Open Access

Open Journal of Internet of Things (OJIOT)
Volume 8, Issue 1, 2022

www.ronpub.com/ojiot
ISSN 2364-7108

IoT Hub as a Service (HaaS): Data-Oriented Environment for Interactive Smart Spaces

Ahmed E. Khaled, Rousol Al Goboori

Computer Science Department, Northeastern Illinois University, Chicago, IL 60625, USA
{aekhaled, ralgoboori}@neiu.edu

ABSTRACT

Smart devices around us produce a considerable volume of data and interact in a wide range of scenarios that guide the evolution of the Internet of Things (IoT). IoT adds informative and interactive aspects to our living spaces, converting them into smart spaces. However, the development of applications is challenged by the fragmented nature due to the vast number of different IoT things, the format of reported information, communication standards, and the techniques used to design applications. This paper introduces IoT Hub as a Service (HaaS), a data-oriented framework to enable communication interoperability between the ecosystem's entities. The framework abstracts things' information, reported data items, and developers' applications into programmable objects referred to as Cards. Cards represent specific entities and interactions of focus with meta-data. The framework then indexes cards' meta-data to enable interoperability, data management, and application development. The framework allows users to create virtual smart spaces (VSS) to define cards' accessibility and visibility. Within VSS, users can identify accessible data items, things to communicate, and authorized applications. The framework, in this paper, defines four types of Cards to represent: participating IoT things, data items, VSS, and applications. The proposed framework enables the development of synchronous and asynchronous applications. The framework dynamically creates, updates, and links the cards throughout the life-cycle of the different entities. We present the details of the proposed framework and show how our framework is advantageous and applicable.

KEYWORDS

Internet of Things (IoT), Interoperability, Cards, Events, Applications, Service, Virtual Smart Spaces, Hub

1 INTRODUCTION

The Internet of Things (IoT), over the last decade, has attracted tremendous community and industry interest for its potential to convert our everyday living and working spaces into smart spaces [14][23][24]. Sensors, appliances, and devices in our smart spaces (e.g., homes, offices, universities, cities) are referred to -in the context of IoT and smart spaces- as *things*. In this paper, we use devices and things interchangeably. Empowered by the recent advances in low-power wireless technologies and embedded processing, the main research focus is shifted to include the different infrastructure options, sensing technologies, communication standards and protocols, in addition to

This paper is accepted at the International Workshop on Very Large Internet of Things (VLIoT 2022) in conjunction with the 2022 VLDB Conference in Sydney, Australia. The proceedings of VLIoT@VLDB 2022 are published in the Open Journal of Internet of Things (OJIOT) as special issue.

many other aspects of IoT [1][7][13]. Users and developers interact with IoT things directly or indirectly through platforms and edge devices. Such interactions include triggering the different offered services and functionalities by the IoT things, updating and managing the different IoT things, and analyzing data collected and reported by such things. Through such interactions (between users or platforms and IoT things) and interconnections (between different things), IoT things bring more informative and interactive dimensions to build intelligent environments and develop domain-related applications.

The huge range of IoT things with different processing capabilities, resources, and sensory technologies enables a wide range of interactions in many scenarios. With such nature of IoT devices and interactions, a broad set of application-layer communication protocols used by application developers, edge devices, software platforms, and IoT things to interact have been developed. The IoT communication protocols are designed to satisfy such dynamic and ad-hoc nature of smart spaces [2][4][6], where each protocol addresses a set of requirements and aspects (e.g., low-power operation, lightweight message header, confirmable interactions). However, the development of IoT applications and the enablement of data analysis and mining tasks are challenged by such a highly fragmented nature of IoT. They require considerable integration effort and human intervention when different communication protocols are utilized. To avoid such isolation and for friction-free communication between IoT things, platforms, and developers, the research topic of communication interoperability [4][16][19] has the potential to homogenize communications with a set of mechanisms and translation schemes.

Under the IoT umbrella, things report information according to their types, capabilities, functions, and offered services. Such information includes 1) identity information about the thing (e.g., name, vendor), 2) context information (e.g., location), 3) status information (e.g., on, sleep, requires maintenance), 4) collected data from the environment (e.g., sensory data). According to thing-to-thing, thing-to-platform, and platform-to-users communication channels and patterns, we point out three main types of interactions that include: 1) commands to services and functions offered by a thing; 2) synchronous query and data analytic related tasks on the currently available data items collected and reported by IoT things and users; and 3) asynchronous tasks by application developers (e.g., call-back functions, event-based scheduled actions) that address future events. As a regular data packet, an interaction is composed of a header with descriptive information about the communicating entities and a payload as the content of the message is structured and formatted according to the used communication protocol.

Proposed solutions in the literature [18][9] target the interoperability challenge by matching interactions by IoT things that use different communication protocols concerning the identifiers. Such solutions utilize centralized platforms or edge devices throughout the IoT things report information and communicate with other IoT things. An identifier of interaction can be a topic in Message Queuing Telemetry Transport (MQTT) protocol [17] or URL in Hypertext Transfer Protocol (HTTP-Rest) or Constrained Application Protocol (CoAP) protocols [26][22]. For example, if HTTP-things POST information on a URL named /Sensor/Data, MQTT-things subscribed to a topic called /Sensor/Data will receive the information announced by the first group. Other proposed solutions [19] address the communication interoperability challenge through specific translation schemas between the different communication protocols. Such solutions allow the translation of the entire interaction from one protocol to another and enable thing-to-thing communication. However, the proposed approaches and solutions restrict the interoperability through only the identifier of the interaction without exploiting other metadata and attributes encoded in the header and payload of each interaction. The proposed solutions also focus on the interactions between IoT things and not the interactions between IoT things and the different users and developers. Users and developers should interact with IoT things and utilize the collected knowledge and information to build queries, tasks, and event-based applications. Abstracting information and knowledge from IoT things and users about interactions and applications enables a broader opportunity for interoperability and interactions between the different entities of the ecosystem.

This paper introduces IoT Hub as a Service (HaaS), a framework that enables data-oriented interoperability and provides an environment for IoT application development. IoT things and users interact with the Hub using different communication protocols. The framework abstracts things' information and reported data items as well as developers' queries and tasks into protocol-independent programmable objects referred to as *Cards*. Each Card represents a specific entity or an interaction of focus with a set of meta-data and attributes. The framework manages and indexes cards' attributes then links the cards' attributes accordingly to enable interoperability and data management and provide an application development environment. The framework, in this paper, defines four types of Cards to represent: 1) participating IoT things, 2) Data-Items reported by things and users, 3) Virtual Smart Spaces

(VSS), and 4) Applications. Using the IoT Hub, developers design and build applications that include: 1) queries, filters, and data analysis tasks; and 2) event-based actions, alerts, and call-back functions. The framework utilizes the Hub's operating system to provide a run-time environment for the applications to run on the collected and reported information and interactions.

The framework allows users to create and configure VSS as virtual representations of physical smart spaces (e.g., smart home, office). Within a VSS, users can identify which IoT things can interact, which data items to be accessed, and authorized users to access such space and run applications. The framework dynamically creates, updates, and links these cards throughout the life-cycle of the different entities. Consider a user-defined VSS that logically groups a set of personal healthcare sensors and their generated data items as an example. Authorized users can utilize the defined VSS to build healthcare monitor applications to answer specific queries.

The contribution of the proposed framework for the IoT Hub can be summarized as follows: 1) The framework allows IoT things and users who use different communication languages to interact with the IoT Hub; 2) The framework presents data-oriented interoperability through abstracting knowledge and information in the various interactions and the various participating entities into programmable entities called Cards; and 3) Indexing and linking the different protocol-independent cards to provide data-oriented interoperability; and 4) providing an IoT application development environment and the design of VSS that define accessibility, visibility, and interaction between the different cards. The framework utilizes On-Cloud resources and platforms to enable accessibility by IP-communicating IoT things, users, and developers from anywhere and provide scalability to consider more communication protocols and dynamic processing capabilities. The Hub may also reside on an edge device (e.g., local server) in a local organization allowing IoT things with other communication technologies (e.g., Bluetooth) to interact with the Hub.

In this paper, we present the architecture of the IoT Hub, the details of the framework's different modules, and the structure of the different Cards and their life-cycle. The paper is organized as follows: Section 2 highlights related work compared to the proposed framework. Section 3 describes the proposed IoT Hub framework, the architecture, and a discussion on the different modules. Section 4 presents the various cards and then discusses the structure and the corresponding meta-data and attributes. Section 5 offers the life-cycle and the interactions between the cards, then presents the different design aspects and implementation plan. Finally, the paper is concluded with a discussion on future directions in section 6.

2 BACKGROUND AND RELATED WORK

This section provides a quick overview of the commonly used IoT application-layer communication protocols that we refer to and use in this paper. We then highlight proposed frameworks and approaches that target IoT interoperability in the literature.

2.1 Overview of Messaging Patterns

Communication between different entities (e.g., things, applications, platforms, users) in an IoT ecosystem follows one or more messaging patterns. A messaging pattern describes how such entities are connected and the sequence of messages required to establish and terminate communication channels for message exchange. This paper focuses on two major messaging patterns: the request/response pattern and the publish/subscribe pattern [26][12][2].

2.1.1 Request / Response Pattern

This pattern describes the client/server messaging model, where clients request resources offered by a server. In a dynamic and distributed peer-to-peer environment like IoT, an entity can be client and server simultaneously, depending on the role such entity plays at a particular point in time. If an on-cloud platform asks a sensor for a data item, the sensor plays a server role. If the same sensor requests information from the platform, the sensor plays a client role. In such a communication pattern, a resource (e.g., file, data record, status information) is identified and referred to by Uniform Resource Identifier (URI) in the client's request to the server. Under this pattern, Hypertext Transfer Protocol (HTTP) and Constrained Application Protocol (CoAP) are the widely used IoT communication protocols [26][22].

- HTTP is the widely used application protocol for distributed and collaborative ecosystems and is believed to be the foundation of data communication on the World Wide Web. HTTP implements the request/response paradigm model over TCP and provides different REST methods such as GET, POST, PUT, and DELETE for clients to access various resources hosted by servers [22].
- CoAP is designed for resource-constrained devices (e.g., battery-powered sensors) and dynamic ad-hoc networks like IoT. CoAP implements the request/response model over UDP for lightweight and reduced-size headers. CoAP compensates for UDP's unreliability through acknowledgment and

retransmission mechanisms. CoAP offers similar HTTP-REST methods (GET, POST, PUT, and DELETE) over confirmable and non-confirmable messages to access the various resources. In addition, CoAP supports multi-casting for things to communicate messages with several other entities [5][15][26].

2.1.2 Publish / Subscribe Pattern

In this messaging pattern, there are three roles: 1) publishers post information over topics to an intermediary message broker, 2) subscribers express interest in a particular topic(s) with the broker, and 3) message broker performs the filtering and routing of the messages from publishers to subscribers. A topic is a user-defined communication channel where messages can be sent (published) and received (subscribed to). Like URLs of request/response patterns, topics can be organized into a namespace with a hierarchical structure. Another essential task for the broker is to enable the publishers and subscribers to be loosely coupled - where they do not know of each other existence and do not need to work at the same time. Such a communication model enables highly scalable and flexible solutions, where clients (publishers and subscribers) only communicate over the topic of interest without knowing each other. The process of selecting messages for reception and processing is known as filtering, with two common forms. In topic-based filtering, the messages are published to publisher-defined topics, and subscribers receive messages published on such topics. In content-based filtering, the messages are delivered to a subscriber if the content of those messages matches subscriber-defined constraints. Under such messaging pattern, Message Queue Telemetry Transport (MQTT) and Advanced Message Queuing Protocol (AMQP) are the widely used IoT communication protocols [12][3].

- MQTT is a TCP-based event-driven paradigm that allows clients (e.g., IoT devices, users) to publish messages over topics and to subscribe to topics. An MQTT broker is the central communication point in dispatching and routing messages between publishers and subscribers mainly for synchronous communication. The broker usually resides on an on-cloud platform or an edge device (e.g., local server). Each MQTT subscriber has a permanently open TCP connection to the broker. If the connection is interrupted, the broker may buffer published messages to be sent when the subscriber is back online [17][12].
- AMQP follows the publish/subscribe model where brokers (also known as mediators) allow messages to be stored in queues for asynchronous communication. The receivers collect messages from queues when they have the capacity to process them. The broker distributes messages to different queues according to routing rules. Messages have routing keys, the queues have binding keys, and the broker learns which queue the message belongs to from the routing-binding relationship. A queue may have several binding keys, and multiple queues can also share one binding key. The broker replicates messages and sends them to multiple recipients accordingly. Another way is through the headers exchange acts with the message's header, instead of using a routing key. A header contains values that match with the binding. The argument with the name x-match determines if all values must match (value: all) or only one must match the binding (value: any). While the former corresponds to the direct exchange, the latter can produce the same effect as a topic exchange [3].

2.2 State of the Art

C. Lee et al. [4] proposed a hybrid IoT communication framework based on a software-defined network (SDN) that intercepts all packets from CoAP and MQTT, and vice versa. The proposed framework defines URL rules to specify the topic and differentiate the homogenous (e.g., from CoAP client to CoAP client) from heterogeneous packets (e.g., from MQTT client to CoAP client). If packets belong to the same protocols, they operate as the original communication scenarios, and the SDN just ignores this traffic. When the traffic of heterogeneous protocols is intercepted, the SDN switch delivers these packets to the SDN controller. The SDN controller is responsible for redirecting the packets to the cross proxy for protocol translation.

P. Bellavista et al. [18] proposed a gateway-oriented architecture where gateways jointly exploit MQTT and CoAP to achieve highly scalable IoT device management through dynamic hierarchical tree organizations. The proposed gateway extends the Kura Eclipse framework, which is based on the interworking of MQTT that is already integrated into the Kura framework, with CoAP coordination functionality as an added protocol. The extended Kura framework offers improved scalability and reduced latency for communication/coordination among wide-scale sets of geographically distributed IoT devices interworking via gateways for efficient resource lookup. MQTT plays a central role and is strongly exploited by the Kura framework, but exhibits nonnegligible limitations in terms of scalability, e.g., inefficient usage of TCP connections toward the broker when growing the number of IoT devices in a gateway locality.

Ponte [9] is an Eclipse IoT project that offers uniform open APIs to let developers create applications supporting CoAP, MQTT, and HTTP REST communication protocols through an independent module for each protocol. The Ponte gateway can reside in a server or edge where clients with different communication protocols can communicate. Ponte provides a centralized solution for interoperability, where the smart space resources reside in the cloud and can be accessed from different clients. Data collected, from the three modules, is stored in a database; therefore, no matter which protocol clients utilize for communication, they can access the same resources.

A. Khaled et al. [2] introduced a framework named Atlas IoT communication framework. The framework tools up IoT things with protocol translators that could be either hosted on the IoT things or in the cloud. The proposed framework is designed to facilitate interoperability among heterogeneously communicating things without taxing the performance of things that are homogenously communicating. The framework itself utilizes the topic concept and uses a meta-topic hierarchy to map out and guide the translations. The translators target seamless thing-to-thing homogenous and heterogeneous communication and do not focus on engaging the user in developing queries, applications, or defining virtual smart spaces.

P. Desai et al. [19] proposed a gateway and semantic web-enabled IoT that provide translation between messaging protocols such as XMPP, CoAP, and MQTT using a multiprotocol proxy with a separate interface for each protocol. The proposed gateway, located between the physical-level sensors and the cloud-based services, has a centralized topic router that maps information from the different communication protocols. EMQTT [10] is a massively scalable MQTT broker for IoT and mobile applications licensed under Apache. An EMQTT server implements the MQTT protocol and supports a set of plugins that allows other communication protocols to coexist in parallel (e.g. MQTT-SN, CoAP).

The proposed approaches target the interoperability challenge concerning the identifiers (e.g., a topic in MQTT, a URL in HTTP) to allow thing-to-thing interaction. Other solutions address the interoperability challenge through translation schemas between the different communication protocols. Such schemas allow the translation of the interaction from one protocol to another. However, the proposed approaches and solutions restrict the interoperability through only the interaction's identifier without exploiting other metadata and attributes encoded in the header and payload of each interaction. The proposed solutions focus on the interactions done by IoT things, not the interactions by the different users and developers to design virtual smart spaces and build applications. Users and developers interact with IoT things and utilize the collected knowledge to build queries, tasks, and applications. Abstracting information about participating entities and their different interactions enables broader interoperability and two-way interactions between the different ecosystem entities.

3 IoT Hub

In this section, we present a brief overview of the proposed IoT Hub in terms of the different entities, cards, and architecture of the proposed framework.

3.1 Main Entities and Cards

Before discussing the architecture and the different components and modules, we first need to define the main entities that communicate and interact throughout the IoT Hub. The main entities can be summarized into:

- *IoT things* communicate with the Hub to share identity and status information and also to report data items (e.g., sensory data collected from the surrounding environment). IoT things employ different communication protocols (e.g., HTTP, MQTT) to communicate and interact with the Hub and also accept interactions and commands from the Hub.
- *Users* communicate with the Hub to visualize information about connected things and the reported data items, then run synchronous applications. Users may also create data items and communicate such items with the Hub. Synchronous applications, as discussed later, include queries and data analysis tasks, and user-defined applications. Users also create and configure virtual smart spaces (VSS) as a logical representation of physical smart spaces and a way to define accessibility and interactions. Within a VSS, users can identify which IoT things can interact and communicate, which data items to be accessed, the authorized users, and which applications to run.
- *Application Developers* communicate with the Hub to build another category of applications, known as asynchronous applications. This category addresses tasks requiring monitoring future information and knowledge from IoT things and data items within user-defined VSS. Asynchronous applications include customized alerts, call-back functions, and event-based scheduled actions. The Hub utilizes simplex and complex processing techniques for event handling, real-time data processing, and the runtime environment for such applications.

The different interactions by the ecosystem's entities include 1) identity information reported by a thing; 2) commands and instructions to services offered by a thing; 3) sensory data streams, periodic status information, and events reported by IoT things; 4) synchronous query (e.g., filter, search) and data analytic related tasks (e.g., accumulated content, average sensory data within a time window) on the currently available things' information and data items; and 5) asynchronous tasks (e.g., alerts, call-back functions, event-based scheduled actions). Any of these interactions, as a regular data packet, is composed of 1) *header* - meta-data and descriptive information about the communicating parties (e.g., IoT thing that created the data item, a user who developed a synchronous query); and 2) *payload* - the actual content of the interaction that can be structured, semi-structured, or unstructured with different types (e.g., text file, plain message, XML page, JPEG image) and formats based on the communication protocol and standard used by the entity.

Utilizing the meta-data and descriptive information embedded in the different protocol-dependent interactions' header and payload enables data-oriented interoperability. The framework of the IoT Hub abstracts the various interactions and the ecosystem's entities into protocol-independent programmable objects referred to in this paper as *Cards*. Each Card represents a specific entity or an interaction of focus with a set of attributes.

The framework, in this paper, defines four types of Cards to represent the ecosystem entities and interactions of focus:

- *Data-Item Card*: describes and captures data and knowledge (e.g., sensory data) collected and reported by IoT things and users.
- *IoT thing Card*: describes identity and activity information of the participating IoT thing involved in creating, modifying, requesting, and accessing the different data items.
- *Virtual Space Card*: describes virtual smart space (VSS) as a logical group of IoT things (e.g., belong to a particular user, belong to a specific organization, co-exist in the same smart space like home) to restrict communication visibility, function trigger, and data item accessibility to authorized users, developers, and things.
- *Application Card*: describes an application (synchronous or asynchronous) designed and developed by developers within accessible VSS to query and analyze available data items, create event-based tasks, develop user-defined application and configurable alerts, and trigger services offered by things.

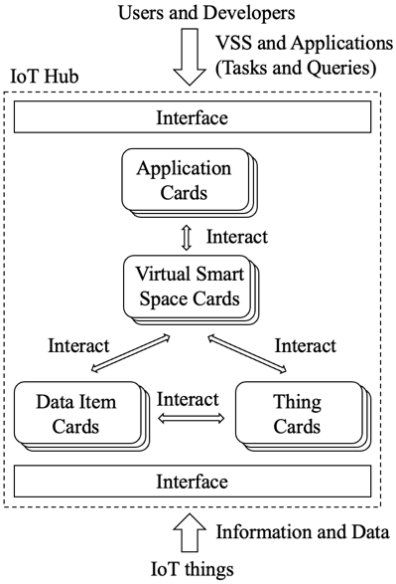

Figure 1: The different cards, along with their interactions and connections

The framework manages and indexes the different cards then links their descriptive attributes to enable interoperability and data management. The framework then allows the development of a wider set of applications as well as provides a runtime environment for IoT application development and execution.

3.2 Architecture

As illustrated in Figure 1, the different cards are dynamically created and updated by the framework throughout the life-cycle of the various entities and interactions by IoT things, users, and developers. The framework links and indexes the descriptive attributes in the different protocol-independent cards to 1) reflect the current status of the ecosystem and the connected entities; 2) enable data-oriented interoperability among heterogeneously communicating entities; 3) allow efficient data management as well as the development of IoT applications. To achieve these goals, in this subsection, we present an overview architecture for the proposed framework then show the details of the different modules. The framework of the IoT Hub, as illustrated in Figure 2, is divided into a set of main modules as follows:

The *Communication Interface* module hosts the different communication channels that IoT things, users, and developers can use to communicate and interact with the IoT Hub. This module is composed of individual and independent sub-interfaces, an interface for each supported communication protocol (e.g., MQTT, HTTP, CoAP). IoT things, users, and

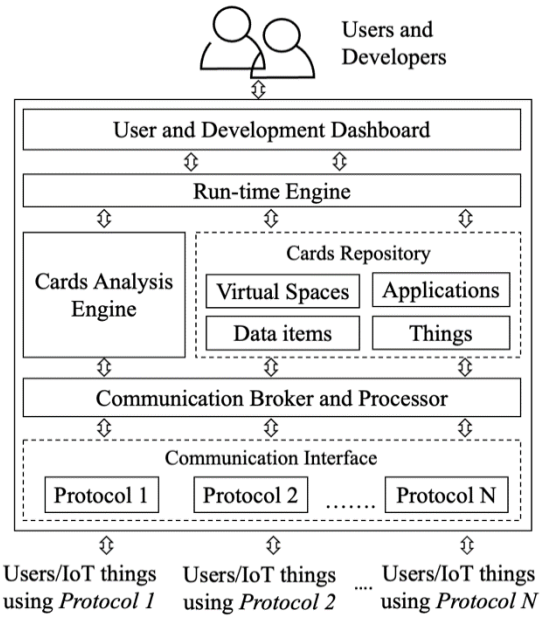

Figure 2: The overall architecture of the IoT Hub

developers -with respect to the communication language and protocol used- address these interfaces to interact with the Hub. For example, HTTP-communicating IoT things POST/PUT data items with the Hub through the HTTP protocol interface. On the other side, MQTT-communicating users connect with the IoT Hub through the MQTT protocol interface, publish queries and subscribe for results and information about the communicated data items.

The *Communication Broker and Processor* module accepts protocol-dependent IoT things' information and reported data items as well as users' queries and developers' applications received through the different communication interfaces. According to the specifications of the communication protocol, the broker parses metadata and attributes embedded in the header and payload of the received interaction. The processor then creates new protocol-independent cards or updates cards in the *Cards Repository*.

The *Cards Repository* is a schema-less lightweight database to store, manage, and access the different cards (things, data items, virtual spaces, and applications) created and updated by other modules. The choice of such a database is for the support it offers to structured, semi-structured, and unstructured data. The *User and Development Dashboard* module is the graphical interface of the Hub through which users visualize available data items and things and then design and configure VSS. The interface also allows the developers to design and build synchronous queries, tasks, and asynchronous event-based applications that utilize available cards, as discussed in details next section.

The *Run-time Engine* performs two main functions. The first function is to accept users' requests to create and manage VSS, which leads to creating and managing virtual space cards in the repository. The second function is to accept the different designed and developed applications. This module checks the validity of an application according to a set of semantic rules then creates and manages application cards in the repository. With the help of the *Cards Analysis Engine* module and the host operating system of the Hub, the Run-time Engine executes the different applications using independent threads in a run-time environment and returns the results.

The *Cards Analysis Engine* hosts libraries for the different data analysis, pattern detection, and real-time event processing techniques as tools for application developers to design and develop IoT applications. The engine processes the new or updated cards; indexes then links them with other available cards in the repository throughout the different attributes and meta-information as discussed in the following subsection. This module also runs periodically to infer and understand certain attributes from the cards (e.g., the domain of specific data items, the nature of the content).

The framework utilizes On-Cloud resources and platforms to 1) enable accessibility by IP-communicating IoT things, users, and developers from anywhere; 2) provide horizontal scalability allowing more communication protocols and interfaces to be integrated; and 3) provide vertical scalability by extra storage and dynamic processing capabilities. The Hub may also reside near things in a local organization on an edge device (e.g., a local server). In this case, IoT things with other communication technologies (e.g., Zigbee, Bluetooth) can communicate and interact with the Hub.

4 CARDS: STRUCTURE AND ATTRIBUTES

The proposed framework abstracts information about participating entities along with their different interactions with the Hub. There is a wide range of interactions from information and data items communicated by IoT things and users to queries and applications by users and developers. Such abstracted knowledge is transformed into a set of programmable protocol-independent objects referred to as *Cards*. The focus of this paper is on four different types of cards that represent entities and specific interactions of focus: Data-Item cards, Thing cards, Virtual-Space cards, and Application cards.

This section presents the different cards then discusses the structure and the attributes that shape every card. The attributes listed in the different cards

represent an initial set of attributes defined by the framework. The Hub allows more framework-defined and user-defined attributes to be added when required. The ability to dynamically add or modify attributes addresses the dynamic nature of the ecosystem's interactions and entities. The user-defined attributes also enable ad-hoc scenarios or use-cases to be considered. The cards are schema-less; some attributes are mandatory, while others are considered optional with respect to the requirements.

4.1 Data Item Card

The data item type of cards represents sensory information collected by IoT things from the surrounding environment and other data streams generated and communicated by things. Users also can create and communicate data items with the Hub. These interactions are delivered to the Hub using different communication protocols and received by the corresponding communication interface. The communication broker of the Hub parses these interactions, analyzes the meta-data in both the header and the payload of the interaction, and then converts them into new or updated data-item cards in the repository. If the meta-data refers to an item with a new identifier, the broker translates the interaction into a new data item card; otherwise -if the identifier already exited- the broker updates an already existing data item card.

As illustrated in Table 1, a data item card -following the key-value format- is composed of a set of framework-defined attributes that are categorized into 1) *Identification info*, set of attributes that uniquely identify the data-item card in the repository; 2) *Users and Things info*, set of attributes representing users and IoT things that modify, request, and subscribe to the data item; 3) *Statistical info*, set of attributes that track access and modification requests to the data item; and 4) *Content info*, set of attributes that capture the format and domain of the content as well as the actual payload of the data item. The values of some attributes (e.g., name, creator, created date) are created upon the reception of the interaction by the Hub. The values of other attributes are added and updated while creating or modifying other types of cards (e.g., list of things subscribed to the data item, the access frequency to the data item). Some attributes receive their values directly through users' queries/inputs (e.g., type and domain of the content) or are inferred by the *Cards Analysis Engine* module. The Analysis Engine also applies different analysis and mining techniques to learn and infer the value of some attributes (e.g., the domain of the content).

Table 1: Structure of a Data Item Card

Attribute	Description
Identification Info	
Unique ID	URL or Topic or Short ID assigned by the framework
Name	Short name describing the item
Creator	ID of the IoT thing or user first created the item (e.g., Publish request, Post request, User's Input)
Virtual Spaces	List of the smart spaces where this item can be accessed. By default, this item is public and can be accessed in any VSS.
Time to Live	Date/time when this item expires and/or be deleted (disabled by default)
Users and Things info	
Modifier	List of users and IoT things update/modify the item (e.g., Publish requests, Post/PUT requests, Users' inputs)
Requester	List of users and IoT things request the item (e.g., GET requests, applications, queries)
Subscriber	List of users and IoT things subscribe for updates to the item (e.g., Subscribe requests, Callback functions, even-based applications)
Statistical info	
Created	Date/time this card is created
Modified	Date/time this card is last modified
Access Frequency	Number of times this card is accessed by requesters and subscribers
Modify Frequency	Number of times this card is modified
Content info	
Format	The format of the data item (e.g., text file, image JPEG, XML, JSON)
Type	The nature of the data item (e.g., sensory data, reported event, configuration, information)
Description	The meaning of the content (e.g., temperature value in Celsius, humidity value, noise level)
Domain	List of possible applications this item can be used in (e.g., healthcare, traffic, smart homes)
Payload	the latest version of the data item's body
Accumulated	The concatenated version of payload's updates

4.2 Thing Card

The thing type of cards represents information about the participating IoT things (e.g., sensors, devices, appliances) that communicate and interact with the Hub. Accordingly, the communication broker of the framework creates new thing cards or updates already

Table 2: Structure of a Thing Card

Attribute	Description
Identification Info	
Unique ID	Defined by the thing or a short ID assigned by the framework
Name	Short name describing the participating thing
Virtual Spaces	List of the smart spaces within such IoT thing can be accessed. By default, this item is public and can be viewed and accessed in any VSS.
Thing Info	
Type	The type of the thing (e.g., temperature sensor, motor).
Vendor	The name/identity of the Manufacturer
Model	The specific model of the thing
Description	Short description on the thing's capabilities
Module	The communication module used by the thing (e.g., Wi-Fi, Bluetooth, Zigbee)
Address	The address of the thing (e.g., IP for Internet enabled thing)
Location	The GPS coordinates (disabled by default)
Statistical Info	
Status	Date/time for the latest data-item and updates such thing engaged in, reflecting the thing's status (e.g., active, sleep)
Access Frequency	Number of times this card is accessed (e.g., GET request, Developers' tasks, Users' queries/inputs)
Data Item info	
Created	List of data items created by this thing
Modified	List of data items modified by this thing
Requested	List of data items requested by this thing
Subscribed	List of data items subscribed to by this thing

existing thing cards. IoT things create, modify -through POST, PUT, PUBLISH, DELETE methods- or request -through GET, SUBSCRIBE methods- different data item cards.

As illustrated in Table 2, the thing card -following the key-value format- is composed of a set of framework-defined attributes that are categorized into 1) *Identification Info*, a set of attributes that uniquely identify an IoT thing card in the repository; 2) *Thing Info*, set of attributes that describes the IoT thing by type, vendor, communication module, and location; 3) *Statistical Info*, set of attributes that track the activity of the thing; and 4) *Data-item Info*, set of links between the IoT thing and the different data-items such thing created, modified, subscribed to, and requested. Similar to Data-item cards, the values of some attributes (e.g., Name, Address) are created upon the connection between the IoT thing and the Hub. The values of other attributes are added and updated while creating or modifying other types of cards (e.g., the access frequency to the thing card). Other attributes receive their values directly through users' queries/inputs (e.g., description, module).

Table 3: Structure of a Virtual Space Card

Attribute	Description
Identification Info	
Unique ID	Created by the user or assigned by the framework
Creator	Information about who designed the VSS
Description	short description of the VSS
Type	public (accessible by any user) or private space (accessible only by specific set of users)
Space info	
Created	Date/time when the VSS is created
Users	List of users and developers allowed to run applications/queries within the VSS
Things	List of IoT things accessible within the VSS
Data Items	List of data items accessible within the VSS

4.3 Virtual Space Card

The virtual space type of cards represents the logical grouping of IoT things and data items. For example, users create VSS for IoT things in their homes or offices and include the data items generated by such things. The authorized users can visualize and interact with IoT things and data items. Also, virtual spaces can define the working areas and boundaries for applications. For IoT applications (e.g., query, task), as an example, users create virtual space and list specific IoT things and sensors needed for such applications.

Through the User and Development Dashboard of the framework, users create VSS to define the communication visibility of IoT things and data item accessibility to authorized members (users, developers, and things). Applications (e.g., healthcare, traffic, homes) can utilize the virtual space to collect context-aware information from the target environment, secure access windows over specific IoT things and data items, and build virtual space-oriented applications. As illustrated in Table 3, the virtual space card -following the key-value format- is composed of a set of framework-defined attributes that are categorized into 1) *Identification Info*, set of attributes that uniquely identify and describes the VSS; and 2) *Space Info*, set of links between the created virtual space and the different accessible data-items and IoT things within such space as well as authorized users and developers.

4.4 Application Card

IoT things engage in a wide range of interactions with other things, users, and developers throughout the IoT Hub. As mentioned before, IoT things report different types of information according to things' type, capabilities, and offered services. Such information includes but is not limited to 1) identity information about the thing (e.g., name, vendor), 2) context information (e.g., location), and 3) collected data from the surrounding environment (e.g., sensory data). The focus of the Hub is to abstract and index the different interactions reported by heterogeneous IoT things and then link the attributes to enable designing and building applications. The Hub creates corresponding programmable objects -the thing and data item cards- and allows users and developers to execute applications (tasks and queries). The Run-time Engine of the Hub accepts designed and developed applications then checks the applications' validity according to a set of semantic rules, as discussed in this sub-section. The engine then creates or modifies application cards in the repository accordingly. An application card represents the different queries and tasks a developer can design, develop, and deploy given the other types of cards (data items, virtual smart spaces, and IoT things).

In this paper, as illustrated in Figure 3, we categorize the applications supported by the proposed framework into two main categories: 1) Synchronous applications and 2) Asynchronous applications.

4.4.1 Synchronous Applications

Each of the data items, things, and virtual spaces cards has meta-data and accumulated information updated and linked to other cards. Synchronous applications refer to tasks and queries that can be answered directly, given the currently available cards in the repository. A query, as illustrated in Table 4, is defined using three parameters:

- *Scope*: specifies the search space and domain for the work of the query, in terms of VSS(s) and specific cards.
- *Return*: lists a set of attributes to be accessed from the cards within the defined scope.
- *Conditions*: defines an optional set of conditions to select specific cards from the defined scope. There are no conditions by default.

The run-time engine of the Hub converts such database-independent queries into database-dependent ones with respect to the type of database used to implement the repository of the cards.

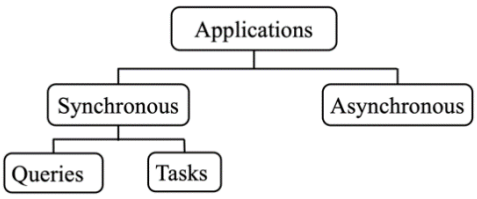

Figure 3: The two main categories of architectures supported by the IoT Hub

Table 4: Structure of a Synchronous Application Card

Attribute	Description
Identification Info	
Unique ID	Created by the user or assigned by the framework
Creator	Information about the developer
Created	Date/time when the application is created
Type	The type of the application can be: Synchronous Query, Synchronous tasks, or Asynchronous task (outside the focus of this paper)
Query Info	
Scope	Domain of the query
Return	The required attributes from the defined scope
Conditions	Optional list of conditions to select specific cards
Task Info	
Scope	Domain of the task
Method	The body of the task
Input	The required inputs
Results Info	
Result	The result of running the query/task
Frequency	Number of times the query/task is executed

On the other hand, a task -as illustrated in Table 4- is defined using three parameters:

- *Scope*: specifies the search space and domain for the work of the task, in terms of VSS(s) and specific cards.
- *Method*: accepts user-defined functions or one of the data-analysis tools available to the developers by the Card Analysis Engine.
- *Input*: lists a set of attributes from the cards within the defined scope as input(s) for the Method's execution.

The Cards Analysis Engine provides a set of tools to the run-time environment to execute the required task. Under the method attribute, developers can design and develop their own methods to work using the defined

input within the defined scope. As an example, the developer can use or develop an aggregate function on a certain data item over a specific time window. Examples of each parameter to build queries and tasks are listed in Table 5.

4.4.2 Asynchronous Applications

Asynchronous applications refer to queries and tasks that cannot be addressed given the current repository of cards. Such applications wait for certain events that may occur in the future to run scheduled pre-defined actions. The Run-time Engine, utilizing the hosting operating system of the Hub, converts such applications into call-back functions on separate threads of execution and utilizes real-time event-processing techniques by the Card Analysis Engine. An example of event-based tasks is the design of customized alerts that are triggered on receiving certain data items within certain VSS. Another example of an asynchronous application is to trigger a data-analysis task when specific IoT things post sensory data that is not within a user-defined range. The syntax of asynchronous applications and the used event-processing techniques, are outside the scope of this paper.

Table 5: Examples of Synchronous Queries and Tasks

Query Structure	Description
Examples on Scope's value	
/ {DT or TH}: * /	All public data-item or thing cards
/ {DT or TH}: {ID_1, ID_2, ... ID_N} /	Certain data-item or thing card, given the card(s) ID(s)
/ {VS}: *ID* / {DT or TH}: * /	All data item or thing cards within a specific VSS
/ {VS}: *ID* / {DT or TH}: {ID_1, ID_2, ... ID_N} /	Specific data-item or thing card(s), within certain VSS
Examples of Input's and Return's value	
{AR_1, AR_2, ... AR_N}	Specific attribute(s) with respect to the defined scope
Examples on Condition's value	
{$AR_1\ \theta\ VL_1$, $AR_2\ \theta\ VL_2$, ... $AR_N\ \theta\ VL_N$}	returns data-item/thing cards satisfying such conditions
Null	No conditions

DT: Data Item Card; VS: Virtual Space Card
TH: Thing Card; AR: an attribute, by name, in a card
VL: value for selection
θ:{ $\neq$, $\geq$, $\leq$, $<$, $>$, $=$ }

5 CARDS LIFE-CYCLE AND IMPLEMENTATION

Understanding the different interactions by IoT things and users is of great importance. Such knowledge enhances the ability to understand and monitor smart spaces and to design and develop a wide range of context-aware and smart space-related applications. Our ongoing efforts target designing and developing the IoT Hub, a central data-oriented environment as an on-cloud service that 1) accepts interactions and data items reported by heterogeneous IoT things and users; 2) allows users to define VSS and visualize available IoT things and data items; and 3) enables different queries and tasks to be designed and run by users and application developers. As detailed in section 4, the current version of the framework focuses on participating IoT things, data items, VSS, and applications as the main entities and interactions of interest in the ecosystem. The Hub accordingly abstracts the interactions into protocol-independent cards, where the meta-data and attributes of each card offer efficient data representation and management as well as a run-time environment for applications and data analysis methods.

The architecture of the IoT Hub, as discussed in section 3, has two interfaces to accept interactions: the User and Development Dashboard and the Communication Interface. The dashboard for users and developers is a graphical interface that allows defining VSS and Data Items as well as developing applications. The Communication Interface allows IoT things and users to use communication protocols (e.g., MQTT, CoAP, HTTP) to interact with the Hub. IoT things and users utilize one of the communication methods (e.g., HTTP GET request, MQTT Subscribe request, CoAP POST request) to access or update either a full card from the repository or a specific attribute from a specific card. Table 6 below shows the format for a request along with few examples for interactions by IoT things and users to communicate with the Hub through the communication interface. The broker analyses and handles the different interactions received from the communication interface and accesses the Cards repository accordingly.

The IoT Hub achieves interoperability through abstracting the interactions into cards then linking the different attributes and meta-information. Such data-oriented interoperability enables communication channels between entities speaking the same or different communication protocols. Figure 4 below summarizes the logical connections and relations between the different cards along with the interactions through the different interfaces provided by the architecture.

Table 6: Format and Examples on Headers for Interactions using Communication protocols

The format of Interaction's Header:

Method /{DT/TH/VS/AP:} Card's ID / {AR: Attribute to access}

Example	Description
GET /{DT:}Sensor/	HTTP GET request for the full Data Item Card with ID /Sensor
GET /{TH:}Actuator/{AR: Name}	HTTP GET request for the *Name* attribute of Thing Card with ID /Actuator
PUBLISH /{DT:}Home14/{AR: Creator}	MQTT Publish request to set the *Creator* attribute of Data Item card with ID /Home14, the body of the request contains the value

DT: Data Item Card; VS: Virtual Space Card; AP: Application Card
TH: Thing Card; AR: an attribute, by name, in a card

5.1 Use Case: Personal Healthcare Monitor

Consider a user-defined virtual smart space that logically groups different personal healthcare sensors and their data items to build a healthcare monitor and to answer related queries by authorized users. As illustrated in Figure 5 below, user A creates a virtual space card (VS_1).

Within *VS1*, IoT thing (TH_1) is a heart-rate sensor that collects and periodically updates the heart rate information of user *A* as a data item (DT_1). IoT thing (TH_2) is a blood pressure sensor that collects and periodically updates the blood pressure information of user *A* as a data item (DT_2). User *A* also creates a data item (DT_3) that represents the contact information of hospital *B* in case of an emergency. Hospital *B* is listed as an authorized user for VS_1 to access the different data items and to build and run applications. Hospital *B* builds a query (Q_1) to access and display the *Accumulated Value* attributes of both DT_1 and DT_2. User *A* uses the 3 data items and designs an event-based application (Q_2). Q_2 is an event-based application that sends an alert to the contact information provided in DT_3 if the current value of both DT_1 and DT_2 move beyond a pre-defined limit.

5.2 Implementation Plan and Future Directions

The current version of the IoT Hub [21] utilizes the Java Spring Boot [25] to build the framework, MongoDB for the repository, and thymeleaf HTML and JavaScript for the User and Development Interface. The communication interface will use the HTTP-REST, Californium-Eclipse CoAP library [8], and Paho-Eclipse MQTT library [20]. The framework targets Amazon Web Services to host the IoT Hub on

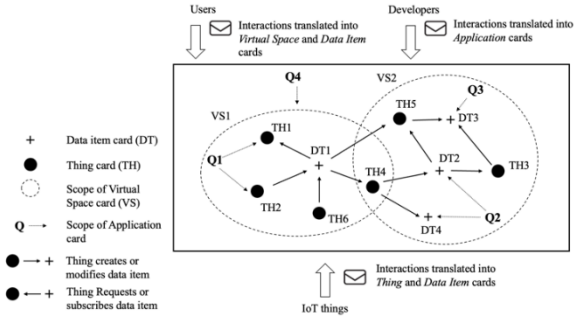

Figure 4: Logical connections between different cards

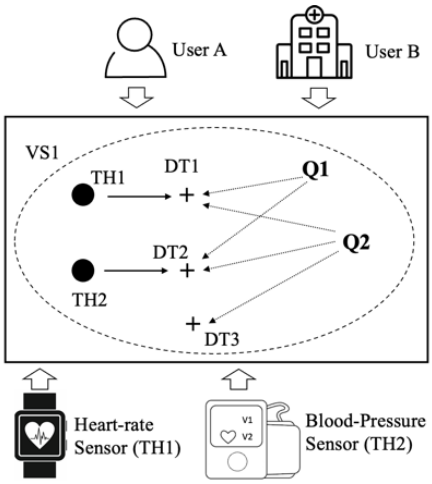

Figure 5: Personal Healthcare Monitor VSS and Applications

Amazon's cloud platform allowing IP-enabled IoT things to communicate and interact with the Hub. The current focus of the proposed IoT Hub is on the four types of cards mentioned in this paper, while more types are considered in future work. The current version of the Hub [21] enables users to design VSS and synchronous applications through the Dashboard. The current version of the Hub does not consider the required security solutions to provide authorized access for cards within VSS.

6 CONCLUSION

This paper introduces the IoT Hub as an on-cloud service providing a data-oriented framework for communication interoperability and the development of IoT applications. The proposed framework abstracts different interactions and information about participating entities into programmable protocol-independent objects referred to as Cards. The framework indexes and links meta-data and attributes in the cards to enable interoperability and data

management and provide an application development environment. The framework allows users to create virtual smart spaces to define cards' accessibility and visibility. Within virtual smart spaces, users can identify things to communicate, access data items, and run specific applications. The framework defines four types of Cards to represent participating IoT things, reported data items by things and users, virtual smart spaces defined by users, and applications designed by developers. The proposed framework enables the development of different applications, including synchronous queries and asynchronous tasks. The framework dynamically creates, updates, and links the cards together throughout the life-cycle of the different entities. We present the details of the proposed framework and the structure of the different cards.

REFERENCES

[1] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi, "Toward better horizontal integration among IoT services," IEEE Communications Magazine, vol. 53, no. 9, pp. 72-79, 2015.

[2] A. Khaled and S. Helal, "Interoperable communication framework for bridging RESTful and topic-based communication in IoT," Future Generation Computer Systems, 92, pp. 628-643, 2019.

[3] Advanced Message Queuing Protocol (AMQP) Protocol Specifications, Version 0-9-1, 2008, rabbitmq.com/resources/specs/amqp0-9-1.pdf

[4] C. H. Lee, Y. W. Chang, C. C. Chuang, and Y. H. Lai, "Interoperability enhancement for Internet of Things protocols based on software-defined network," In 2016 IEEE 5th Global Conference on Consumer Electronics, pp. 1-2, 2016.

[5] CoAP, RFC 7252 Constrained Application protocol, 2016, http://coap.technology/

[6] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," Ad hoc networks, vol. 10, no. 7, pp. 1497-1516, 2012.

[7] D. Namiot and M. Sneps-Sneppe, "On internet of things programming models," In International Conference on Distributed Computer and Communication Networks, Spring, Cham, pp. 13-24, 2016.

[8] Eclipse Californium (Cf) implementation of RFC7252 CoAP, 2021, https://github.com/eclipse/californium

[9] Eclipse Ponte by Eclipse Foundation, 2021, http://www.eclipse.org/ponte/

[10] EMQ - Massive scalable MQTT broker for IoT and Mobile Applications, 2017, http://emqtt.io/

[11] H. Derhamy, J. Eliasson, J. Delsing, P. Puñal Pereira, and P. Varga, "Translation error handling for multi-protocol SOA systems," In 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), pp. 1-8, 2015.

[12] IBM MQTT v3.1 protocol specification by IBM Eurotech, 2021, public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

[13] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," Future generation computer systems, vol. 29, no. 7, pp. 1645-1660, 2013.

[14] L. Babun, K. Denney, Z. B. Celik, P. McDaniel, and A. S. Uluagac, "A survey on IoT platforms: Communication, security, and privacy perspectives," Computer Networks, 192, pp. 108040, 2021.

[15] M. Koster, A. Keranen, and J. Jimenez, "Publish-Subscribe Broker for the Constrainted Application Protocol (CoAP)", proposed standard to the Internet Engineering Task Force (IETF), February 2018, https://tools.ietf.org/id/draft-ietf-core-coap-pubsub-03.html

[16] M. Noura, M. Atiquzzaman, and M. Gaedke, "Interoperability in internet of things: Taxonomies and open challenges," Mobile networks and applications, vol. 24, no. 3, pp. 796-809, 2019.

[17] MQ Telemetry Transport connectivity protocol, http://mqtt.org/

[18] P. Bellavista and A. Zanni, "Towards better scalability for IoT-cloud interactions via combined exploitation of MQTT and CoAP," In 2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), pp. 1-6, 2016.

[19] P. Desai, A. Sheth, and P. Anantharam, "Semantic gateway as a service architecture for iot interoperability," In 2015 IEEE International Conference on Mobile Services, pp. 313-319, 2015.

[20] Paho Eclipse by Eclipse Foundation, 2019, https://www.eclipse.org/paho/

[21] R. Al Goboori and A. Khaled, Initial IoT Hub Implementation, March 2022, https://github.com/RousolAlGoboori/iot-hub

[22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Less, "Hypertext Transfer Protocol HTTP/v1.1" Draft Standard to the Internet Engineering Task Force (IETF), June 1999, https://tools.ietf.org/html/rfc2616

[23] R. Want and S. Dustdar, "Activating the Internet of Things [Guest editors' introduction]," Computer, vol. 48, no. 9, pp. 16-20, 2015.

[24] S. Helal, "Programming pervasive spaces," IEEE Pervasive Computing, vol. 4, no. 1, pp. 84-87, 2005.

[25] Spring Boot 2.7.0, 2022, https://spring.io/projects/spring-boot

[26] Z. Shelby, K. Hartke, C. Bormann, "The Constrainted Application Protocol (CoAP)", proposed standard to the Internet Engineering Task Force (IETF), June 2014, https://tools.ietf.org/html/rfc7252

AUTHOR BIOGRAPHIES

Ahmed Ezzeldin Khaled received his Ph.D. degree in Computer Sciences in 2018 from the University of Florida, Gainesville, FL, USA. He is currently assistant professor at the Department of Computer Science, Northeastern Illinois University, Chicago, IL, USA. He received his B.Sc. and M.Sc. degrees in computer engineering from Cairo University, Egypt in 2011 and 2013, respectively. His current research interests include the Internet of Things, Distributed Systems, Cloud Computing, and Healthcare Systems.

Rousol Al Goboori received her B.S. degree in Electronics and Communications Engineering from Al-Nahrain University, Baghdad, Iraq, in 2013. and she is a graduate student at Northeastern Illinois University, Chicago, IL, USA, in Computer Science. She is currently a Junior Software Developer for ATC Transportation LLC, Pleasant Prairie, WI , USA.