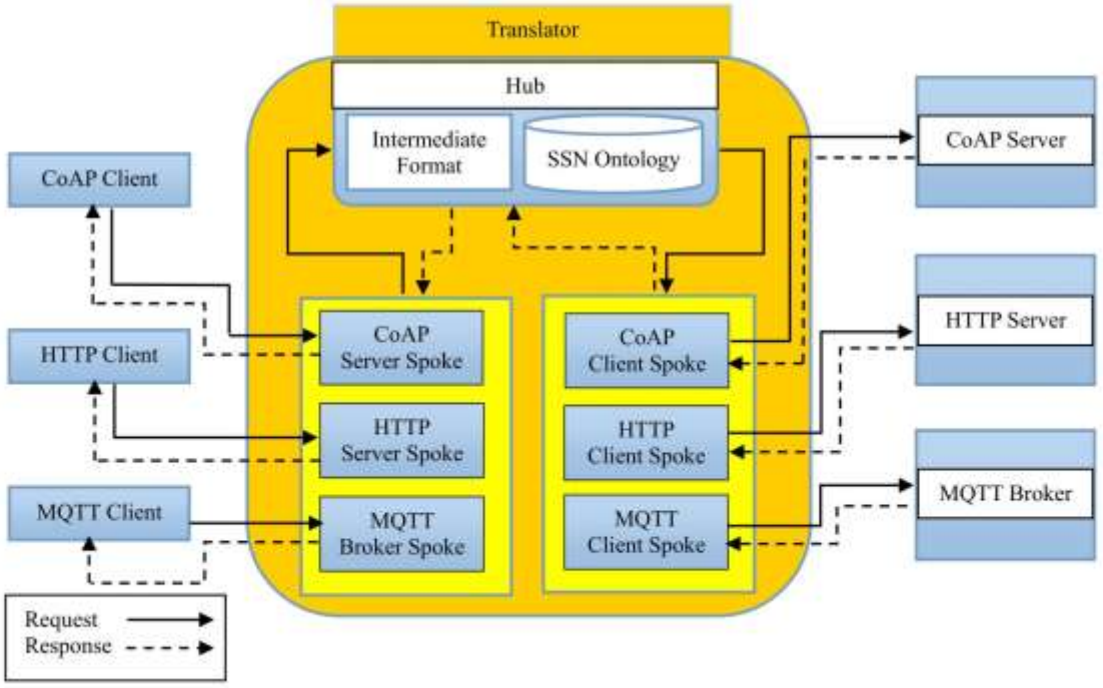


Fig. 1. Proposed Architecture.

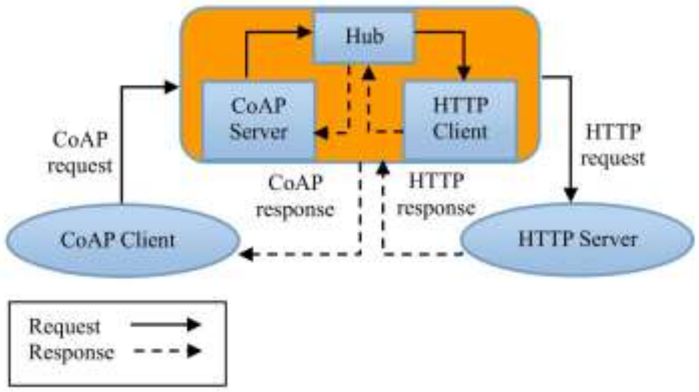


Fig. 2. Request from Constrained Application Protocol Client to Hypertext Transfer Protocol Server.

B. *Publishing a Resource in Representational State Transfer Server (Constrained Application Protocol or Hypertext Transfer Protocol)*

As shown in Fig. 3, in this scenario the service consumer is an MQTT client and the service provider is a REST server. Thus, the MQTT client must publish a message on any resource in the REST server.

The MQTT broker spoke listens to the PUBLISH messages from the MQTT service consumer. The MQTT message is converted to the PUT REST request using the intermediate format in the hub. Subsequently, the PUT request is forwarded to the REST service provider. In addition, the SSN ontology is required only if the syntactic interoperability is activated. In that case, the MQTT server spoke updates the value of the resource in the SSN ontology. Note that UPDATE query exists in the SPARQL query language, and therefore, DELETE and INSERT queries must be used together to update the data in the SSN ontology. As the MQTT protocol does not contain a content format field in its header, the content format is checked manually. According to the content format, it is possible to determine the manner in which the message should be processed to extract the resource or topic name with its payload and store them in the SSN ontology. The pseudocode for this is shown in Algorithm 1. When the REST client spoke receives a response, an MQTT message response is generated by the MQTT server spoke and forwarded to the MQTT publisher. The response of the PUBLISH message can be either a None or PUBACK packet or a PUBREC packet according to the quality of service (QoS) level [21].

C. *Subscribing a Resource in Constrained Application Protocol Server*

The subscription of a message to the CoAP server differs slightly from that to the HTTP server, because the HTTP protocol does not have an OBSERVE request as does the CoAP server. As shown in Fig. 4, the main steps in this scenario are the same as described previously. The difference is that the SUBSCRIBE message in the MQTT protocol corresponds to the OBSERVE request in the CoAP protocol. As the MQTT protocol does not contain the content format field in its header, it is not necessary to use the SSN ontology, although syntactic interoperability is activated.

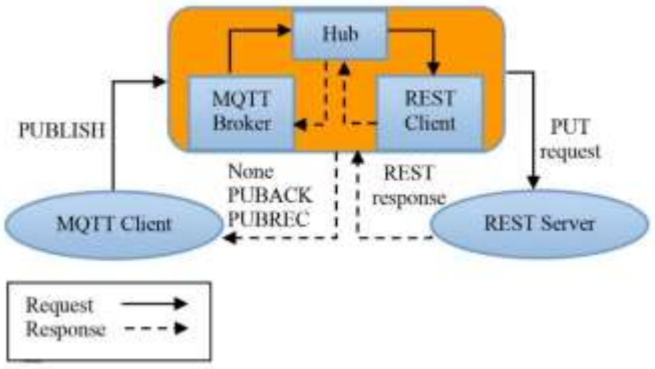


Fig. 3. PUBLISH Message from Message Queue Telemetry Transport Client to Representational State Transfer Server.

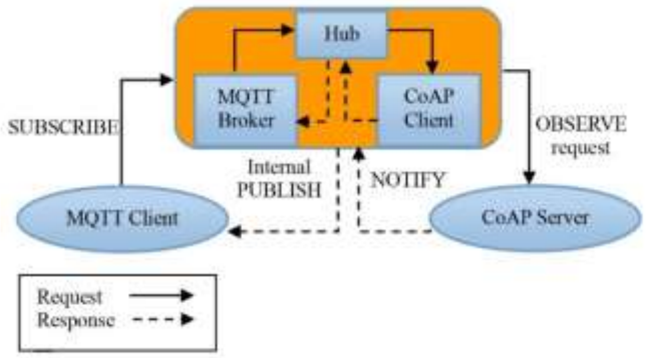


Fig. 4. SUBSCRIBE Message from Message Queue Telemetry Client to Constrained Application Protocol Constrained Application Protocol Server.

D. *Subscribing a Resource in the Hypertext Transfer Protocol Server*

As mentioned above, the HTTP protocol does not have an OBSERVE request. Therefore, the GET request, which is implemented periodically, is used. As shown in Fig. 5, when the MQTT client subscribes to a specific resource in the HTTP server, an HTTP client makes a GET request with a pre-configured periodic time. If the returned payload is different from the last payload, the MQTT broker publishes internally.

Algorithm 1: Checking the content format for the MQTT protocol

```
Input: A string variable message which is MQTT payload
Output: An integer variable type which is the format of MQTT payload
initialization;
if message begins with "{" and ends with "}" then
 type ←0; // zero means it is JSON format
end
else if message begins with "<" and ends with ">" then
 type ←1; // one means it is XML format
end
else if the number of commas is equal in each line then
 type ←2; // two means it is CSV format
end
else
 type ←3; // three means it is not supported format
end
return type
```

with the new payload to the MQTT client. The periodic GET request would be canceled if the MQTT client unsubscribed on this topic.

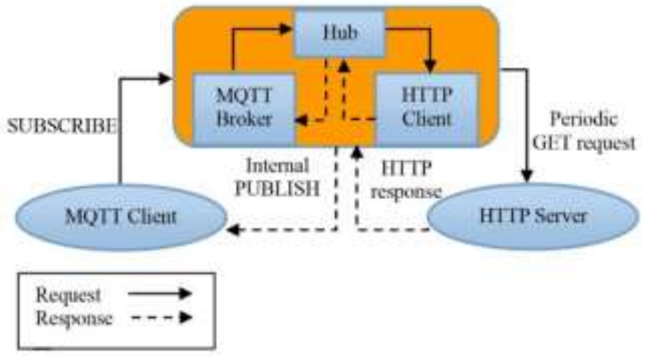


Fig. 5. SUBSCRIBE Message from Message Queue Telemetry Transport Client to Hypertext Transfer Protocol Server.

IV. Implementation

The architecture was implemented using the JAVA language, and the spokes were implemented using the Java libraries. For example, the CoAP spokes, for both client and server, were implemented using the Californium CoAP library [7]. The CoAP server spoke contained only the CoAP server with RootResource [7]. The CoAP client spoke was used to perform the CoAP request and return the response to the calling spoke. For the MQTT protocol, the MQTT server spoke was implemented using the Mosquitto broker [22], and the MQTT client spoke was implemented using the Eclipse.

Paho client developed by Eclipse Foundation; however, the Jersey open-source libraries created by Eclipse Foundation and Oracle Corporation for the HTTP protocol were used to implement the HTTP server and HTTP client spokes.

A. *Mapping among different Protocols*

The mapping among the three different protocol spokes was implemented as shown in Table I.

B. *Intermediate Format*

The intermediate format was used in the hub to enable interchanges between the protocol spokes. It holds the basic header fields of request and response. The structure of the intermediate format is presented in Table II.

C. *Semantic Sensor Network Ontology*

The SSN ontology [20] is an ontology developed by W3C to provide standard modeling for sensor devices, actuators, sensor platforms, their observations, and so on. It was used in the RDF format [23]. The purpose of using the SSN ontology is to achieve syntactic and semantic interoperability. Here, the syntactic interoperability was achieved to solve the problem of encoding and decoding messages between the sender and receiver in different formats. By doing so, if, for example, the payload is stored in the plaintext format at the REST server, the REST client can obtain the payload in different formats, such as JSON or XML. This study converted three different formats: CSV, JSON, and XML. In Fig. 6, a subset of this ontology is shown. It represents the classes and properties that used in the proposed architecture.

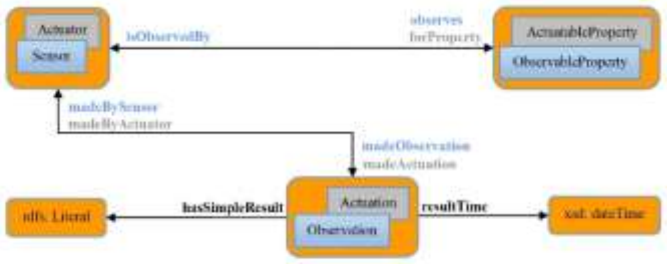


Fig. 6. Subset of Semantic Sensor Network Ontology.

TABLE I. Mapping among Three Protocols

	HTTP	CoAP	MQTT
Code Request	GET request in case of mapping with GET request of CoAP and with the MQTT SUBSCRIBE.	GET request in case of mapping with GET request of HTTP. OBSERVE request in case of mapping with MQTT SUBSCRIBE	SUBSCRIBE
	PUT request	PUT request	PUBLISH retained message
	POST request	POST request	PUBLISH retained message
	DELETE request	DELETE request	PUBLISH retained message with zero-byte payload
Code Response	“200” OK	“2.00” OK	0x00 - Success - Maximum QoS 0 0x01 - Success - Maximum QoS 1 0x02 - Success - Maximum QoS 2
	“404” NOT FOUND	“4.04” NOT FOUND	Not supported
	“406” NOT ACCEPTABLE	“4.06” NOT ACCEPTABLE	Not supported
	“204” NO CONTENT	“2.04” CHANGED	Not supported
Code Error	“400” BAD REQUEST	“4.00” BAD REQUEST	0x02 Connection Refused, identifier rejected
	“401” UNAUTHORIZED	“4.01” UNAUTHORIZED	0x05 Connection Refused, not authorized
	“500” INTERNAL SERVER ERROR	5.00” INTERNAL SERVER ERROR	0x03 Connection Refused, Server unavailable
Object	Resource name	Resource name	Topic name

TABLE II. Definition of Intermediate Format

Variable Name	Type	Discussion
uniqueKey	int	is the message Id
Code	String	is the CRUD operation in case of request and response code or error code in case of response
Object	String	is the resource or topic to be operated on
Payload	String	is the body of the message
payloadFormat	String	is the format of the payload

V. Testing and Evaluation

In this simulation, a CoAP server, HTTP server and MQTT broker were used in a weather information service transmitting information from different geographic locations. This service measured the temperature, humidity, pressure, wind direction, and wind speed. The HTTP and CoAP protocols could support different formats of the payload, such as plaintext, CSV, JSON, and XML formats. However, in the MQTT protocol, the content format was application-specific, as this protocol does not contain the content format or accept fields in its header. If the service used one sensor and provided only the value, the plaintext format was used; otherwise, any one of the three formats was used. The XML, CSV, and JSON payload structures are shown in Fig. 7(a), Fig. 7(b), and Fig. 7(c), respectively. The approximate lengths of each format are listed in Table III.

For converting the payload from one format to another, the intermediary ontology was used, as it represents the data in a structured form.

The delay caused by the translator for protocol translation and format conversion was measured and evaluated. All the tested scenarios are shown in Fig. 8. In test 8-a), a CoAP request was generated from a CoAP client to the translator, which then generated the corresponding HTTP request to the HTTP server.

Test 8-b) followed the form of test 8-a), except that the translator generated the corresponding MQTT message to the MQTT broker. However, test 8-c) involved an HTTP request generated from an HTTP client to the translator, which then generated the corresponding CoAP request to the CoAP server. Test 8-d) was similar to test 8-c), except that the translator generated the corresponding MQTT message to the MQTT broker. Test 8-e) involved an MQTT message generated from an MQTT client to the translator, which then generated the corresponding HTTP request to the HTTP server. Finally, test 8-f) followed the form of test 8-e), except that the translator generated the corresponding CoAP request to the CoAP server.

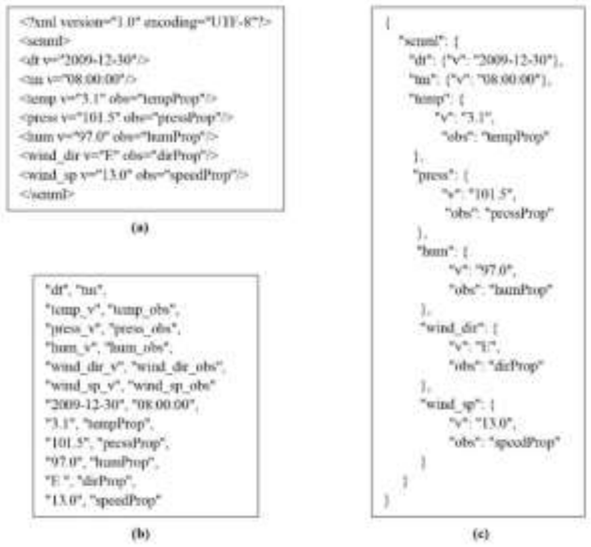
The translator was run on a laptop with an Intel Core i5-2520M processor running Windows 8 at 2.50 GHz and 4.00 GB RAM. The delay introduced by the translator and the delay caused by using the SSN ontology were measured. As shown in Fig. 9, six Java milliseconds timers (t_1 - t_6) were used to compute these delays. The descriptions of these timers are listed in Table IV.

The following equations were used in the delay calculations, where (1) represents the time the packet takes within the translator and (2) the time required to perform processing on the different formats of data plus the execution time of the SPARQL query.

$$D_{translate} = (t_2 - t_1) + (t_4 - t_3) \quad (1)$$

$$D_{ssnOnt} = t_6 - t_5 \quad (2)$$

The tests were performed 1000 times per scenario. The average time required for protocol translation is summarized in Table V.



```
<?xml version="1.0" encoding="UTF-8"?>
<senml>
<dt v="2009-12-30"/>
<tm v="08:00:00"/>
<temp v="3.1" obs="tempProp"/>
<press v="101.5" obs="pressProp"/>
<hum v="97.0" obs="humProp"/>
<wind_dir v="E" obs="dirProp"/>
<wind_sp v="13.0" obs="speedProp"/>
</senml>
```

(a)

```
"dt", "tm",
"temp_v", "temp_obs",
"press_v", "press_obs",
"hum_v", "hum_obs",
"wind_dir_v", "wind_dir_obs",
"wind_sp_v", "wind_sp_obs"
"2009-12-30", "08:00:00",
"3.1", "tempProp",
"101.5", "pressProp",
"97.0", "humProp",
"E ", "dirProp",
"13.0", "speedProp"
```

(b)

```
{
  "senml": {
    "dt": {"v": "2009-12-30"},
    "tm": {"v": "08:00:00"},
    "temp": {
          "v": "3.1",
           "obs": "tempProp"
     },
     "press": {
           "v": "101.5",
            "obs": "pressProp"
      },
      "hum": {
            "v": "97.0",
            "obs": "humProp"
       },
      "wind_dir": {
            "v": "E",
            "obs": "dirProp"
       },
      "wind_sp": {
            "v": "13.0",
            "obs": "speedProp"
       }
    }
}
```

(c)

Fig. 7. Different Payload Structure Formats.

TABLE III. Length of Payloads in Bytes

The Payload Structure	Length (bytes)
XML	236
JSON	246
CSV	234

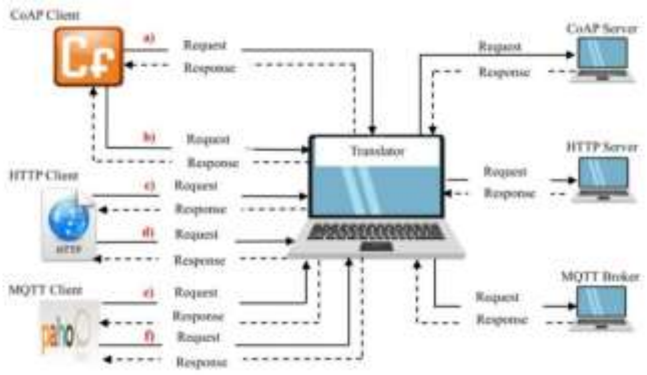


Fig. 8. Test Scenarios.

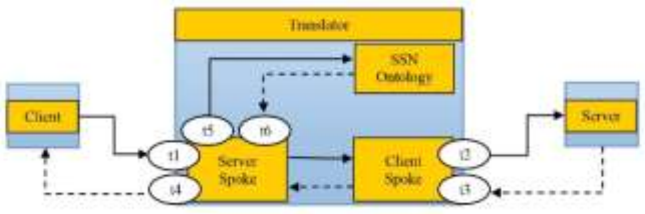


Fig. 9. Time Stamp for Delay Measurements.

As shown in Table V, the time required to convert the protocols is very short. Scenarios (e) and (f) have the minimum delay because the server spoke in this case is the MQTT broker. The MQTT broker operates as a pipeline and does not perform any processing, as do HTTP and CoAP servers. Therefore, the MQTT broker is very simple as compared to the CoAP and HTTP servers.

TABLE IV. TIMING INSTRUMENTATION

t_1	=	Request arrives at the application server spoke of the translator.
t_2	=	Request leaves the application client spoke of the translator.
t_3	=	Response arrives at the application client spoke of the translator.
t_4	=	Response leaves the application server spoke of the translator.
t_5	=	Query leaves the server to the SSN ontology plus preprocessing the data if needed.
t_6	=	Response from the SSN ontology plus processing the data if needed.

TABLE V. AVERAGE DELAY OF THE PROTOCOL TRANSLATION

Scenario	**Client**	**Server**	**Method**	**Delay (ms)**
a)	CoAP Client	HTTP Server	GET	3.96
			PUT	4.03
b)	CoAP Client	MQTT Broker	GET	3.10
			PUT	3.05
c)	HTTP Client	CoAP Server	GET	3.21
			PUT	3.38
d)	HTTP Client	MQTT Broker	GET	3.81
			PUT	3.09
e)	MQTT Client	HTTP Server	Subscribe	3.10
			Publish	2.64
f)	MQTT Client	CoAP Server	Subscribe	2.45
			Publish	2.21

TABLE VI. AVERAGE PROCESSING TIME

Scenario	**From**	**To**	**Delay (ms)**
1)	JSON		23.69
2)	XML	SSN	23.57
3)	CSV		30.19
4)		JSON	11.78
5)	SSN	XML	11.86
6)		CSV	11.71

In Table VI, the average computed delay in achieving syntactic interoperability is shown. Scenarios 1, 2, and 3 represent the delay in processing and updating the data in the SSN ontology. These three scenarios are used when performing a POST, PUT, or PUBLISH request. Scenarios 4, 5, and 6 represent the average delays caused by selecting the data from the SSN ontology and converting it to the required format. These scenarios were used when performing GET requests. As can be seen in this table, updating the ontology (Scenarios 1–3) consumes more time than selecting the data from it (Scenarios 4–6). However, this delay is acceptable in IoT applications and does not affect the performance of the architecture. Using the SSN ontology, different clients can receive the payload in any format they require, even if this payload exists in the server in a different content format.

VI. DISCUSSION

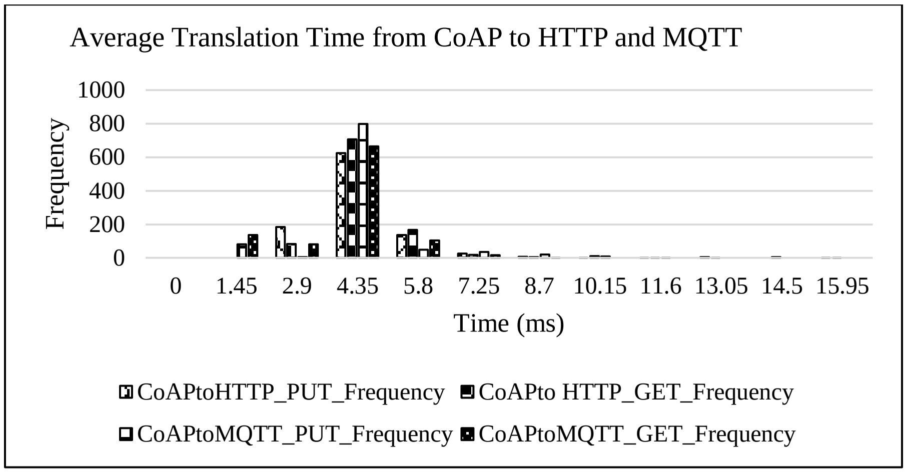
To validate the efficiency of the suggested translator, the delay caused by the proposed translator was compared with that caused by the Arrowhead translator [5]. The comparison was applied only to scenario (c) in Fig. 8, as the authors implemented only this scenario. The Arrowhead translator was run on hardware that was different from that used here on the proposed translator. For this reason, it was unavailable to compare the proposed translator with the Arrowhead translator directly. In the case of the Arrowhead translator, the authors measured and evaluated the delay introduced in comparison with the Californium proxy [22]. They ran the Californium proxy on the same hardware they used for running their proposed translator. The same procedure was followed to evaluate the proposed translator in comparison with the Arrowhead translator.

As can be seen in Table VII, the Arrowhead translator's delay is about 43.5\% of that of the Californium proxy when implemented on the same platform, whereas the suggested translator`s delay is only about 23.6\% of that of the Californium proxy on the same platform. In addition, the proposed translator can map between different data encoding standards, a capability that is not available in the Arrowhead translator. In Fig. 10 and 11, the results of 1000 requests for each scenario are shown.

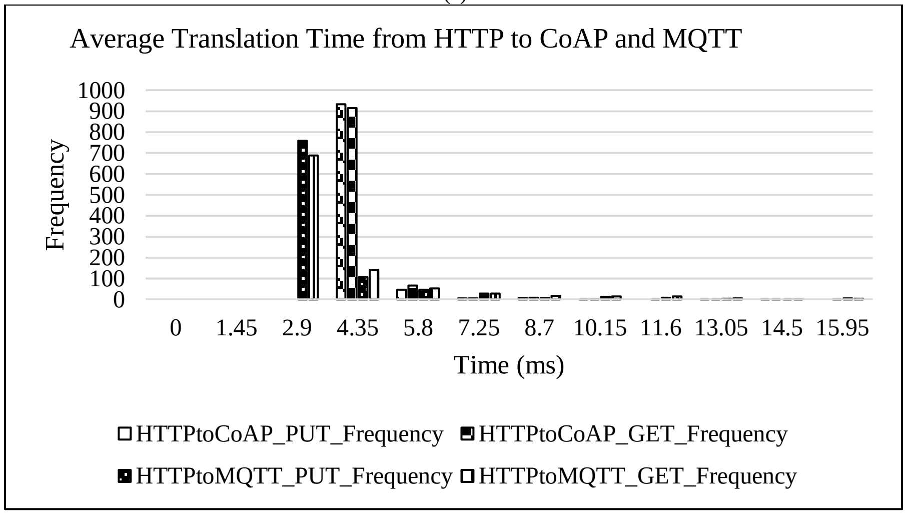
The histogram charts show that there exists some anomaly. However, this anomaly is due to the Java libraries that were used, and therefore, it was beyond control.

TABLE VII. COMPARISON OF THE PROPOSED AND THE ARROWHEAD TRANSLATOR

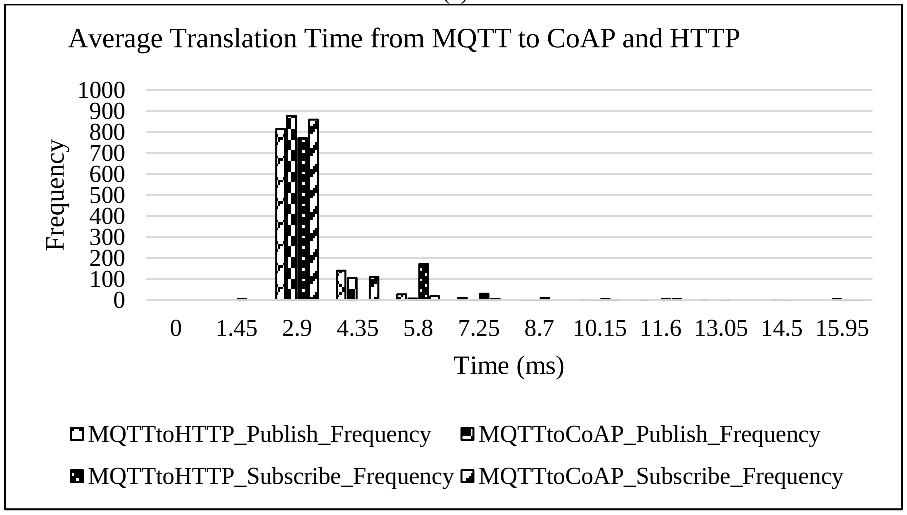
The suggested Translator		The Arrowhead Translator	
Delay of it (ms)	Delay of CP on the same hardware (ms)		Delay of it (ms)
3.21	13.59	177	77



(a)

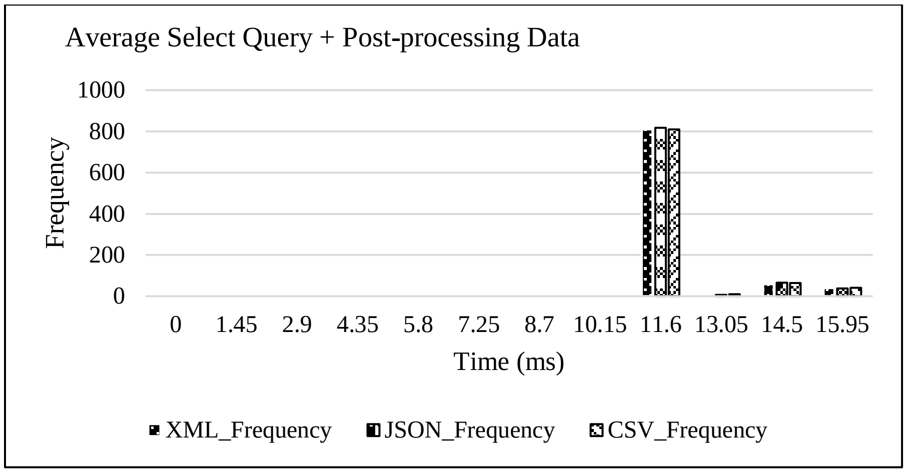


(b)

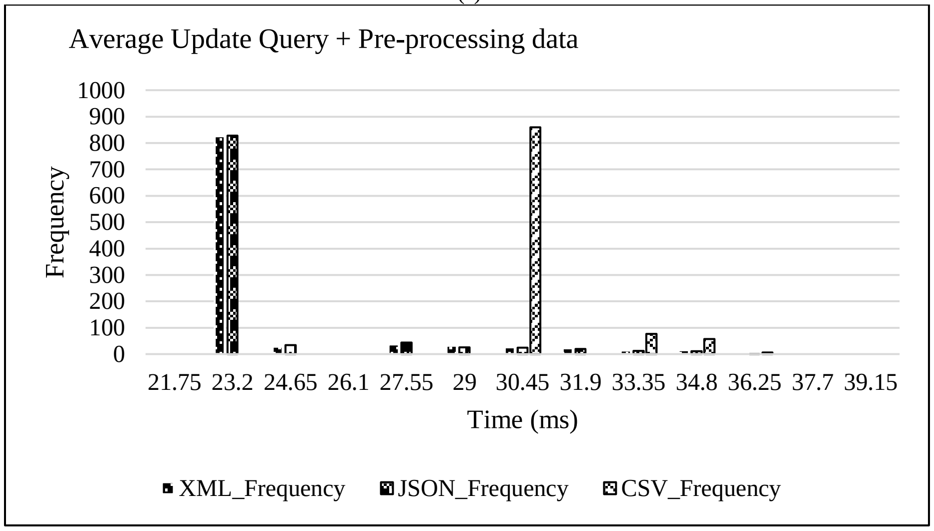


(c)

Fig. 10. Average Translation Time among different Protocols.



(a)



(b)

Fig. 11. Average Delay of Query on Semantic Sensor Network Ontology and Processing Data.

VII. Conclusion

One of the greatest challenges in the IoT is to achieve interoperability. A transparent and reliable architecture based on SOAs was suggested in this paper. The goal was to achieve protocol and syntactic interoperability using the SSN ontology. The proposed architecture can integrate different IoT application protocols, such as MQTT, HTTP, and CoAP. In addition, it can convert between different payload formats, such as JSON, XML, and CSV.

The evaluations showed that the proposed translator's performance is better than that of the Arrowhead translator and introduced a shorter delay. The proposed translator`s delay is approximately 19.9\% of that of the Arrowhead translator. This difference in delay is due to the simplicity of the implementation. In addition, the suggested translator has an advantage over the Arrowhead translator in that it can achieve syntactic interoperability, unlike the Arrowhead translator using SSN ontology.

Future work will attempt to achieve semantic interoperability using the current architecture by reasoning the data stored in the SSN ontology. For this reason, it was preferred to use semantic Web technology to achieve syntactic interoperability rather than any other database. This allowed us to achieve two types of interoperability using one technology without introducing a long delay.

References

[1] M. Noura, M. Atiquzzaman, and M. Gaedke, "Interoperability in internet of things: taxonomies and open challenges," Mob. Networks Appl., vol. 24, no. 3, pp. 796–809, 2019.

[2] P. Desai, A. Sheth, P. Anantharam, "Semantic gateway as a service architecture for IoT interoperability," Proc. - 2015 IEEE 3rd Int. Conf. Mob. Serv. MS 2015, pp. 313–319, Aug. 2015.

[3] A. D. P. Venceslau, R. M. C. Andrade, V. M. P. Vidal, T. P. Nogueira, and V. M. Pequeno, "IoT semantic interoperability: A systematic mapping study," ICEIS 2019 - Proc. 21st Int. Conf. Enterp. Inf. Syst., vol. 1, no. Iceis, pp. 523–532, 2019.

[4] E. Palm, C. Paniagua, U. Bodin, O. Schel ́en, "Syntactic translation of message payloads between at least partially equivalent encodings," Proc. IEEE Int. Conf. Ind. Technol., vol. 2019-Febru, pp. 812–817, 2019.

[5] H. Derhamy, J. Eliasson, J. Delsing, "IoT interoperability—on-demand and low latency transparent multiprotocol translator," IEEE Internet Things J., vol. 4, no. 5, pp. 1754–1763, 2017.

[6] P. Varga et al., "Making system of systems interoperable – The core components of the arrowhead framework," J. Netw. Comput. Appl., vol. 81, pp. 85–95, Mar. 2017.

[7] M. Kovatsch, M. Lanter, Z. Shelby, “Californium: scalable cloud services for the internet of things with coap,” 2014 Int. Conf. Internet Things, IOT 2014, pp. 1–6, 2014.

[8] C.-H. Lee, Y.-W. Chang, C.-C. Chuang, Y. H. Lai, “Interoperability enhancement for internet of things protocols based on software-defined network,” 2016 IEEE 5th Glob. Conf. Consum. Electron. GCCE 2016, pp. 4–5, 2016.

[9] M. Dave, M. Patel, J. Doshi, and H. Arolkar, “Ponte Message Broker Bridge Configuration Using MQTT and CoAP Protocol for Interoperability of IoT,” Commun. Comput. Inf. Sci., vol. 1235 CCIS, pp. 184–195, Mar. 2020.

[10] A. E. Khaled, S. Helal, “Interoperable communication framework for bridging restful and topic-based communication in IoT,” Futur. Gener. Comput. Syst., vol. 92, pp. 628–643, 2019.

[11] A. E. Khaled, A. Helal, W. Lindquist, C. Lee, “Iot-ddl–device description language for the “t” in IoT,” IEEE Access, vol. 6, pp. 24048–24063, 2018.

[12] A. Gyrard, C. Bonnet, K. Boudaoud, M. Serrano, “LOV4IoT:a second life for ontology-based domain knowledge to build semantic web of things applications,” Proc. - 2016 IEEE 4th Int. Conf. Futur. Internet Things Cloud, FiCloud 2016, pp. 254–261, 2016.

[13] D. P. Abreu, K. Velasquez, A. M. Pinto, M. Curado, E. Mon-teiro, “Describing the internet of things with an ontology: The suscity project case study,” Proc. 2017 20th Conf. Innov. Clouds, Internet Networks, ICIN 2017, pp. 294–299, 2017.

[14] J. Fernandes et al., “Building a smart city IoT platform - the suscity approach”, 48nd Spanish Congr. Acoust. Iber. Encount. Acoust., pp. 557–566, 2017.

[15] A. Gyrard, M. Serrano, “Connected smart cities: interoperability with seg 3.0 for the internet of things,” Proc. - IEEE 30th Int. Conf. Adv. Inf. Netw. Appl. Work. WAINA 2016, no. 2, pp. 796–802, 2016.

[16] O. Kleine, S. Ebers, M. Leggieri, “Monitoring urban traffic using semantic web services on smartphones - a case study,” 2015 12th Annu. IEEE Int. Conf. Sensing, Commun. Netw. - Work. SECON Work. 2015, pp. 1–6, 2015.

[17] A. Kamilaris, A. Pitsillides, F. X. Prenafeta-Bold, M. I. Ali, “A web of things based eco-system for urban computing-towards smarter cities,” Proc. 24th Int. Conf. Telecommun. Intell. Every Form, ICT 2017, 2017.

[18] A. Kamilaris, F. Gao, F. X. Prenafeta-Boldu, M. I. Ali, “Agri-IoT: a semantic framework for internet of things-enabled smart farming applications,” 2016 IEEE 3rd World Forum Internet Things, WF-IoT 2016, pp. 442–447, 2017.

[19] R. Zgheib, R. Bastide, E. Conchon, “A semantic web-of-things architecture for monitoring the risk of bedsores,” Proc. - 2015 Int. Conf. Comput. Sci. Comput. Intell. CSCI 2015, pp. 318–323, 2016.

[20] M. Compton et al., “The SSN ontology of the W3C semantic sensor network incubator group,” J. Web Semant., vol. 17, pp. 25–32, Dec. 2012.

[21] J. Toldinas, B. Lozinskis, E. Baranauskas, and A. Dobrovolskis, “MQTT Quality of Service versus Energy Consumption,” Proc. 23rd Int. Conf. Electron. 2019, Electron. 2019, Jun. 2019.

[22] R. A. Light, “Mosquitto : server and client implementation of the MQTT protocol,” J. Open Source Softw., vol. 2, pp. 1–2, 2017.

[23] J. Z. Pan, “Resource Description Framework,” in Handbook on Ontologies, Springer, Berlin, Heidelberg, 2009, pp. 71–90.