sustainability

MDPI

Article

A DIY Approach for the Design of Mission-Planning Architecture Using Autonomous Task–Object Mapping and the Deployment Model in Mission-Critical IoT Systems

Shabir Ahmad, Faisal Mehmood and Do-Hyeun Kim *

Department of Computer Engineering, Jeju National University, Jeju 63243, Korea
* Correspondence: kimdh@jejunu.ac.kr

Received: 13 May 2019; Accepted: 28 June 2019; Published: 2 July 2019

Abstract: Recently, the World Economic Forum (WEF) highlighted mission-critical Internet of Things (MC-IoT) applications as one of the six enablers of sustainable development of smart cities. MC-IoT refers to systems which exacerbate properties like availability, reliability, safety, and security in an application environment of heterogeneously connected physical things and virtual things whose failure could lead to severe consequences such as life loss. The sole characteristic of the mission-critical system is its compliance with real-time behavior. As a result of the critical nature of these systems, it is essential to design the system with sufficient clarity so that none of the requirements is misinterpreted. For this, the involvement of non-technical stakeholders and policymakers is crucial. Previous studies on mission-critical structures mainly focus on the communication overheads, and overlook the design and planning of them. Therefore, in this paper, we present an architecture which enables mission planning on a do-it-yourself plane. We present a task–object mapping and deployment model where different tasks are mapped onto virtual objects and deployed on physical hardware in a task–object pair. The system uses semantic knowledge for autonomous task mapping and suggestions to further aid the orchestration of the process. The tasks are autonomously mapped onto the devices based on the correlation index; this is computed based on the attribute similarities, thus making the system flexible. The performance of the proposed architecture is evaluated with different key performance indicators under different load conditions and the response time is found to be under a few seconds even at peak load conditions.

Keywords: Internet of Things; Mission-Critical Systems; smart space; Do-It-Yourself; embedded devices; task mapping

1. Introduction

The World Economic Forum (WEF) has a vision to shape a sustainable and reliable digital future, and the Internet of Things (IoT) is considered to be one of largest enablers for sustainable digital transformation [1]. In recent years, there has been a shift from physical space to cyberspace [2], with the sole aim to interconnect every physical thing to form a network of things (the IoT) [3–5]. The various enabling technologies proposed commonly include Quick Response (QR) codes ,and Radio Frequency Identification (RFID) for the identification of things. Moreover, wireless sensor networks and related networking technologies are exploited to interconnect these smart objects to form an IoT space. The main motivation of the IoT is to enhance business processes concerning efficiency and costs in enterprise systems [6–10]. Every physical object, in such a smart space, which connects to the IoT is called an edge node which senses ambient scenarios and bridges the gap between the real world and networks [11]. The massive number of links enables the interaction between edge

nodes in such IoT-empowered smart spaces. One of the recent advancements in the field of IoT are mission-critical Internet of Things (MC-IoT) applications. In MC-IoT, operations are performed in real-time by physical nodes and failure of the deadline could have severe consequences [12,13]. Apart from the real-time behavior of MC-IoT, these systems must exhibit properties such as safety, interoperability, and security [14]. The use of cloud technologies has enabled intelligent IoT services as it acts as a server to process and hosts a massive volume of contextual data emitting from edge-nodes of the IoT [15]. The IoT and cloud are so often integrated and so have recently become known as the Cloud of Things (CoT) [15–17].

A growing trend in CoT is to exploit the cloud as an open platform to connect every digital and physical thing and host the massive amounts of contextual data that benefits from being connected. Ericson estimates that more than 25 billion physical devices are expected to be connected to the Internet by the end of 2020, and Cisco estimates that this number will grow to 500 billion by 2030 [18]. When considering IoT systems, regarding both the hardware and software, heterogeneity is very important. In other words, very similar software functionalities are required to be deployed on heterogeneous devices which have a confined set of common core features. Additionally, things may be lightweight, thus having constrained resources and a minimal battery; this makes the situation even more complex concerning the deployment and re-deployment of software functionalities on vastly different devices with different capabilities. On the one hand, the enormous amount of heterogeneous hardware empowering an IoT smart space brings forth an unprecedented opportunity to raise the quality of life. On the other hand, the challenges of heterogeneity, run-time adaptability, privacy, and context-aware processing need to be tackled in order to benefit from the advantages that the IoT can potentially bring [19].

Among IoT systems, an important class is that of MC-IOT systems, i.e., IoT systems running applications whose failure may have severe consequences [19]. As illustrated in a Hewlett Packard Enterprise (HPE) report [5], a survey that involved 200 information technology and business decision-makers, mission-critical computing is becoming increasingly important. As an example, one of the respondents of the performed survey reported: "Without it [mission-critical computing] we will cease to remain even reasonably competitive in our crowded market". It is important to note that in mission-critical applications every single element is critical, from the most straightforward RFID tag to a database, to a robot. For what concerns user experience, damage to any of the elements in the mission-critical system would produce the same effect. The design of mission-critical IoT, unlike traditional mission-critical systems, is very challenging because of the non-deterministic delays of networks. However, it also brings many benefits, as long as the missions are monitored and planned remotely without being on premises. Therefore, in their design, the involvement of stakeholders is more important than that of technical people.

The stakeholders of mission-critical applications are the general public who usually lack technical depth, and in order to enable them to tinker with the design and planning of these critical systems, a do-it-yourself (DIY) visual programming language needs to be adopted [20]. The DIY principal has been gaining popularity in recent years. It not only enables the general public to design systems with zero coding but also presents the architecture more robustly and cleanly. Different DIY-based systems are presented in the literature which focus on the ease of the design process and enable non-technical users to design systems without the need to dig deeper into the technical world [20]. As described, mission-critical systems require a very clear and robust picture of the design and architecture due to the severity of its consequences. Therefore, the DIY paradigm suits best for such systems. In this paper, we present a multi-platform architecture which uses a DIY approach to design and monitor missions in an autonomous and optimal way. The architecture employs autonomous task–object mapping and deployment. The tasks are presented with the top-most virtual objects having a suggestion index higher than others for autonomous mapping. The approach is adopted because the task and object will form a small microservice which can be deployed on physical hardware. The size of the data is very crucial in mission-critical systems due to the stringent requirements of time and performance.

Therefore, it is preferred to deploy a small message more often than a big chunk of the message. A case study was performed on the proposed architecture which intelligently detects smoke in a smart building and prevents the hazard of fire. The contributions of this paper include

- The design and implement a multi-platform architecture for mission planning and analysis based on the DIY paradigm to encourage the involvement of non-technical people in mission design;
- The proposal of an intelligent task–object mapping model for microservice deployment to decrease message size and improve efficiency;
- The implementation of a case study regarding the the proposed architecture to test its effectiveness.

The rest of the paper is organized as follows. Section 2 presents related work and outlines the relevant research in MC-IoT and DIY-based IoT applications. Section 3 exhibits the proposed system model and describes its various components. Section 4 portrays the design in terms of mission-planning and interaction models. Section 5 discusses the implementation environment and outlines the tools and technologies being used in this work. Section 6 presents a case study involving smoke detection mission planning on the proposed architecture and illustrates the execution flow of the system. Section 7 evaluates the performance of the system with state-of-the-art methods and discusses its significance. Section 8 concludes the paper and identifies future directions of this work.

2. Related Work

The IoT is revolutionizing everything, and there are billions of IoT devices around us as a consequence [18]. As part of this evolution, traditional consumer-based applications such as smart homes based on applications and wireless body area networks are being shifted to mission-critical applications such as remote surgery and remote energy distribution in a smart grid. With this proliferation of applications, the requirements of mission-critical applications are starting to receive special consideration and be seen as a distinct set in contrast to traditional consumer-based IoT applications. One reason for this shift is that the devices involved in mission-critical applications must work every time with a zero percent chance of failure. Failure could have severe consequences; therefore, these applications have a specialized requirement in which not only the application architects should be involved but also the stakeholders which are generally not technical people [21].

A variety of studies have been conducted in the area of mission-critical applications. One of the important aspects of these applications is the communication delay [22]. With the advances in 5G networks and Industry 4.0, the design of MC-IoT systems gaining greater attention; this is partly because the ultra-latency which 5G networks support will remove the hurdles which have long been restricting these applications [22]. For instance, in [23], the authors focus on ensuring availability by establishing alternative connectivity such as device-to-device and drone-assisted access to help achieve the requirements of the MC-IoT. The data dissemination in MC-IoT is another challenge caused by the message transfer delay between devices. The work carried out by Muhammad et al. [24] focuses on data dissemination in an optimal way, finding the best tradeoff between conflicting performance objectives such as spatial frequency reuse and transmission quality. Similar efforts [25–28] have been made to improve the latency and to overcome the challenges to ensure communication; however, none these studies consider the design aspects of these applications, instead focusing on the communication layer of the design.

As far as the design of MC-IoT systems is concerned, it has been noted that ample clarity is very important. The model-driven engineering approach has some presence in the literature regarding the design of MC-IoT systems [14,19]. It has been stated that with this model-driven approach, the technical challenges of MC-IoT system development can be achieved by utilizing features such as high-level abstraction of heterogeneous devices, separation of concerns, self-adaptation, and reusability [14]. The problem with this is that although it is a very good idea, there are no case studies or instances in which the effectiveness of MDE design of mission planning is assessed. Another paradigm worth considering is the DIY approach [29]. As modern IoT applications demand the involvement of the

general public, and non-technical people have business shares, an approach to allow them to create their own systems with minimal technical effort was developed [20,29]. This approach offers several advantages: firstly, it gives the humans involved a sense of creativity and achievement. Secondly, regarding the economic interests of the stakeholders, it allows them to interact with the actual system rather than just giving mere feedback. Accordingly, it has been stated in the literature that the end users should be involved in the creation process, thus giving them the power to discover new things [30,31].

Numerous notable open-source tools propose utilizing the DIY approach and enabling the masses to become involved in the creation process. In the field of electronic device design, open-source hardware boards are used, which enable the general public to tinker with them and express themselves. Raspberry PI [32] and Arduino [33] are among the leading examples. On the software end, there are many open-source tools to facilitate the end users in the process of making. SAM [34], Node-RED [35], Glue.thing [36], and Super stream collider [37] are some of the leading tools which allow drag-and-drop features which facilitate coder and end-users who have little technical knowledge. Despite the above studies, there is very little focus on the design of MC-IoT applications using the DIY approach. This paper proposes a toolbox for mission planning which supports DIY features and at the same time stores the configuration in the cloud to enable triggering and monitoring of the devices from anywhere. This, to the best of authors' knowledge, is the first approach of its kind. A comparative analysis of the proposed system with reference to existing state-of-the-art (SoA) methods is shown in Table 1.

Table 1. Overview of the state-of-the-art (SoA) methods with reference to the proposed architecture.

SoA	Goal	PoC	MC-IoT Support	DIY Support	Open Source	Target Audience	General Purpose
[23]	Mobility in MC-IoT applications	No	Yes	No	Partial	Users	No
[24]	Optimal Data Dissemination in MC-IoT networks	No	Yes	No	No	Users	No
[26]	General-purpose plateform for low-powered, reliable, and guaranteed real-time data dissemination and analysis	Yes	Yes	No	No	Users	Yes
[27]	Designs a protocol for MC-IoT applications to model deterministic latency and improved throughput	No	Yes	No	Yes	Application Designer	No
[25]	Proposes uses of edge computing to optimize the latencies in communication	Yes	Yes	No	No	Application Designer	No
[14,19]	Proposed a Model-Driven approach for the design of MC-IoT applications	No	Yes	No	No	Application Designer	Yes
[20]	Remote Monitoring and Configuration in Smart Space	Yes	No	Yes	Yes	General Public	Yes
[34]	Provide SAM template to setup business cases	Yes	No	Yes	Yes	General Public	Yes
[35]	IoT Application Service Composition	Yes	No	Yes	Yes	General Public	Yes
[36]	Wiring data of web-enabled IoT devices	Yes	No	Partial	Yes	General Public	Partial
[37]	Web-based interface for building IoT mashups	Yes	No	Yes	Yes	General Public	Yes
Proposed System	Enabling the envolvement of stakeholders in the design of MC-IoT applications	Yes	Yes	Yes	Yes	General Public	Yes

3. Proposed System Model

In this paper, the author proposes a novel architecture which consists of three main components. The first one is a DIY-based toolbox aimed at assisting the overall mission design. The DIY approach is used because MC-IoT applications have very clear requirements and have to be evaluated before deployment on actual physical devices. The DIY approach, in this regard, is the preferred approach because once the requirements are clear, the design is straightforward, even by the general public. The proposed design is shown in Figure 1. The mission-planning toolbox has a registry of mission-critical devices. It creates tasks based on the profile of these devices. The tasks are

autonomously mapped onto the device virtual representations based on an approach which will be described in the subsequent section. The configuration is stored in a cloud-based web server. The web server is where the end users can interact and monitor their applications. The system makes use of modularization and has three different sub-modules. In the next subsections, each sub-module is illustrated.

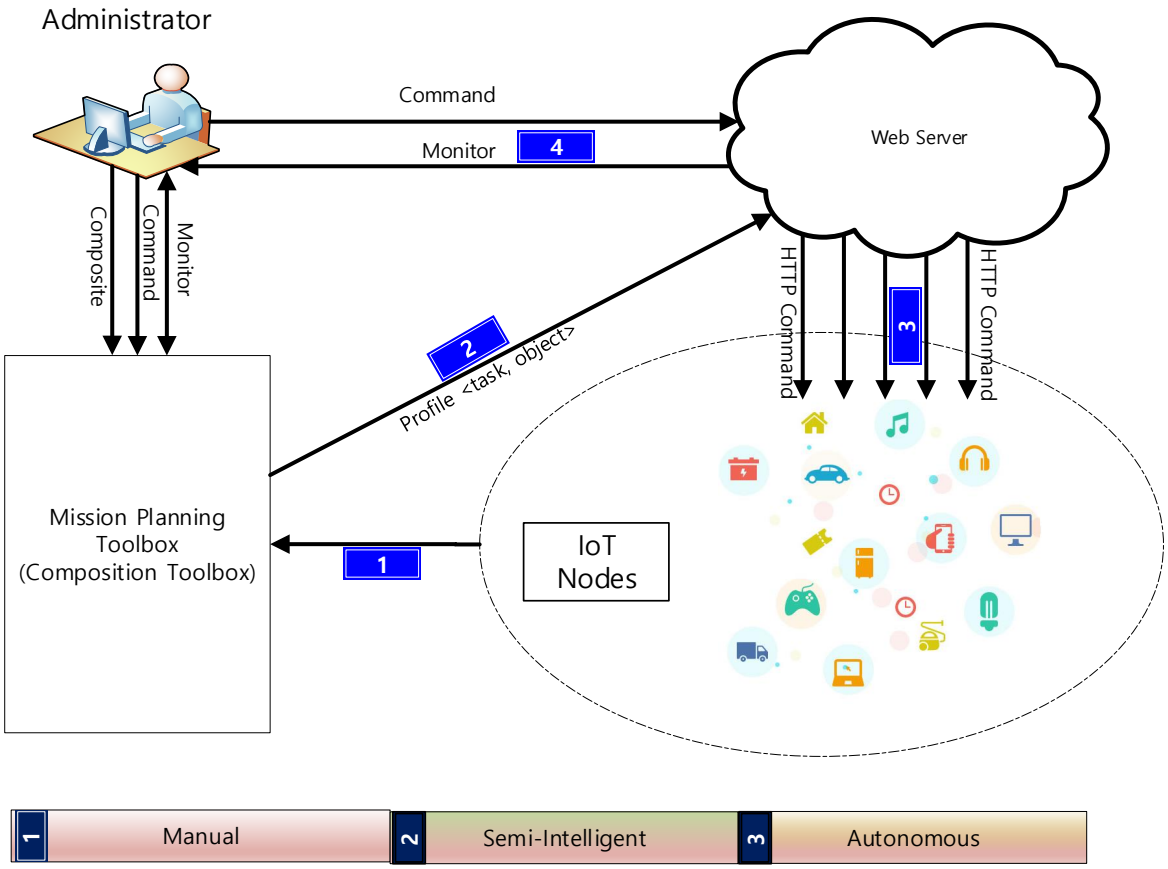


Figure 1. Proposed architecture overview.

3.1. Cloud-Based Web Server

The cloud-based web server is a central repository for all the configuration, and additionally, the number of operations that have been performed. Figure 2 shows the expanded version of the cloud-based web server. There are four main sub-modules: Profile Receiver, Profile Parser, Mission Planning, Client Application.

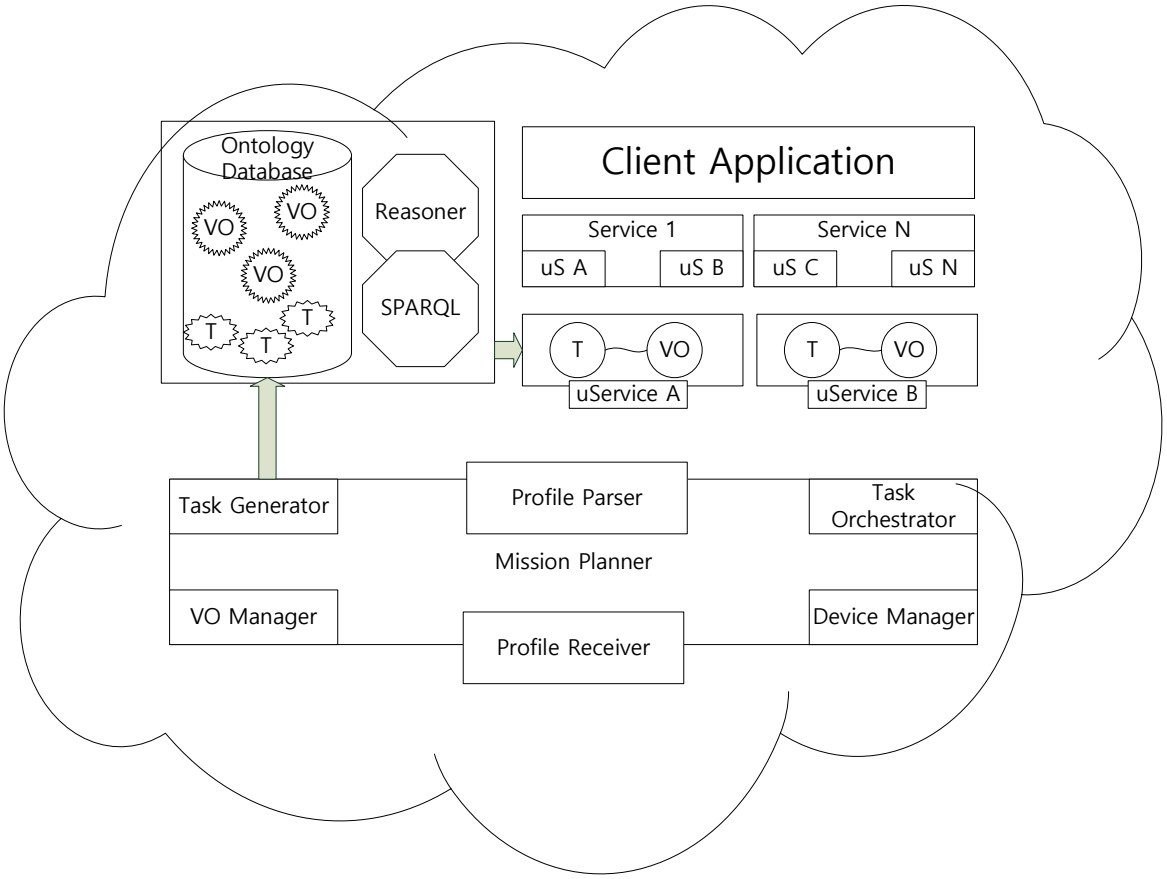


Figure 2. Inner view of cloud-based web server.

The profile receiver receives input data, which contains resources information and the services. The input data is composed of virtual objects (VO) in eXtensive Markup Language (XML) format. A VO has resource name, resource location, resource URI, and properties.The XML format of the configuration of the service is the combination of the resources needed to perform a useful operation. The combination of these operations which are deployed as a unit is called microservice denoted by *uS* and *uService* in figure. The interface through which the communication occurs from the mission-planning toolbox is two-fold: DIY to the web server using web sockets and DIY to the IoT server based on Raspberry PI using XML Constrained Application (CoAP) commands. Finally, the IoT server to the web server using HTTP requests and queries the string parameters. The profile parser parses the request and gives it to the mission-planning toolbox. The data passed is in the form of XML, or a query string, and the protocols used are HTTP and CoAP. The data which are passed include resources ID, location, and URI on the IoT server. The interface through which data are shared also has two forms: web sockets for web transfer, and General Purpose Input/Output (GPIO) for IoT server. The data is parsed and presented in a more useful way for visual representation and persisted on a web-based MySQL database. The mission-planning sub-module provides the overall logic for mission planning. The input data for the mission-planning sub-module are sensors and services, and the output data are actuators. The data are represented in the form of XML. The interfaces are web-browsers for the web end and GPIO for the IoT server. Lastly, the client application module which is an abstract view for the end users where end users can plan their missions and leverage the application power to enable their critical missions and plan them accordingly. The data on this layer is visualized using better User Experience (UX) interfaces to provide them with an easy way to interact with the underlying system. Interfaces are well designed using the latest front-end tools such as bootstrap and HTML5/CSS3. The configuration objects which are created as a result of the drag-and-drop operation in the DIY toolbox are represented in XML and passed to the web end which

in turn represents it in an interactive UX interface for sufficient clarity of the mission to the users. In this layer, the client can also plan their mission by adding the mission statement and the necessary resources needed for the mission.

3.2. Mission-Planning Toolbox

The second vital module of the proposed system is the DIY-based mission-planning toolbox. The architecture of the mission-planning toolbox is represented in Figure 3.

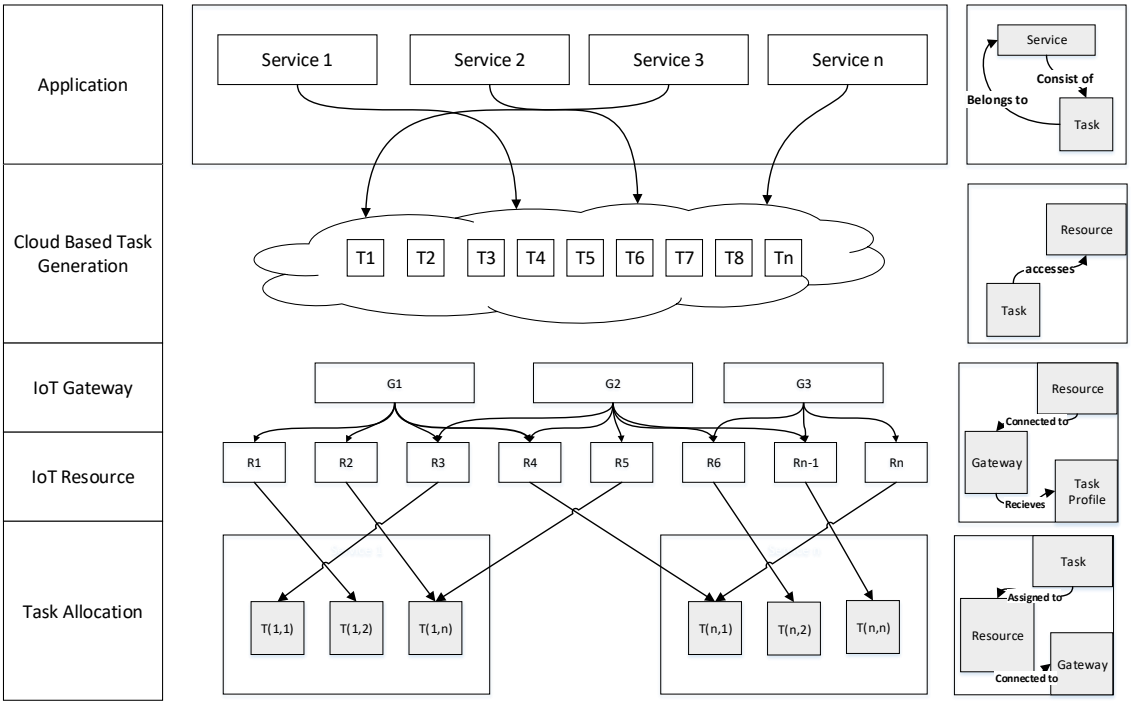


Figure 3. Layer representation of the proposed architecture.

The mission-planning toolbox is also a layered architecture. The top layer is the service layer, which provides mission-critical services to end users. The mission-critical services' configuration is stored in the cloud-based web server. In the cloud-based web server, the tasks are generated based on specific requirements such as device profile and service description. Beneath the cloud layer is the IoT gateways layer, which is responsible for managing the gateways. The gateways are important because tasks are mapped onto the virtual objects using the nearest gateway. Every gateway has specific physical resources associated with it. A physical resource can be managed by a gateway, but the same resource can communicate with a different gateway. Finally, the task allocation layer is the layer which performs task allocation to the physical resources. For every layer, semantic vocabularies are maintained which serve a repository for the metadata of that layer. This helps in task allocation, for example finding the overall information such as which resource is connected to which gateway.

3.3. Raspberry PI-Based IoT Gateway

The third vital module is the IoT gateway, which in most cases is the bridge between the application and the embedded IoT resources. There are numerous roles of the IoT gateway, one of which is the protocol conversion. For instance, the industry standard application protocol is HTTP, but for constrained devices, the optimized and lightweight CoAP is preferred. Thus, the role of the gateway is to translate one request into another, making the process transparent to the users. For this,

the IoT server is deployed as part of the gateway to listen to HTTP requests and translate them into CoAP commands to operate the connected IoT resources such as sensors and actuators.

4. System Design

In this section, the design and architecture of the proposed system is discussed in terms of the modular model which is discussed in Section 3. The primary role of the architecture is the use of tasks to design and plan missions. The design part of the mission includes task generation, task mapping, and allocation. In the following subsections the approach for mission planning is illustrated.

4.1. Task Mapping and Allocation Model

In this paper, the motivation is a clear design of mission-planning systems. For this, a task–object method is proposed. The tasks and objects belong to the mission are received, and the mapping is performed. The mapping is performed using two methods: manual and autonomous. In manual mapping, the mission-planning toolbox has a plan in which the visual programming tool is integrated to enable DIY makers to perform drag and drop operations. We use JSPlumb javascript library to serve this purpose. In autonomous mapping, the mapping is performed according to the highest suggestion index for a virtual object. Supposing we have a vector V_o, representing the virtual representation of the IoT resource, hereafter called virtual object, and a vector τ, representing a task:

$$V_o = \begin{bmatrix} id \\ name \\ methods \\ attributes \\ tags \end{bmatrix}$$

and

$$\tau = \begin{bmatrix} id \\ title \\ period \\ execution \\ tags \end{bmatrix} \quad (1)$$

Suggestion Index η for task τ_i is a function of the attributes name, methods, attributes, and tags of any virtual object V_o. In this work, the suggestion index is computed on the basis of task name, tags and the methods of VOs. Let ω be the similarity index vector $[\omega_n \ \omega_m \ \omega_a t]$ which contains weights for name correlation, methods correlation, and attributes correlation. A user-defined constant α is multiplied with ω to boost the value according to the case explained in the lemmas listed beneath.

Lemma 1. *If a task name contains the sensor name, it has the highest correlation with the virtual object. For task τ_i, the similarity index ω_n is found as:*

$$\omega_n(\tau_i) = \sigma(\tau_i[title], V_o[name]) \quad \forall \quad V_o.$$

For name, the value of α is set as 0.5 and is given the highest boost among all others.

$$\omega_n(\tau_i) = \alpha\omega_n(\tau_i) \ \ where \ \alpha = 0.5.$$

Lemma 2. *If a task name has matching keywords with the virtual object methods name, it has second best correlation.*

$$\omega_m(\tau_i) = \sigma(\tau_i[title], V_o[methods]) \quad \forall \ V_o.$$

For methods, the value of α is set as 0.25 as methods correlation has got lower priority than that of name.

$$\omega_n(\tau_i) = \alpha\omega_n(\tau_i) \;\; where\; \alpha = 0.25.$$

Lemma 3. *If a task name has matching attributes with a virtual object, it also contributes but to a less extent.*

$$\omega_a(\tau_i) = \sigma(\tau_i[title], V_o[attributes]) \quad \forall\, V_o.$$

For attributes, the α value is set as 0.1 because it plays a lesser role in finding the correlation.

$$\omega_n(\tau_i) = \alpha\omega_n(\tau_i) \;\; where\; \alpha = 0.1.$$

Lemma 4. *Tags of tasks are the keywords which are associated with tasks. Similarly, tags are also stored for indexing virtual objects. The number of similar tags in both contributes to the suggestion index, but a number of dissimilar tags also play a role in computing suggestion node.*

$$\omega_a(\tau_i) = \sigma(\tau_i[tags], V_o[tags]) - \frac{1}{\sigma^{-1}(\tau_i[tags], V_o[tags])} \quad \forall\, V_o.$$

For tags, the value of α is set as 0.5 and is considered of equal importance as name.

$$\omega_t(\tau_i) = \alpha\omega_n(\tau_i) \;\; where\; \alpha = 0.5.$$

On the basis of the above lemmas, the suggestion index η for τ_i is computed as follows:

$$\eta[\tau_i] = \sum_{j=0}^{4} \omega_j.$$

The flow of operations in planning missions and task–object mapping is described in Figure 4.

The mission-planning toolbox is responsible for the making and planning of the missions and the mapping and allocation of the tasks on IoT virtual objects through the mission plan. The mission plan connects with the cloud-centric application to get the mission object using a RESTful Application Programming Interface (API) . The mission object has reference to the tasks and virtual objects which are relevant to the received mission. The received data are parsed, and tasks and objects are created. The mapping plan is then populated with the tasks and virtual objects on the left and right end, respectively. The suggestion index is computed for each task and appended with the task to facilitate manual mapping and autonomous mapping. Once the mapping is done, it has been asserted if a certain object has more than one task, then the scheduling algorithm comes into place to provide consensus among tasks of that particular object. If one-to-one mapping is done, then the tasks are allocated and deployed on the physical devices. Different device-specific function calls are considered for the proper execution of the tasks on the physical resources. The mission is evaluated against its goal and policies, and if the policies and goals are not met, it calls the task–object mapping back.

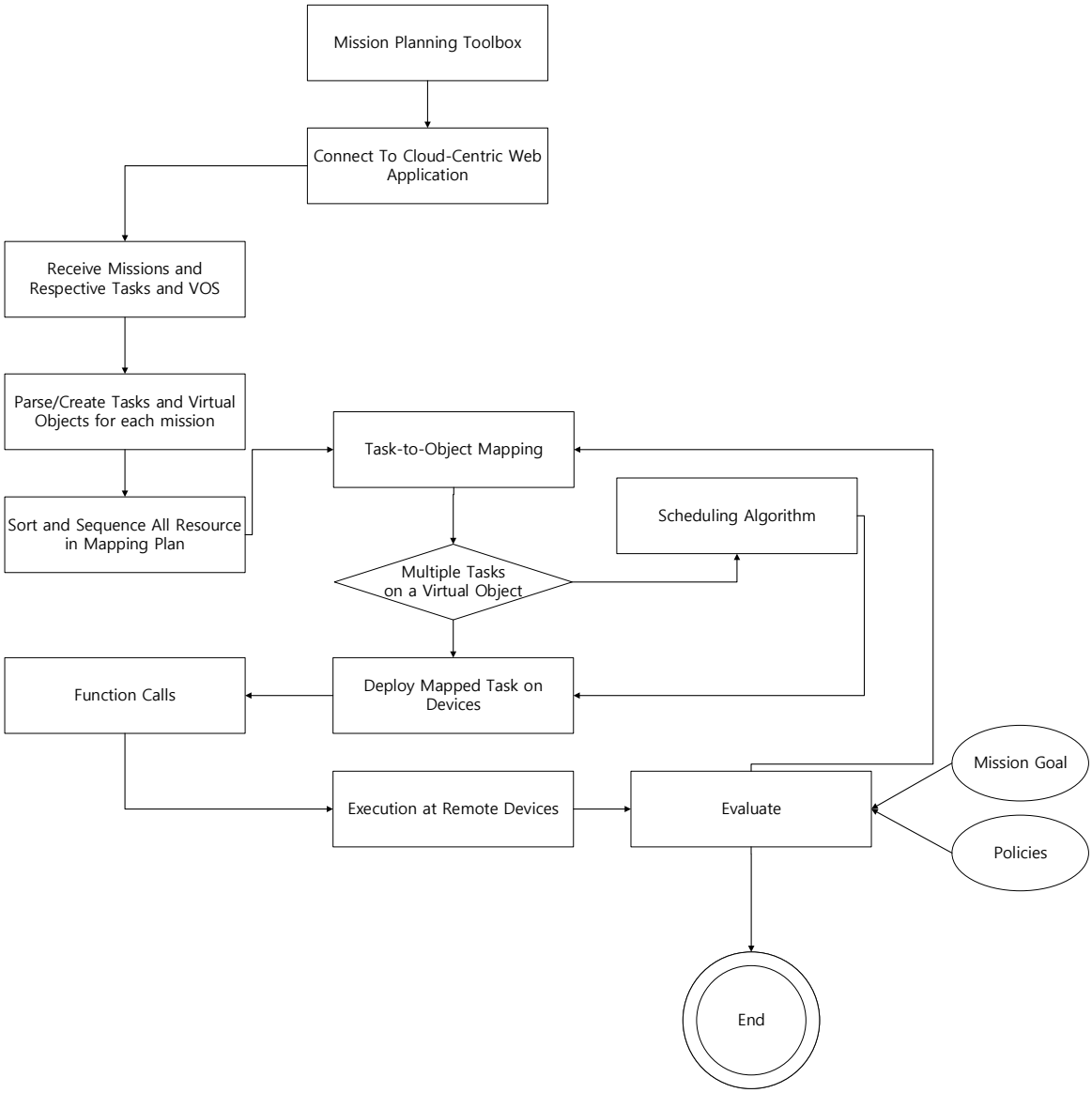
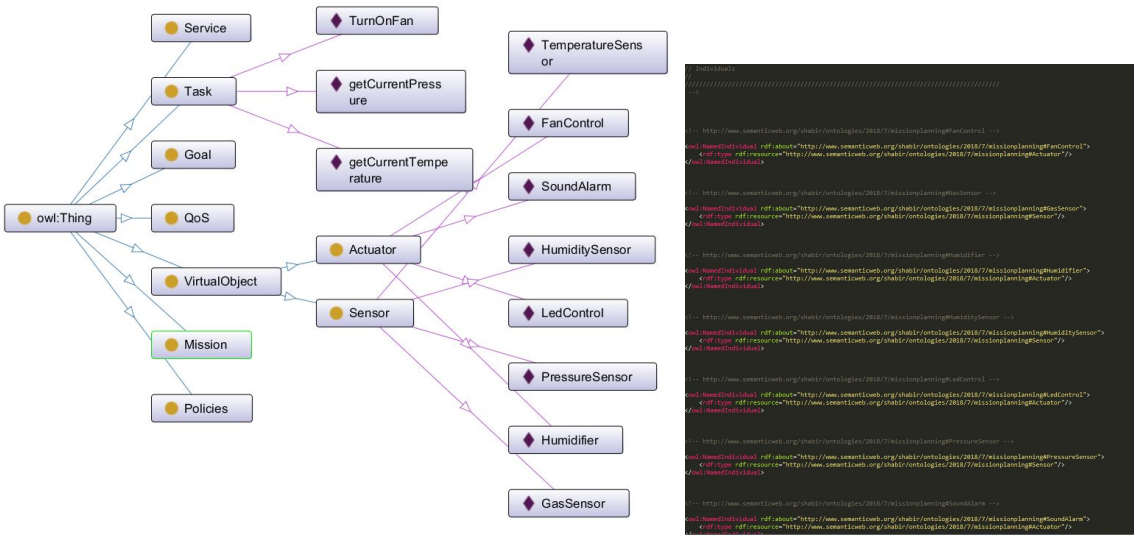


Figure 4. Flow of operation in planning missions and task–object mapping.

4.2. Task–Object Semantic Modeling and Inference

A knowledge base of semantic models of the proposed system is maintained. The knowledge base is stored using the Protege ontology editor. The knowledge base contains individuals for tasks, virtual objects, tags, and their relationship information. The inference system based on tag mapping provides the candidate virtual object for the task. The semantic representation of the ontology is provided in Figure 5. In Protege, every class is a subclass of the owl:thing class. In this architecture, there is a class for the mission which represents the semantic modeling of the mission and its ontology. A mission has a goal which must be achieved. In addition to the goal, a mission has a set of policies such as security, maximum latency, reliability, etc. Similarly, the Quality of Service (QoS) of the mission is also one of the major things that need to be considered. A mission is composed of many services, each of which consists of tasks. Each task is mapped onto a certain virtual object. A virtual object has two subclasses, namely sensor and actuator. There are several individuals created for the task, mission and sensors. The corresponding XML representation for the Ontology Web Language (OWL) graph is shown in Figure 5b.



(**a**) Ontology graph (**b**) XML representation

Figure 5. Semantic modeling of proposed system.

4.3. Interaction Model

One of the vital factors in the design of any architecture is the analysis of the interaction among its different modules. This interaction can be best described using a Unified Modeling Language (UML) sequence diagram. In this subsection, we describe the interaction of a sequence among various processes of the proposed system. The proposed system communicates across two main subsystems, i.e., the embedded plane and mission-planning plane. On the mission-planning plane, tasks are generated, mapped, and the configuration is stored. The tasks are then allocated based on the persisted configuration. The sequence diagram is shown in Figure 6. As can be seen, in this work, we have two main subsystems. The embedded plane and mission-planning plane. In the IoT Gateway, which hosted on the embedded plane, resides a Flask-based IoT server which at first initializes and registers the physical devices connected with it. The devices can be sensors and actuators. The registry information is passed onto the application. The bootstrapper component of the application is the central place which runs for the whole lifespan of the application. The bootstrapper prepares the tasks' parameters and passes them to the task generator. The task generator creates the tasks in loop based on the parameters passed to it by the the bootstrapper. Once the tasks are created and persisted, the resultant task data are loaded to the mapper. The physical device data is also loaded to the bootstrapper which in turn passed to the VO composer. The VO composer creates virtual objects based on the physical device information and passes it to the mapper. The mapper passes all the data to the correlator module which analyzes the tasks with the provided VO and returns the correlation index. On the basis of the correlation index, the tasks are mapped onto the best virtual objects. The information is communicated to the task scheduler which schedules the tasks and returns the status information to the bootstrapper, which in turn passes it to the IoT gateway. The IoT gateway uses the information to deploy the correct task on the physical devices.

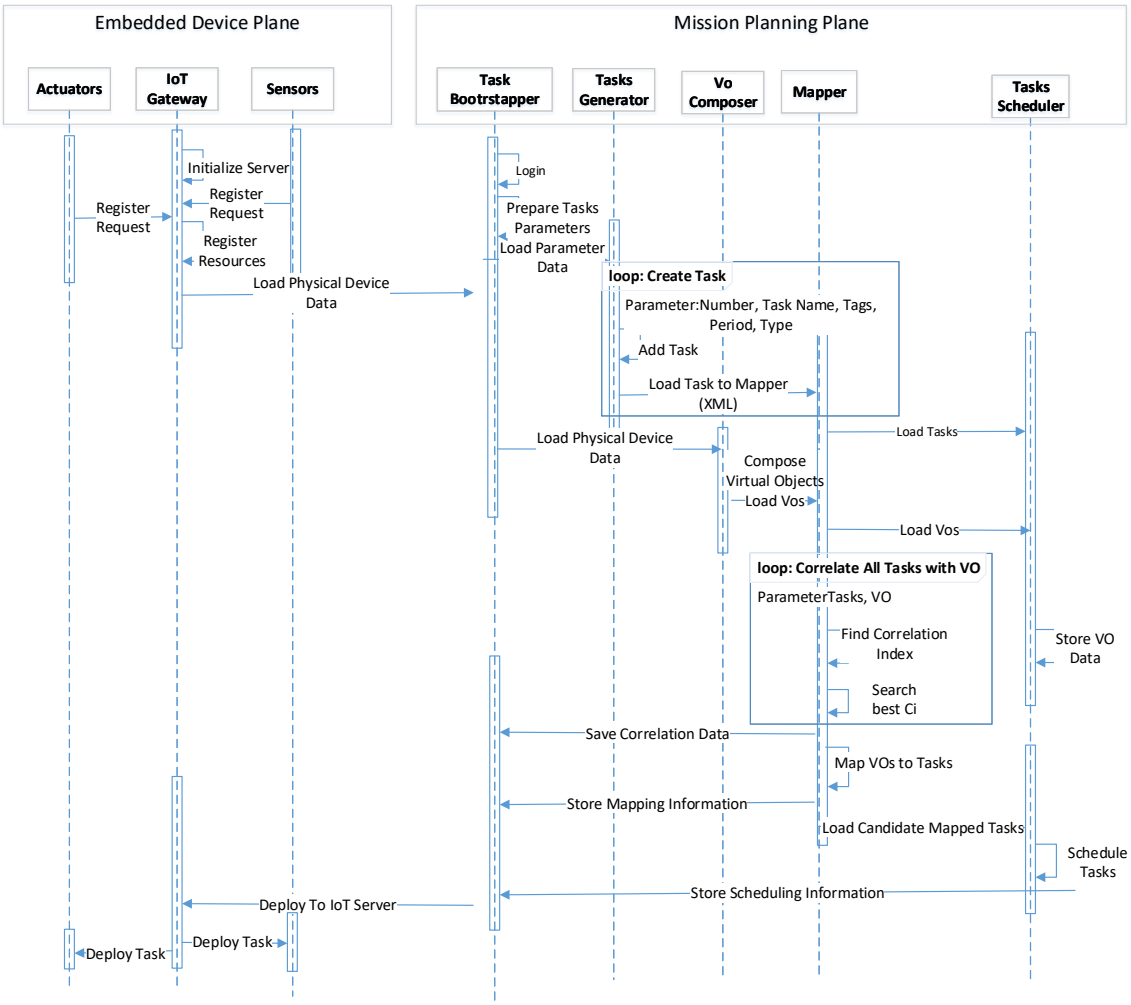


Figure 6. Sequence of operation among different subsystems of the proposed architecture.

5. Implementation Details

The proposal of each novel piece of architecture is illustrated with the model, architectural design, and then finally by the implementation. In the previous sections, we discussed the modeling of the system and the design considerations of the architecture. In this section of the paper, we summarize the technologies and tools utilized while implementing this work. As described in Figure 1, the proposed architecture has three sub-modules: mission-planning, the cloud web server, and the IoT gateway. Therefore, the implementation tools are summarized in their respective tables. Table 2 reviews the tools and technologies utilized for the IoT gateway which is based on Raspberry PI model B. Physical IoT resources such as sensors and actuators are connected with IoT gateway. Furthermore, the IoT server is deployed on the gateway to listen to the request from cloud-based wep client application. The server is implemented in Python 3. We use a popular Model View Controller (MVC) framework built on top of Python 3, named as Flask.

Table 2. Overview of implementation technologies for IoT gateway based on Raspberry.

Component	Description
Hardware	Raspberry PI 3 Model B
OS	Raspbian
Memory	1 GB
IoT Server	Flask Webserver
Physical IoT Resources	RGB LED, Fan, Temperature Sensor, Humidity Sensor, Breadboard
Additional Add-ons	GPIO, Untangle for XML Parsing, Jinja Template
Editors	Vim, Sublime Text 3, PyCharm
Programming Language	Python 3

The second sub-module of the proposed architecture is the cloud-based web server. Table 3 summarizes the technologies and tools used in it. The application is hosted on Amazon Web Service Elastic Compute 2 (AWS EC2). It is implemented using Python 3 as in the case of the IoT gateway. The Front End User Interface (UI) is implemented using Bootstrap 3, which is a responsive design-based framework.

Table 3. Technology stack of cloud-centric web applications.

Component	Description
Operating System	Linux AWS EC2 Compute Node
IDE	Sublime Text 3, PyCharm, IDLE
Programming Language	Python 3
Framework	Flask, Javascript, HTML5, CSS3
Libraries	MySQLAdapter , Bootstrap3, Jinja3 for templating
Core Programming Language	Python 3
Browser	Chromium, Google Chrome, Firefox, Safari
Server	Apache Webserver
Persistence	MySQL
Semantic Modeling	Protege 5.2
Visualization	OWLViz, OntoGraf

The last sub-module of the architecture is shown in Table 4, which shows the technology stack used to implement the mission-planning toolbox utilizing the DIY approach. The prime role of this sub-module is to allow users to configure the system without any real programming skills. For this, an HTML 5 canvas is provided where the mission configuration is developed by simple drag-and-drop features.

Table 4. Technology stack of mission-planning toolbox.

Component	Description
Operating System	Windows 8, 64 bits
CPU	Intel (R) Core(TM) i5-4570 CPU @ 3.20 GHz
Primary Memory	12 GB
IDE	PyCharm
Programming Language	Python 3.6
Framework	Flask 2
Communication	RESTFul API
Libraries	Jinja 2, HTML 5, CSS3, Bootstrap 4, jPlumb
Persistence	MySQL
Additional Tools	PHPMyAdmin, MobaXterm

The technology stack for each sub-module is selected considering its compatibility with the features of the MC-IoT applications. For instance, Python 3 was used as a core programming language; it is backed in many recent research studies because of its equal popularity in research projects as well as in development [20]. Moreover, Flask architecture is selected because it is very lightweight, and consequently the response time is minimal. For the cloud, EC2 is used rather than dedicated IoT services such as AWS IoT or Azure IoT because the proposed system utilizes the cloud to persist and monitor the missions remotely, i.e., without the need of being on the premises. Thus, using such specialized services would have been overkill for such a job. Finally, for the DIY components, JSPlumb is used because it is really flexible and easy to use. It has a robust API and customization of the existing

features is easy. It is a JavaScript library, thus mitigating the need to install any additional tools to run it because of the native support of JavaScript in every client browser.

6. Intelligent Smoke Detection and Notification Case Study

To assess the significance of the proposed system, a proof-of-concept case study was implemented utilizing the proposed architecture. A simple mission was considered, whose goal was the detection and notification of smoke in a smart home to prevent it from fire hazards. Many resources, like buzzer actuator, temperature sensor, humidity sensor, $C0_2$ gas sensor, and pressure sensor were used and deployed to detect smoke and set off the alarm. The mission is summarized in Table 5.

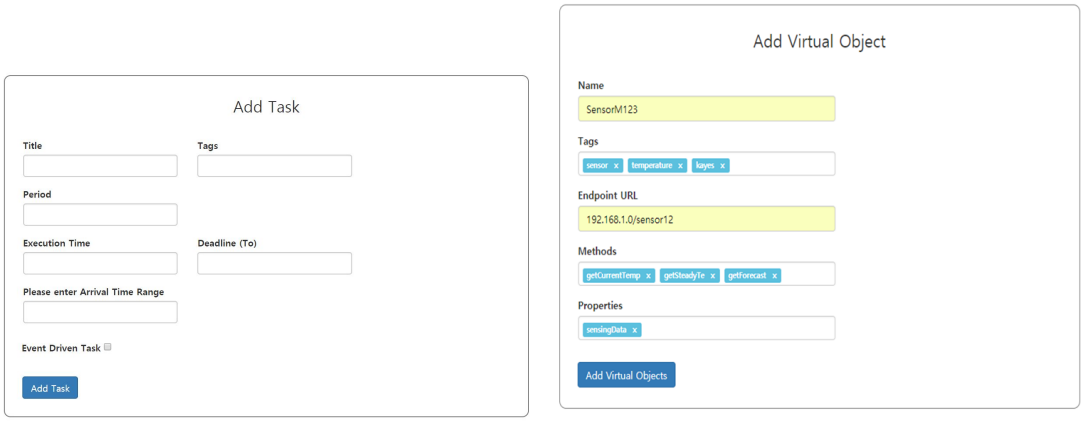
Table 5. Intelligent smoke detection and notification mission overview.

Attribute	Value
Goal	To prevent hazards occurring as the result of a fire
Policies	Security, Efficiency, Scalability, QoS
Services	Smoke Detection, Alarm Notification
Microservice	getTemperature->TemperatureSensor, getHumidity->HumiditySensor, getPressure->PressureSense, getCurrentGas->GasSensor, ringBuzzer->BuzzerActuator, turnLEDRED->RGBLed
Tasks	Get Current Temperature, Get Pressure, Get Current Atmosphere Gas, Get Current Humidity, Ring Buzzer, Turn LED green, Turn LED Red
Virtual Objects	PressureSensor, HumiditySensor, GasSensor, RGBLed, BuzzerAlarm
Gateways	Raspberry PI, Flask
Physical Object	BMP(180) Sensor Kit, MQ-2 Gas Sensor, RGB Led, VersaSense Buzzer Actuator

In order to execute the mission on the proposed prototype, mission tasks were added. IoT resources were registered and virtualized to generate virtual objects. The tasks were mapped onto virtual objects based on the correlation index. Once the mapping was done, the mapping pairs were deployed on physical IoT resources. In the following subsection, the flow of the mission addition and execution is illustrated.

Execution Flow

First off, the mission related entities were generated which include tasks, virtual objects, and the mapping methods to form the microservices. The cloud-based web application has numerous modules for task management, mission management, and virtual objects management. Figure 7 shows a web form in which different entities are added. Figure 7a shows an interface for task addition whereas Figure 7b shows an interface for virtual object creation.

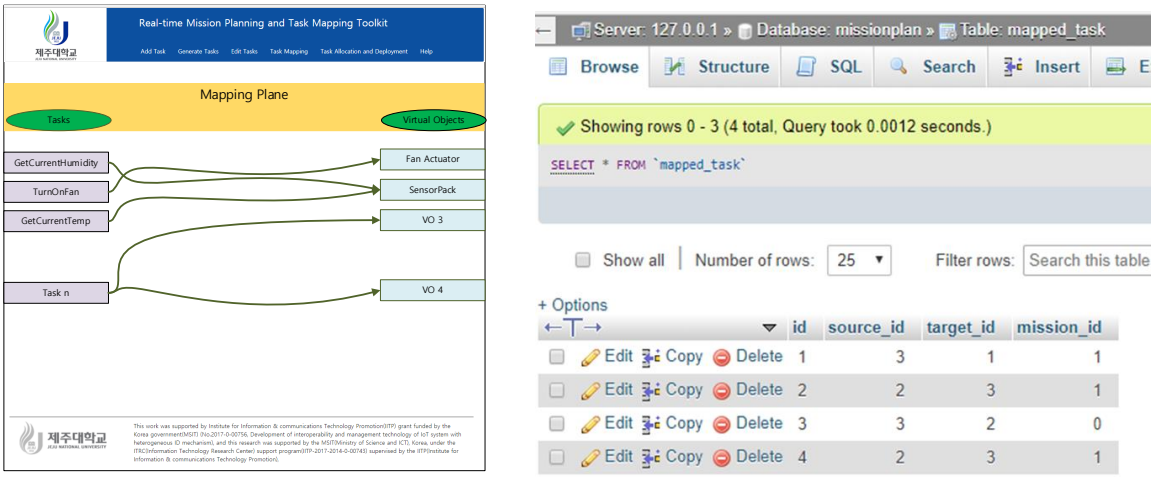


(**a**) Task adding interface (**b**) Virtual object adding interface

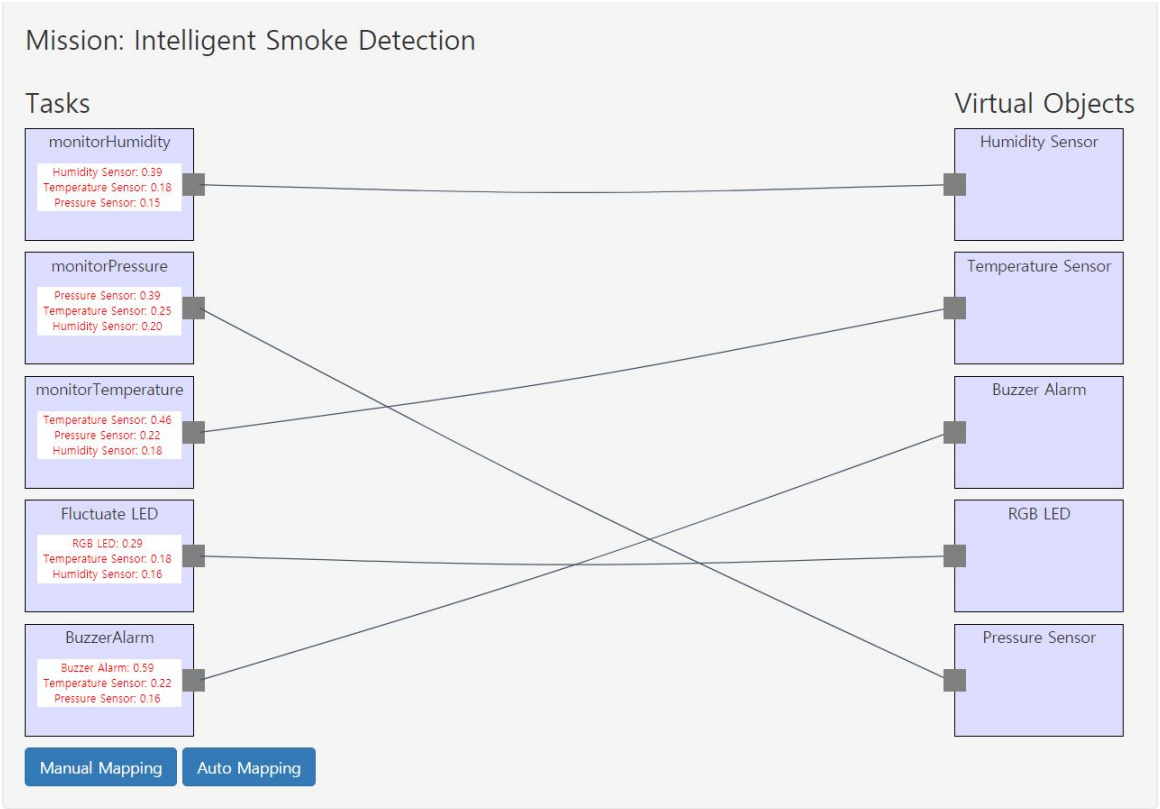
Figure 7. Tasks and virtual objects management.

Each form has attributes relevant to the entity being added. A task can have tags, period, execution (periodic tasks only), and arrival time. Additionally, if the event-driven checkbox is marked, the following three attributes will be ignored. For virtual objects, information about tags, methods, properties, and URI are stored. This toolbox has the RESTFul endpoint, which exposes the data in XML format to the mission-planning toolbox. Once the tasks and virtual objects are added, the next step is the mapping of tasks on virtual objects based on the correlation among them. The mapping is done in a DIY plane which enables stakeholders to configure and make microservices. This plane is also called the task–object deployment plane. In this plane, the left side is populated with tasks while the right side is populated with virtual objects. The mission plane gets these data using RESTful API from the cloud-based web application. Once the tasks are mapped using JSPlumb drag-and-drops, the mapping configuration is persisted in the MySQL database against the mission. In the mapping process, the tasks can be mapped either in a manual fashion or in an automatic way. In manual mapping, suggestions are shown against each task along with the index of the suggestion. The mapping plane can consider the suggestions or can withdraw in case the suggestions are not fully accurate. The highest suggestion index represents the virtual object which is the most suitable for the designated task. Suggestions are based on the similarity index of names, methods, and attributes of the task and the common tags among them. Figure 8 shows the mission-planning interface for the smoke detection and notification case study.

In the task–object deployment plane, as discussed in the earlier sections, the task can be mapped manually or autonomously. The manual mapping is necessary because of the critical nature of IoT missions: a small error can lead to a hazardous situation. In manual mapping, suggestions are shown against each task along with the index of the suggestion. The mapping plane can consider the suggestions or can withdraw in case the suggestions are not fully accurate. The highest suggestion index represents the virtual object which is the most suitable for the designated task. Suggestions are based on the similarity index of names, methods, and attributes of the task and the common tags among them. Figure 8 shows the mission-planning toolbox. Figure 8a shows the mapping plane and the resultant MySQL persistence. Figure 8b is a screenshot taken from the mission-planning toolbox which contains sensor and actuator tasks and the virtual objects along with their suggestions relevant to the case being presented. Lines are drawn from tasks to virtual objects. The task is mapped onto a virtual object and is stored in cache temporarily. Once the manual mapping button is clicked, the configuration of the plumbing is persisted in the database. In case of an autonomous mapping, once the respective button is clicked, every task is mapped onto the virtual object which has the highest suggestion index. If plumbing is done before the autonomous mapping, the configuration will be ignored. For instance, in Figure 8b, it is evident that each task has a suggestion box which shows the top 3 virtual objects. For instance, the monitorHumidity task has three virtual objects suggestions listed. The highest of them is the Humidity sensor with the value 0.39. Similarly, for all the remaining tasks, the respective suggestion boxes provide an idea of the candidate virtual object.



(**a**) Mapping plane and configuration persistence



(**b**) Correlation index-based suggestions

Figure 8. Pictorial representation of mission planning with mapping plane and JSPlumb.

Once the task–object deployment is done, the resultant [task-vo] pair is persisted, which is called a microservice. The term microservice is used because the pair is an atomic deployable operation for achieving the mission goal and is very lightweight. These microservices are deployed on physical IoT resources as shown in Figure 9. The microservices are deployed in the form of HTTP requests. The gateway has Flask, a server which listens to the HTTP requests. The cross-platform requests are performed using the Cross Origin Request Service (CORS) module of Flask, which in addition to the policy checking also validates and executes the request if authorized. The gateway receives the request and gives the command to IoT resources.

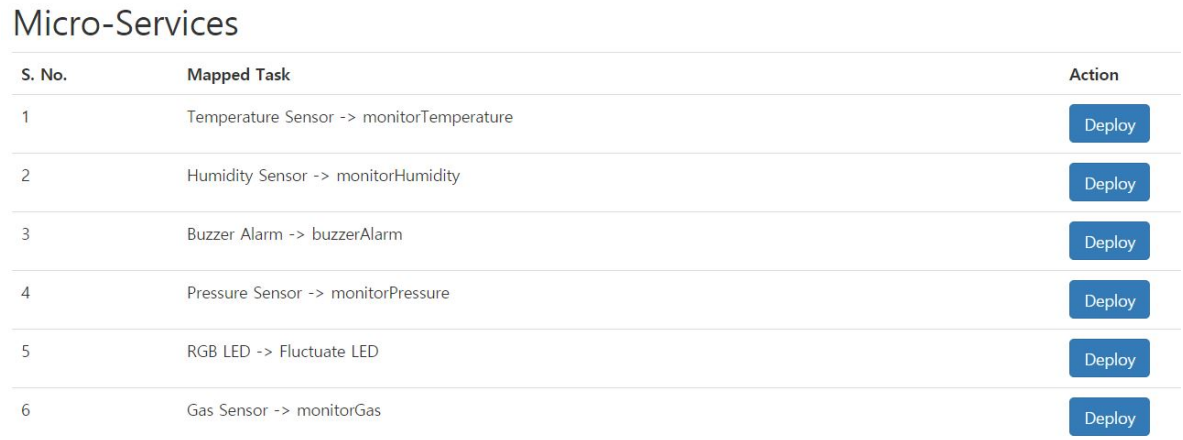
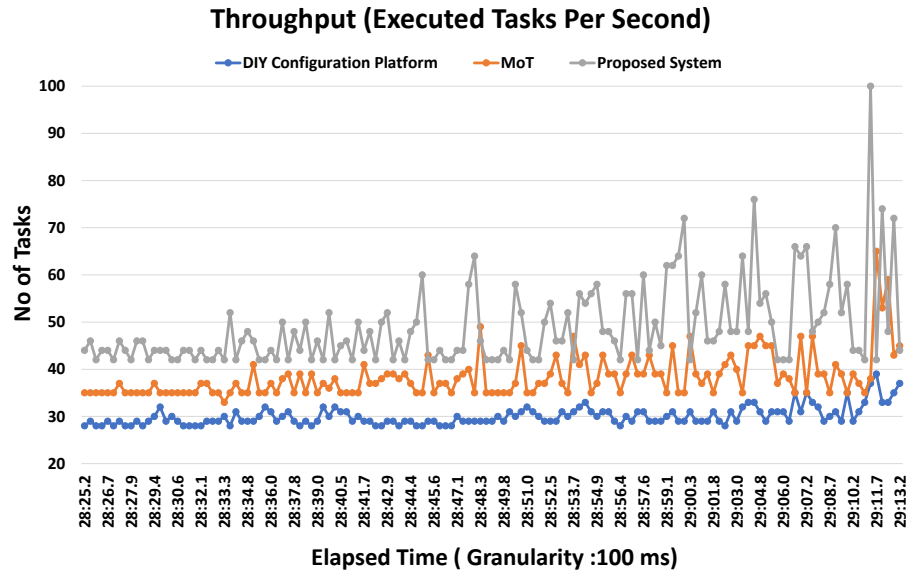


Figure 9. Microservice deployment interface.

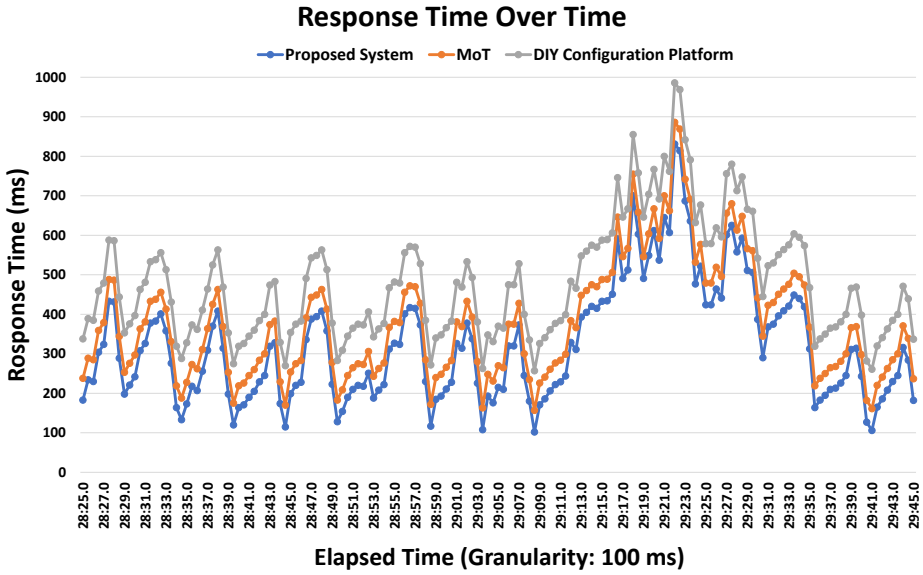
The tasks are deployed on their respective physical devices once the deploy button is clicked. Since most of the microservices are polling tasks, they are deployed once and then repeatedly monitored to detect the event, which is the presence of smoke. The RGB led and Buzzer alarm are event-driven tasks which are only executed if smoke is detected.

7. Performance Evaluation and Discussion

For mission-critical IoT systems, one of the most important factors is the performance of the system. These systems should meet the properties of real-time systems, and thus, the response time of these should be minimal enough to satisfy real-time constraints. We compared the proposed platform with two relevant state-of-the-art solutions. The first solution is purely designed for MC-IoT systems, whereas the second solution is more of a DIY platform claiming to be generic for most of the categories of IoT systems. To test the performance of the proposed system, we simulated these systems for 50 virtual users and investigated the average response time and throughput of the respective systems. For this, the open-source tool Blazemeter was used. We recorded sample cross-domain requests using Apache JMeter and imported them into Blazemeter, which in turn simulated the script for a set of virtual users. The test statistics were recorded and visualized in graphical form, as shown in Figure 10. Figure 10a illustrates the throughput of the proposed system in comparison with the two approaches. It is evident that for 50 users, the throughput was approximately 55 tasks per second on average. In other cases, the throughput was on average approximately 47 and 33, which is less than the proposed system by some margin. However, there were occasional spikes for the proposed system, which show that for specific requests, the throughput was better by an even greater margin. The response time over time was also recorded, as shown in Figure 10b. Much like with the throughput, the response time for the proposed system was lower than the corresponding systems. Despite the aim of MAC on Time (MoT) being latency optimization, the average response time was marginally lower than MoT. The reason for this is that the proposed system uses a very lightweight microservice, which reduces the latency on the communication medium and thus results in an overall lower response time. From these results, we see that it does not entirely outperform the existing systems, but given the range of applications, it can be considered a much better architecture for MC-IoT applications.



(**a**) Throughput for 50 users



(**b**) Response time for 50 users

Figure 10. Performance evaluation of proposed system w.r.t to the state-of-the-art methods.

Another crucial factor is the reliability of the system, which is vital for MC-IoT applications. An engine health test was performed to find the condition of the engine. The engine refers to the occupancy rate of network I/O, Memory, CPU, and connections. These attributes forecast the reliability of the system. For mission-critical applications, the engine health is of vital importance, given its critical nature. An overloaded engine must be avoided, otherwise it can lead to an unsafe state in which the mission can fail on any of its crucial requirements. The result of the engine health test for the proposed system is shown in Figure 11. It is evident that the CPU always remained below 5% and the maximum network I/O took 8 KB/Sec, which is very low. Similarly, the main memory was occupied to 10% on average irrespective of the increase in the number of connections, which proves that the system is stable and can respond even if the load increases. The per-request statistics of the test are summarized in Table 6. We evaluated the performance against parameters like average response time, maximum response time, average latency, standard deviation, 90% Line, 99% Line,

average throughput, and error rate. These are standard benchmarking parameters of apache jMeter for evaluating the performance of the test.

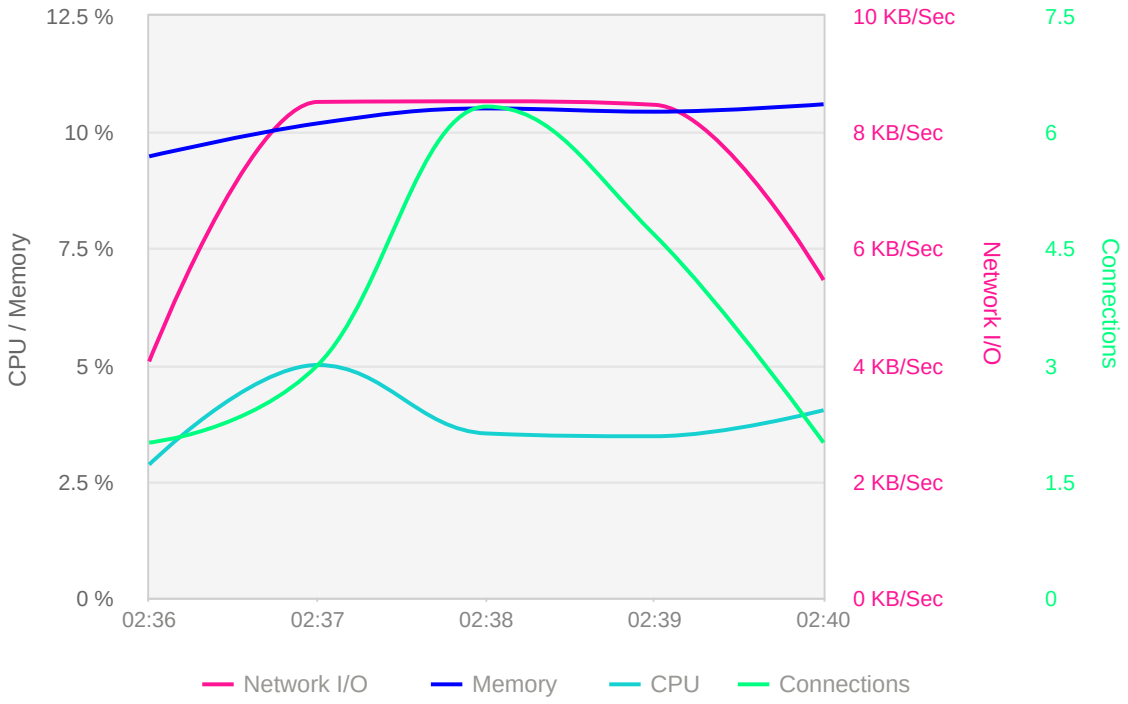


Figure 11. Engine health analysis.

From the above simulation, it is proved that the proposed system is very reliable in terms of load and scalability which is crucial for any mission-critical planning system.

Table 6. Load testing statistics overview.

Label Name	Samples	Avg Response Time	90line	99line	Max Response Time	Avg Latency	stDev	Duration	Avg Bytes	Avg Through-put	Errors Rate
ALL	4531	0.77	2	5	56	1	1.59	1200	12.8	3.776	2
addtask	660	0.385	1	1	53	1	2.143	1200	1.5	0.55	1
add- virtual- obj	660	0.489	1	1	4	1	0.552	1176	1.8	0.561	2.5
deploy- tasks	640	0.456	1	1	1	1	0.498	1167	1.102	0.548	0.6
gentasks	644	0.474	1	1	6	1	0.569	1171	1.105	0.55	1.2
map- tasks	1287	0.459	1	1	5	1	0.532	1173	2.205	1.097	2.1
Test	640	2.697	5	6	56	1	2.718	1167	6.613	0.548	1.5

8. Conclusions

In this paper, we consider the design aspects of an MC-IoT system; this field has received very little attention despite many studies focused on MC-IoT. Herein, we propose an approach based on a well-known DIY paradigm focused on letting the general public, such as the stakeholders, become involved in the design process, helping to achieve sufficient clarity, which is required for MC-IoT systems. We propose a mapping plane to autonomously map tasks onto virtual objects based on the best correlation with the devices. The tasks are deployed in the form of very lightweight microservices, which are simply references to the task and the device URI. We have utilized a modern programming language for long-term support of the proposed architecture. The architecture is further illustrated with a case study, which is a simple surveillance mission of smoke detection on a smart home. The performance was analyzed and compared with two state-of-the-art methods and was simulated under severe load conditions using Blazemeter load testing. It was found that it performs better in terms of throughput and response time than its counterparts. The engine test also indicated that it always remains in a stable region irrespective of the load. In this work, we considered the design of MC-IoT and considered the data size and load of the system. Future works regarding this paper will target other characteristics such as the security and heterogeneity of the devices in a more dynamic context.

Author Contributions: S.A. conceived the idea for this paper, designed the experiments and wrote the paper; F.M. assisted in model designing and experiments. D.-H.K. conceived the overall idea of Cloud-Centric Mission-Critical IoT planning, and proof-read the manuscript.

Acknowledgments: This research was supported by Energy Cloud R&D Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT (2019M3F2A1073387), and this research was supported by Institute for Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.2018-0-01456, AutoMaTa: Autonomous Management framework based on artificial intelligent Technology for adaptive and disposable IoT). Any correspondence related to this paper should be addressed to Dohyeun Kim.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Internet of Things Guidelines for Sustainability. 2018. Available online: http://www3.weforum.org/docs/IoTGuidelinesforSustainability.pdf (accessed on 25 June 2019).
2. Da Xu, L. Enterprise systems: State-of-the-art and future trends. *IEEE Trans. Ind. Inform.* **2011**, *7*, 630–640.
3. Vermesan, O.; Friess, P.; Guillemin, P.; Gusmeroli, S.; Sundmaeker, H.; Bassi, A.; Jubert, I.S.; Mazura, M.; Harrison, M.; Eisenhauer, M.; et al. Internet of things strategic research roadmap. *Internet Things-Glob. Technol. Soc. Trends* **2011**, *1*, 9–52.
4. Friess, P. *Internet of Things-Global Technological and Societal Trends from Smart Environments and Spaces to Green ICT*; River Publishers: Aalborg, Denmark, 2011.
5. Xia, F.; Yang, L.T.; Wang, L.; Vinel, A. Internet of things. *Int. J. Commun. Syst.* **2012**, *25*, 1101. [CrossRef]
6. Guo, B.; Zhang, D.; Wang, Z.; Yu, Z.; Zhou, X. Opportunistic IoT: Exploring the harmonious interaction between human and the internet of things. *J. Netw. Comput. Appl.* **2013**, *36*, 1531–1539. [CrossRef]
7. Leu, J.S.; Chen, C.F.; Hsu, K.C. Improving heterogeneous SOA-based IoT message stability by shortest processing time scheduling. *IEEE Trans. Serv. Comput.* **2014**, *7*, 575–585. [CrossRef]
8. Ding, Y.; Jin, Y.; Ren, L.; Hao, K. An intelligent self-organization scheme for the internet of things. *IEEE Comput. Intell. Mag.* **2013**, *8*, 41–53. [CrossRef]
9. Vlacheas, P.; Giaffreda, R.; Stavroulaki, V.; Kelaidonis, D.; Foteinos, V.; Poulios, G.; Demestichas, P.; Somov, A.; Biswas, A.R.; Moessner, K. Enabling smart cities through a cognitive management framework for the internet of things. *IEEE Commun. Mag.* **2013**, *51*, 102–111. [CrossRef]
10. Lazarescu, M.T. Design of a WSN platform for long-term environmental monitoring for IoT applications. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2013**, *3*, 45–54. [CrossRef]
11. Lima, D.F.; Amazonas, J.R. TCNet: Trellis Coded Network-Implementation of QoS-aware Routing Protocols in WSNs. *IEEE Lat. Am. Trans.* **2013**, *11*, 969–974.

12. Ahmad, S.; Malik, S.; Ullah, I.; Fayaz, M.; Park, D.H.; Kim, K.; Kim, D. An Adaptive Approach Based on Resource-Awareness Towards Power-Efficient Real-Time Periodic Task Modeling on Embedded IoT Devices. *Processes* **2018**, *6*, 90. [CrossRef]
13. Ahmad, S.; Malik, S.; Kim, D.H. Comparative Analysis of Simulation Tools with Visualization based on Real-time Task Scheduling Algorithms for IoT Embedded Applications. *Int. J. Grid Distrib. Comput.* **2018**, *11*, 1–10. [CrossRef]
14. Ciccozzi, F.; Crnkovic, I.; Di Ruscio, D.; Malavolta, I.; Pelliccione, P.; Spalazzese, R. Model-driven engineering for mission-critical iot systems. *IEEE Softw.* **2017**, *1*, 46–53. [CrossRef]
15. Aazam, M.; Khan, I.; Alsaffar, A.A.; Huh, E.N. Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved. In Proceedings of the 11th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Islamabad, Pakistan, 14–18 January 2014; pp. 414–419.
16. Skouby, K.E.; Lynggaard, P. Smart home and smart city solutions enabled by 5G, IoT, AAI and CoT services. In Proceedings of the International Conference on Contemporary Computing and Informatics (IC3I), Mysore, India, 27–29 November 2014; pp. 874–878.
17. Petrolo, R.; Loscri, V.; Mitton, N. Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms. *Trans. Emerg. Telecommun. Technol.* **2017**, *28*, e2931. [CrossRef]
18. How Far is the Hype of IoT. 2016. Available online: https://www.rcrwireless.com/20160628/opinion/reality-check-50b-iot-devices-connected-2020-beyond-hype-reality-tag10 (accessed on 15 March 2019).
19. Ciccozzi, F.; Spalazzese, R. MDE4IoT: Supporting the internet of things with model-driven engineering. In *Proceedings of the International Symposium on Intelligent and Distributed Computing*; Springer: Berlin, Germany, 2016; pp. 67–76.
20. Ahmad, S.; Hang, L.; Kim, D.H. Design and Implementation of Cloud-Centric Configuration Repository for DIY IoT Applications. *Sensors* **2018**, *18*, 474. [CrossRef] [PubMed]
21. 3 Tips to Succeed in Mission-Critical IoT. Available online: https://www.rs-online.com/designspark/3-tips-to-succeed-in-mission-critical-iot (accessed on 15 March 2019).
22. Zhang, Q.; Fitzek, F.H. Mission critical IoT communication in 5G. *Future Access Enablers of Ubiquitous and Intelligent Infrastructures*; Springer: Berlin, Germany, 2015; pp. 35–41.
23. Orsino, A.; Ometov, A.; Fodor, G.; Moltchanov, D.; Militano, L.; Andreev, S.; Yilmaz, O.N.; Tirronen, T.; Torsner, J.; Araniti, G.; et al. Effects of Heterogeneous Mobility on D2D-and Drone-Assisted Mission-Critical MTC in 5G. *IEEE Commun. Mag.* **2017**, *55*, 79–87. [CrossRef]
24. Farooq, M.J.; ElSawy, H.; Zhu, Q.; Alouini, M.S. Optimizing mission critical data dissemination in massive IoT networks. In Proceedings of the 15th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), Paris, France, 15–19 May 2017; pp. 1–6.
25. Orsino, A.; Farris, I.; Militano, L.; Araniti, G.; Andreev, S.; Gudkova, I.; Koucheryavy, Y.; Iera, A. Exploiting d2d communications at the network edge for mission-critical iot applications. In Proceedings of the 23th European Wireless Conference European Wireless 2017, Dresden, Germany, 17–19 May 2017; pp. 1–6.
26. Daneels, G.; Municio, E.; Spaey, K.; Vandewiele, G.; Dejonghe, A.; Ongenae, F.; Latré, S.; Famaey, J. Real-Time data dissemination and analytics platform for challenging IoT environments. In Proceedings of the Global Information Infrastructure and Networking Symposium (GIIS), St. Pierre, France, 25–27 October 2017; pp. 23–30.
27. Hassan, G.; Hassanein, H.S. MoT: A deterministic latency MAC protocol for mission-critical IoT applications. In Proceedings of the 14th International Wireless Communications & Mobile Computing Conference (IWCMC), Limassol, Cyprus, 25–29 June 2018; pp. 588–593.
28. Wu, N.; Liang, Q. Sparse nested cylindrical sensor networks for Internet of mission critical things. *IEEE Internet Things J.* **2018**, *5*, 3353–3360. [CrossRef]
29. Valderrama, C.; Vachaudez, J.; Bettens, F.; Vinci dos Santos, F.; Menezes, N.; Roelands, M. Architecting the Internet of Things: The DiY Smart Experiences Project: A European Endeavour Removing Barriers for User-generated Internet of Things Applications. *Archit. Internet Things* **2011**, doi:10.1007/978-3-642-19157-2_11.
30. Gama, K.; Touseau, L.; Donsez, D. Combining heterogeneous service technologies for building an Internet of Things middleware. *Comput. Commun.* **2012**, *35*, 405–417. [CrossRef]
31. Atzori, L.; Iera, A.; Morabito, G. The internet of things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805. [CrossRef]

32. PI, R. What is Raspberry PI? 2015. Available online: https://www.raspberrypi.org/help/what-is-a-raspberry-p/ (accessed on 11 January 2019).
33. Arduino. 2015. Available online: http://www.arduino.cc/ (accessed on 11 January 2019).
34. SAM: The Ultimate Internet Connected Electronics Kit. 2015. Available online: https://www.kickstarter.com/projects/1842650056/sam-the-ultimate-internet-connected-electronics-ki (accessed on 11 January 2019).
35. Heath, N. How IBM's Node-RED is Hacking Together the Internet of Things. 2015. Available online: http://www.techrepublic.com/article/node-red/ (accessed on 11 January 2019).
36. Kleinfeld, R.; Steglich, S.; Radziwonowicz, L.; Doukas, C. glue. things: A Mashup Platform for wiring the Internet of Things with the Internet of Services. In Proceedings of the 5th International Workshop on Web of Things, Cambridge, MA, USA, 8 October 2014; ACM: New York, NY, USA, 2014; pp. 16–21.
37. Quoc, H.N.M.; Serrano, M.; Le-Phuoc, D.; Hauswirth, M. Super stream collider–linked stream mashups for everyone. In Proceedings of the Semantic Web Challenge Co-Located with ISWC2012, Boston, MA, USA, 11–15 November 2012.