Hindawi Publishing Corporation
Journal of Robotics
Volume 2015, Article ID 631420, 10 pages
http://dx.doi.org/10.1155/2015/631420

Research Article

Integrating UAVs into the Cloud Using the Concept of the Web of Things

Sara Mahmoud,[1] Nader Mohamed,[1] and Jameela Al-Jaroodi[2]

[1]*College of Information Technology, UAE University, P.O. Box 15551, Al Ain, UAE*
[2]*Department of Engineering, Robert Morris University, Moon Township, PA 15108, USA*

Correspondence should be addressed to Sara Mahmoud; 201370014@uaeu.ac.ae

Received 24 July 2015; Revised 7 September 2015; Accepted 13 September 2015

Academic Editor: Shahram Payandeh

UAVs, Unmanned Aerial Vehicles, have gained significant attention recently, due to the increasingly growing range of applications. Most UAVs use radio frequency transmission to communicate with the ground station to receive commands and send data. However, this model of communication restricts the user to being in specific locations and limits missions to narrow areas. This paper proposes a Cloud Computing (CC) integration where the UAVs become part of the cloud infrastructure and can be accessed ubiquitously. This concept is similar to the Internet of Things (IoT) where smart objects are connected to the Internet and are given unique identification. Moreover, to provide an appropriate way of communication, UAV resources are developed as web services. They provide their resources and services through a uniform interface using the RESTful HTTP architecture. This concept is relevant to the Web of Things (WoT) that provides smart objects with interfaces to be accessed through the World Wide Web (WWW). However, in UAVs, not only the web services are computational services but they are also physical elements affecting and affected by the real world environment.

1. Introduction

Unmanned Aerial Vehicles (UAVs), aircraft without human pilots on board, have developed rapidly. They are used not only in military missions but also in civil applications. UAVs can be very useful in many applications like agriculture, search and rescue, security and surveillance, environmental monitoring, large infrastructure monitoring, and terrain mapping. Some UAV applications involve multiple UAVs working together to quickly achieve a specific task. However, building applications that will effectively and concurrently operate multiple UAVs and utilize them for a certain problem area requires a huge number of man hours in design, development, and testing [1]. This is mainly due to the lack of technologies that can be utilized to effectively coordinate the operations and facilitate the cooperation of multiple UAVs.

As they need to rely on some form of radio frequency communication, UAV applications need to establish direct links among themselves and with the ground station(s). These communication links can be point-to-point wireless links or can be multihop wireless links going through other intermediate nodes such as other UAVs or ground communication stations [2]. However, this peer to peer communication is not suitable for many of the UAVs dynamic distributed and heterogeneous environments. It restricts the location of the ground station to the mission's location and requires UAVs to be in direct line of sight of the ground station or available communication hubs to maintain communication and control. In addition, the control and monitoring of UAV applications become more complicated and limited to specific devices that UAVs are connected to.

Our approach integrates UAVs with the CC paradigm. CC has been expanded not only for computers and mobile devices but also for embedded systems [3]. Smart objects such as sensors, actuators, and embedded devices are connected to the Internet through the IoT architecture [4]. The main focus of IoT is establishing network connectivity between smart objects and the Internet. While the Web of Things (WoT) builds the application layer on top of the network [5], accordingly, the web tools and protocols can be used for

developing and interacting with these objects. Similarly, UAVs have embedded systems that conform to the concept of IoT and WoT such that they can be connected to the Internet to be accessed and monitored through the web. For example, a client application can monitor the mission progress as well as the status and location of each UAV through a web browser. Moreover, it enables access to the UAVs' resources such as cameras, sensors, and actuators using web service requests.

This paper continues our previous work [6], where we proposed the general framework of collaborative UAVs through CC. UAVs offer their capabilities as services to be accessed and requested as cloud services. Here we focus on monitoring UAVs' resources for autonomous UAVs. The main contributions of this paper are the following:

(i) Connecting UAVs to the Internet to be a part of the cloud infrastructure following the concept of WoT.

(ii) Providing UAVs resources and capabilities through RESTful web servers' uniform interface.

(iii) Specifying the server side and the client side roles of integrating UAVs to the cloud.

The paper is organized as follows. Section 2 offers background information on earlier efforts to integrate UAVs to the cloud and about the IoT and WoT. Section 3 introduces the Resource-Oriented Architecture (ROA) describing the UAVs resources and services and presenting the RESTful architecture used for UAVs resources interface. Section 4 describes the integration of UAVs with the cloud and the client-server web architecture. The opportunities and considerations of integrating UAVs to the cloud are presented in Section 5. Section 6 illustrates the implementation of the proposed architecture and Section 7 concludes the paper.

2. Background

2.1. UAVs and the Cloud. There are some research efforts aiming to provide ways to connect UAVs and the cloud. Lin et al. proposed connecting a UAV to cloud services such as Google Earth [7]. This was done using an Android-based smartphone that provides its data to a MySQL database. The user accesses the UAV information in the database using a web browser. UAVs are controlled using a specific flight plan defined through a waypoint in the database. Then the mission is followed using Google Earth software. The authors demonstrated the system for a single UAV. However they did not cover its use for multiple UAVs and their communication among each other. In addition, monitoring and controlling the UAVs through a database is generally an inappropriate architecture as it lacks the ability to offer real-time responses.

Another proposed approach is following the Service-Oriented Architecture (SOA) [8] as described in [9]. Here, four prototypes were developed using SOA and smartphones. The communication was supported using the SOAP/XML (Simple Object Access Protocol and Extensible Markup Language) format. The model made it possible to integrate some functions between the UAV and cloud services. However, this approach showed an overload of exchanged messages as well as high complexity in transmission. Furthermore, the video exploitation tool is another example of SOA applications for UAVs [10]. It allows the user to choose the Region of Interest (RoI) to view the UAV path as well as the video footprint on a map. The framework stores files that can be referenced using the exploitation services via SOAP documents. Here as well, the communication generates high traffic and it may not always be possible to achieve real-time interactions.

2.2. IoT and WoT. Some efforts have also been vested in IoT and WoT aiming to connect devices and embedded systems to the Internet and build applications for the client to use them. For example, [11] proposed the REST architecture by defining an object as a server that provides its resources in a Resource-Oriented Architecture. Guinard et al. used the web tools as a solution for the WoT. In [12] they proposed two methods for accessing objects. First, they connected devices for measuring power consumption to a smart gateway. It is a web server that provides its resources for the clients to monitor and control electrical devices. This architecture allows the smart gateway to calculate the overall consumption of all devices connected to it. Second, the authors provided direct access to wireless sensor networks. This allows client applications to access each sensor node in the network using a uniform interface in a way similar to accessing a web server.

There are different architectures for web services on the cloud. First, in the standardized WS* web service architecture, the client requests and service response objects are encapsulated using SOAP and transmitted using XML. Second, the Representational State Transfer (RESTful) architecture is a web service architecture that identifies resources through a uniform interface using Uniform Resource Identifiers (URIs) and Hypertext Transfer Protocol (HTTP) service interface. The resources are represented in different media types, such as Hyper Text Markup Language (HTML) and JavaScript Object Notation (JSON).

Guinard et al. [13] compared the two approaches (the standard WS* web services and the RESTful web services) for the WoT. They found that the SOAP architecture is a complicated approach. Although it is suitable for digital services with emphasis on business architecture, it is not suitable for the physical world. In addition, it requires high computing power, bandwidth, and storage. On the other hand, the RESTful architecture is a reusable and loosely coupled set of web services. Moreover, it is easier to learn and use. Furthermore, the authors recommended the use of RESTful web service for the WoT rather than the standard WS* web server unless the application has advanced security and quality of service requirements. As a result, we selected the RESTful architecture to integrate UAVs with the web. Due to the limited capabilities and resources of UAVs such as energy level and processing power, a simple lightweight web service architecture such as RESTful is more suitable than the heavyweight complex web service like the WS*.

3. A Resource-Oriented Architecture for UAVs

Integrating UAVs with the cloud means that the UAV and its resources will be available through client applications in a ubiquitous manner. The client could be either a human using

web browser and applications or a UAV accessing another one's resources. Therefore, the most important step is to identify the resources and services that should be made available to the clients. This is done by defining the HTTP interaction of the RESTful API, URI, operations, and messages for each resource.

3.1. UAVs Resources and Services. UAVs resources are assets/devices that the UAV has and can benefit from. The system monitors the status and condition of the UAV resources. These resources could provide services and functionality for the users or other UAVs. UAVs define their resources and services that vary from one to another due to the heterogeneity of UAVs. Each UAV should have uniform interfaces to enable the client to do the following:

(i) Monitoring the UAV itself: this involves monitoring the UAV's status (i.e., whether it is idle or on a mission), the current status of its storage, flight conditions, direction and orientation, speed, energy level, current position coordinates, altitude, and latitude and longitude values.

(ii) Accessing the UAV resources: this usually involves issuing requests to collect some information such as sensors readings, radar data, camera images or videos, thermal camera images, and time of day information. Services offering this type of information may require some input such as specifying the time, location, or size of data requested. Other services may also generate some form of action by the UAV or the devices on board. For example, a client request may lead the UAV to change flight direction or altitude, request pictures or video to be taken, activate devices such as sprayers or sensors, or start activities onboard the UAV. These services usually require some parameters such as what to do, when, how long, or where.

(iii) Monitoring the UAV resources' status: this involves keeping track of the different resources on board such as finding out if a certain resource is available, currently in use, or damaged. Another example is determining the remaining amount of fuel during a mission.

3.2. RESTful Architecture. The central concept of RESTful [14] is that a resource is any component worth being uniquely identified and linked to the WoT. RESTful is described as follows:

(i) Resource identification: the URI identifies the resources of each UAV.

(ii) Uniform interface: resources are available for interaction through uniform interfaces with well-identified interaction semantic, which is HTTP that has a set of operations to optimize interactions with the resources.

(iii) Self-describing message: along with the HTTP interactions, the client and server exchange messages in an agreed-upon format. In machine-oriented services, there are two media types supported by HTTP, XML and JSON. The JSON format has gained widespread support for embedded systems due to its readability for both humans and machines; it is also lightweight and can be directly parsed to JavaScript compared to XML.

(iv) Stateless interactions: the server does not hold previous interaction information that affects the following requests. Therefore, each request contains all the information needed to correctly satisfy it. The request information is contained in HTTP using a self-describing message by the JSON object.

3.3. RESTful HTTP Components. A resource is accessed through an HTTP interface. The following are the three particular parts of this interface: operations, content negotiation, and status codes.

(i) *Operations*. The RESTful HTTP has four operations: GET, POST, PUT, and DELETE. In UAVs, the GET operation is used to retrieve the representation of a resource. For example, GET with the resource URI can be used to retrieve the current energy level of a UAV or the status of the camera on board. Moreover, in UAVs, the POST operation is used to request a service and provide the required parameters if any are needed, for example, requesting a camera on board the UAV to take pictures at a certain location. In this case, the camera resource has a URI operation (i.e., POST) and the body request is the specified location to capture the pictures. There is also the DELETE operation, which is used to stop a UAV's task or remove it from the mission.

(ii) *Content Negotiation*. The negotiation between a client and a server is built into the HTTP request. It represents the exchanged messages in an agreed-upon manner to represent the needed resource information. The header supports both JSON (*application/json*; $q = 1$) and XML (*application/xml*; $q = 0.5$). These media types are specified in the Content-Type of the HTTP response. It is agreed that JSON has widespread support; therefore, the HTTP header in a UAV is set to *Content-Type*: *application/json*.

(iii) *Status Codes*. The statuses of responses have standardized status codes in HTTP. These are well known to the client to represent the status of a request. For example, a return code of 200 to the client represents the success of the request. On the other hand, a return code 400 means that the request does not follow the server request rules.

4. UAVs as Part of the Cloud Infrastructure

By connecting UAVs to the Internet, they become part of the cloud infrastructure. Each UAV is considered a web server having a set of resources and services to offer. Most UAVs support Wi-Fi connections; as a result, a UAV will have a unique identifier over the Internet. When properly identified,

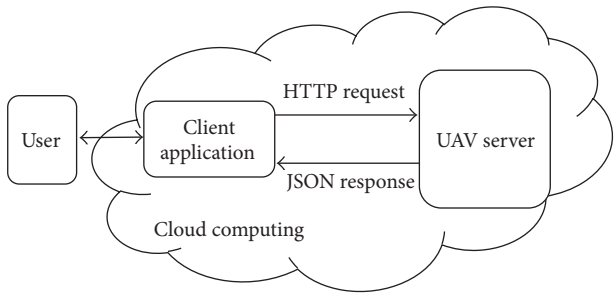

Figure 1: Client-server architecture of integrating UAVs to the cloud.

the UAVs' resources become available to the web through the RESTful Application Programming Interface (APIs).

One of the considerations in integrating UAVs with the cloud is the distribution of UAVs and their scalability and ability to offer their services and resources through the APIs. The RESTful architecture supports loosely coupled services. Therefore, using it for UAVs is the most effective solution because it is lightweight for embedded systems; it supports loosely coupled services, high usability, and flexibility; and it is a trendy technology for web services.

4.1. UAVs as Cloud Infrastructure. Integrating UAVs with the cloud makes the UAVs a part of the cloud. As a result, the cloud infrastructure is composed of the powerful servers provided by the cloud for services and computations as well as UAVs being servers offering resources and services to interact with the physical world. The web architecture is a client-server architecture, where the UAV is the back-end server that provides its service and resources on the environment, while the front-end side is the client side application that monitors and accesses these resources through standardized protocols regardless of the programming language used on either side (see Figure 1). UAV services can be written in different languages that support web services such as Node.JS, Ruby and rails, python, or PHP, while the client application can be written in JavaScript, HTML5, or Java to name a few. The variety of programing languages for the UAV side and the application side overcomes the limitations of the embedded systems programing languages that restrict the developer with a certain programming language.

To develop the UAV server side, it is necessary to make the UAV's object representation of its information as well as its resources and services. The UAV representation is a JSON object that could be easily parsed into JavaScript and also readable for humans. For example, each UAV is given a unique identifier and a name. Then, it holds its current parameters such as energy level, location coordinates, and operational status. Moreover, each UAV identifies its service object representation including the service ID, name, status, value, and requester. For instance, one can retrieve the UAV's information by requesting the UAV's URI using *GET*:

http://. . ./UAV_info

It returns the following response:

HTTP/1.1 200 OK

Content-Type: application/json

{"id": 1, "Name": "uav1", "services": ["camera", "temperature"], "energy level": 85, "status": "available", "orientation": 61.5, "location": [{"latitude": 36.872, "longitude": 140.0704, "altitude": 260}]}

However, giving a specific attribute will return a JSON object of that attribute like in the following URI using *GET*:

http://. . ./UAV/status

Returns the status of the UAV as JSON object:

HTTP/1.1 200 OK

Content-Type: application/json

{"status": "available"}

Similarly, requesting a *GET* to a service URI;

http://. . ./UAV/services/temperature

Returns a JSON object of the temperature resource under the services resource of UAV;

HTTP/1.1 200 OK

Content-Type: application/json

{"id": 2, "name": "temperature", "status": "available", "value": 28.5}

Requesting a service requires a header and body of the POST request, for example, requesting the temperature sensor for a specific location using *POST*,

http://. . ./UAV/services/temperature

with the JSON body of the request is defined as:

{"location": [{"latitude": 12.8145, "longitude": 45.64827, "altitude": 87.91}]}

Returns the JSON object with the new values:

HTTP/1.1 200 OK

Content-Type: application/json

{"id": 2, "name": "temperature", "status": "available", "value": 30}

However, if the UAV service is requested by another client or it is unavailable during that request it returns:

HTTP/1.1 200 OK

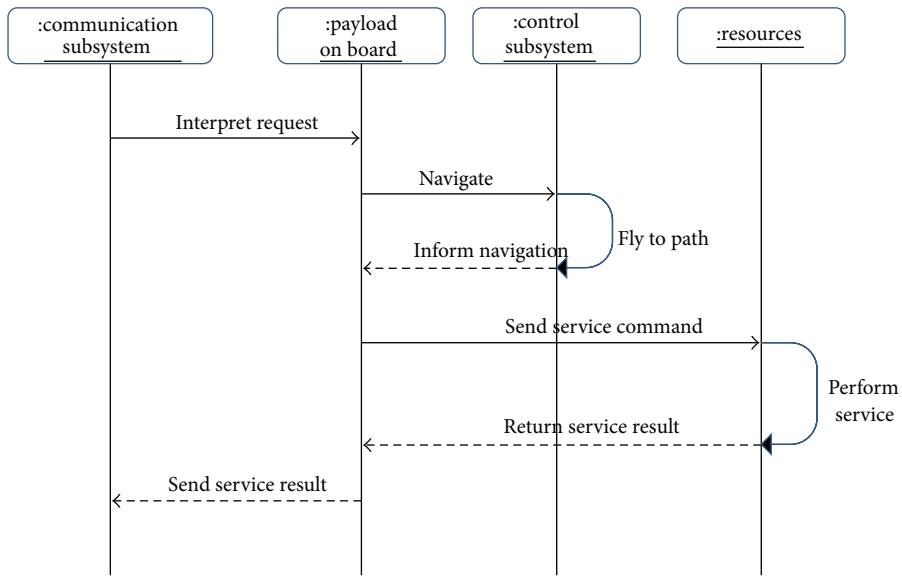

Figure 2: Service request sequence diagram for UAV subsystems.

Content-Type: application/json

{"id": 2, "name": "temperature", "status": "unavailable"}

These URIs are expressive and present their meaning for human interpretation. In addition, the retrieved JSON object is easy to parse into JavaScript. Note that for the previous examples the returned status code is *200* that represents a successful request. Moreover, *Content-Type*: *application/json* indicates that the client accepts messages that are in JSON object format. The received JSON message could be then represented in the browser for the user in an HTML file. RESTful URIs of resources are considered a tree representation not just a string of a URI. This means that a temperature is a resource of services of the UAV resource. Moreover, these resources and services are accessed not only by the users but also by other UAVs. This allows each UAV to offer its resources and services to other UAVs through these uniform interfaces. This access among UAVs enables the implementation of collaborative UAVs applications.

When the UAV receives the request through the communication subsystem (i.e., Wi-Fi or 3G/4G), the request is then passed to the onboard computer to be interpreted to the requested UAV API. In case the service is requested for a certain location, the control subsystem gets the specified location and navigates to it. When the location is reached, the control subsystem informs the onboard computer. After that, the onboard computer sends the command to the requested resource which accordingly performs the service and returns the result. Finally, it marshals the return message so that the communication subsystem sends it as shown in Figure 2.

4.2. UAV Application Side. The front-end application is on the client side and it is what the client uses to access services and receive responses. It is deployed to connect to the cloud and interacts with the UAVs' back-end servers using RESTful.

Table 1: A comparison between pull model and push model for requests and data exchange.

Pull (AJAX)	Push (Comet)
Sends requests frequently to the server	Sends the data when an event occurs
Client workload	Server workload
Suitable for requesting the current value	Suitable for getting notifications of event occurrence
Suitable for monitoring slow/continuous changes	Suitable for monitoring sudden/intermittent changes
Fast changes require low time intervals	Requires client subscription to an event

The application has a user-friendly interface designed mostly for web browsers. This interface provides users with the ability to monitor and access the UAVs' resources easily (e.g., see Figures 3 and 4 for an application's user interface on a browser).

As a result of the loosely coupled RESTful architecture, the application layer is built easily on top of the web services. Therefore, different applications could be built for the same UAVs using their uniform interfaces without modifying the server side. This is also useful for offering UAVs as services for multiple client applications with varying missions. Each application needs to know the uniform interfaces of the UAVs' resources to easily request them using the agreed-upon format and get the results through a user-friendly interface. There are two main designs for the front-end applications for monitoring and accessing resources [15] as shown in Table 1.

(i) *AJAX Model.* Asynchronous JavaScript and XML (AJAX) has proven to be a good way of transferring some of the server workload to the client. In this

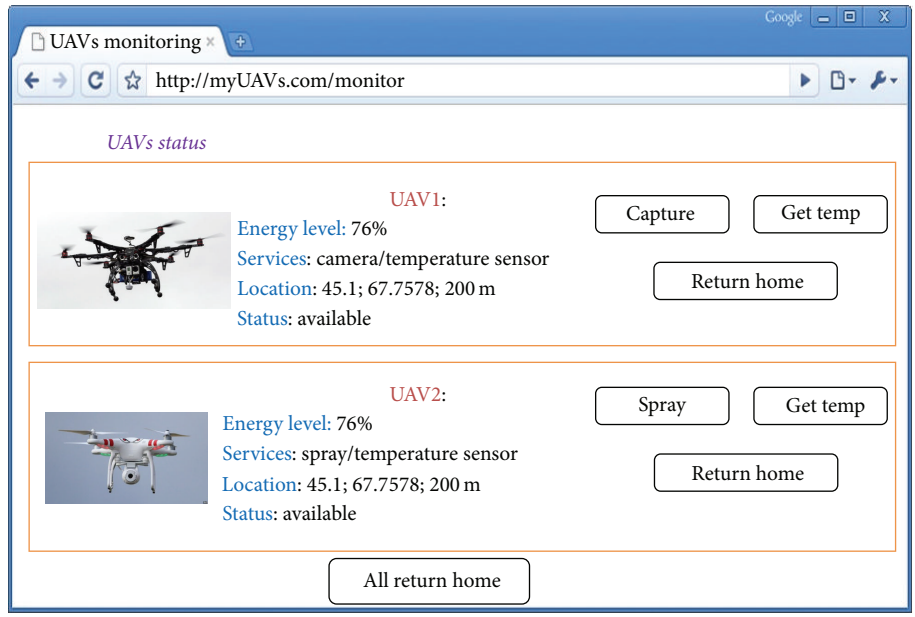

Figure 3: A user-friendly interface on browser of the client application for monitoring two UAVs.

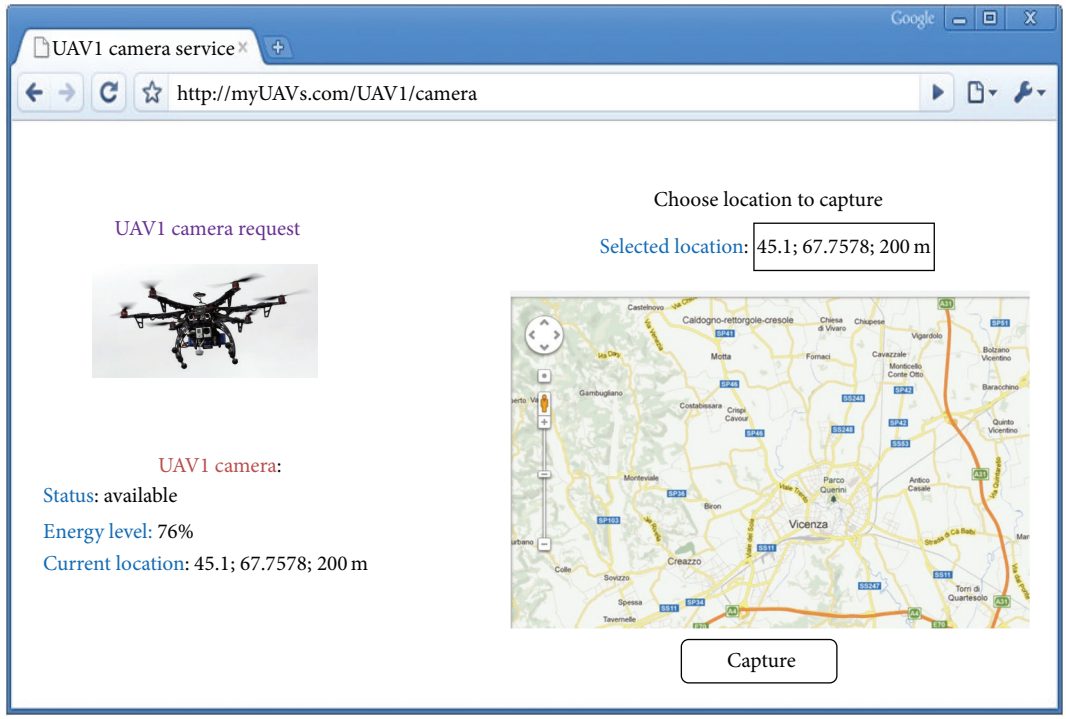

Figure 4: Client user interface application for requesting camera service from UAV1.

model the application content is refreshed automatically without refreshing the whole page. AJAX reduces web traffic because dynamic contents are only semantic information interpreted by the client. The workload is deported from the web server to the browser. In this model the client pulls data from the UAVs. This is suitable for the limited capabilities of UAVs; therefore the load is on the client side.

(ii) *Comet Model*. Here the server pushes the data to the client when an event occurs. Although this model seems suitable for monitoring, it suffers from two main limitations. First, it is built using the classical web content generator, thus not allowing request handles. Second, the UAVs have to store information of each client listening to the events. As a result, this model puts the load on the server, thus straining its

already limited resources and restricting the application to a limited number of clients that the server can identify.

5. Opportunities and Considerations

When considering the integration of UAVs as services and/or service providers in the cloud, several issues need to be considered and carefully addressed.

5.1. Opportunities. There are many opportunities that Cloud Computing opens to collaborative UAVs. The ubiquitous property of Cloud Computing allows users to monitor the UAVs and use the platform from anywhere at any time. In addition, as the cloud has a huge infrastructure of processing power, most UAV data computations could be made on the cloud rather than the UAVs, which reduces the UAV consumption of power. Moreover, Cloud Computing provides large and scalable storage services that can be used rather than the limited UAV storage. As a result, storing data on the cloud increases reliability by ensuring data back-up, thus offering access to previous log data even when the UAV is out of service. Furthermore, Cloud Computing provides ubiquitous services such as Google Earth 3D maps and computations that can be integrated with the UAV services to develop larger, more versatile, and more efficient applications.

The cloud uses web service APIs and standardized communication protocols to request services and exchange data. Therefore, heterogeneous UAVs can use these standards regardless of their operating systems and commands. The standardized protocols make the development easier for building heterogeneous UAVs in different programming languages that are used in web applications. Not only that but also the standardized protocols afford the ability to integrate other nodes and components that use the same standards as the UAV application such as ground nodes and sensor networks. Furthermore, adding more UAVs or resources is easier by registering these UAVs to the platform as plug-and-play, so that UAVs are attached to the mission at run time. Additionally, the web service architectures support reusability so that the UAV resources are used for different applications according to their availability. In addition, the users do not have to own the UAVs but only use them as services. This decreases the cost for users and open huge business opportunities for utilizing UAVs as services where they are provided. Another advantage is that UAVs resources are pooled so they can be used by multiple users.

5.2. Considerations. UAVs have limited capabilities in memory, processor, and energy; therefore, they require a lightweight software and web services that do not heavily consume their resources. UAVs should be developed following the platform web service APIs to ensure the communication ability between UAVs and the platform. Moreover, UAVs' locations play an important role in task operations such as capturing specific areas. UAVs require an efficient method to allocate their positions with minimum power consumption, for example, the trade-off between GPS and Wi-Fi triangulation.

The availability of some services depends on some contexts such as the UAVs' locations, energy levels, or specific sensor readings. Therefore, if a UAV is currently near the mission location, it is preferable to choose it rather than a similar UAV which is far from the specified location. Moreover, UAV flight control algorithms should be provided for real-time execution and path planning management as well as collision avoidance. Internet connection reliability is another important consideration. UAVs require continuous connectivity to the cloud so that they can access the cloud and their resources to be invoked through their APIs. The assumption of a reliable connection is valid for operations in city areas such as smart cities. Otherwise, the operation location should be provided with connection infrastructure for the UAV operation. Besides, the services provided by the UAVs are real world services; thus they sense and affect the physical environment. UAV services that make changes in the environment such as spraying should be managed carefully; that is, these services should not be duplicated over the same area. In case of a repeated request, there should be approval or acknowledgment before performing the service.

6. Implementation

The proposed Cloud Computing framework was implemented in real Arduino hardware and requester software to illustrate the resources and service APIs for UAVs.

First, the hardware: the UAV was built using the Arduino board (http://www.arduino.cc/) which is open source hardware for embedded systems. For this research, the Arduino was implemented as the UAV payload subsystem that is the onboard device for resources and services, and then sensors were connected to the Arduino such as DHT11 (https://github.com/adafruit/DHT-sensor-library) sensor for temperature and humidity, ultrasonic for distance measurements. In addition, a buzz and some LEDs were attached to represent actuators as shown in Figure 5. Moreover, for the Internet connectivity, an Adafruit CC3000 Wi-Fi board (https://www.adafruit.com/products/1469) was used to connect the Arduino to the Internet and get an IP address. The Arduino was developed using the Arduino software (http://arduino.cc/en/main/software) in C language with the Adafruit CC3000 library (https://github.com/adafruit/Adafruit_CC3000_Library) to read the requests. Each resource was implemented with a RESTful API.

The UAV resources were implemented for four UAVs. Each one has different resources, IP address, and RESTful APIs. However, UAVs that have a similar resource define their API interface in the same way. For simplicity, only GET method was used for the implementation. The implemented UAV resources and services are summarized in Table 2. The implemented system was tested using the Postman Chrome extension (https://www.getpostman.com/docs/requests) for each device and resource. The test focuses on the pull data model of HTTP requests. The test begins with testing the UAV resource APIs, by requesting the UAV RESTful HTTP by its address, URI, and operation for each resource. The UAV got the request, defined the service, performed it according to its

Table 2: Implemented UAVs resources and their interfaces.

UAV1	/temp	Gets the temperature from the DHT sensor
	/humidity	Gets the humidity from the DHT sensor
	light/1	LED turns on
	/light/0	LED turns off
UAV2	/lighting/1	LED blinks on and off continuously with time interval of 200 ms
	/lighting/0	LED stops blinking
	/spray/1	Buzzer beeps continuously with time interval of 200 ms while decreasing the tank capacity
	/spray/0	Buzzer stops beeping
	/spray/tank	Returns the remaining tank capacity
	/spray/tank/full	Refills the tank capacity to the maximum
UAV3	/temp	Gets the temperature from the DHT sensor
	/humidity	Gets the humidity from the DHT sensor
	/lighting/1	LED blinks on and off continuously with time interval of 200 ms
	/lighting/0	LED stops blinking
UAV4	/distance	Returns the distance in centimeters from ultrasonic sensor
	/lighting/1	LED blinks on and off continuously with time interval of 200 ms
	/lighting/0	LED stops blinking

Table 3: Response times for UAV resources access.

Edogawa	/temp	460	475	571	440	458	460	466	465	450	435	**468**
	/humidity	432	426	545	152	419	432	433	429	440	437	**414.5**
	/light/1	256	174	169	170	160	185	246	184	170	187	**190.1**
	/light/0	532	158	157	141	162	199	188	161	166	205	**206.9**
Mori	/lighting/1	166	170	187	200	467	208	167	167	166	160	**205.8**
	/lighting/0	288	285	301	227	316	322	173	280	337	218	**274.7**
	/spray/1	223	169	166	168	160	331	230	342	231	154	**217.4**
	/spray/0	159	238	155	664	190	364	235	196	227	291	**271.9**
	/spray/tank	161	363	156	142	279	153	163	185	152	161	**191.5**
	/spray/tank/full	167	155	180	216	200	156	176	173	159	193	**177.5**
Kudo	/temp	445	163	489	493	499	477	492	153	426	458	**409.5**
	/humidity	449	437	494	442	246	452	423	214	455	214	**382.6**
	/lighting/1	198	259	164	158	175	157	155	159	158	181	**176.4**
	/lighting/0	272	417	298	149	351	333	260	328	396	253	**305.7**
Hibara	/distance	161	145	201	201	173	165	174	228	196	252	**189.6**
	/lighting/1	158	232	175	193	174	156	153	179	170	254	**184.4**
	/lighting/0	272	378	371	397	199	259	191	155	215	188	**262.5**
												266.4117647

resources, and then returned the response of the requested service.

The response times were recorded ten times for each resource and the average was calculated as shown in Table 3 and Figure 6. The response time of requesting a UAV resource directly varies between 180 and 470 milliseconds with an average of 266 milliseconds. The variety of response time depends on the resource process; for example, the "LED on" resource is a simple digital output of LOW and HIGH, while the temperature sensor resource reads the analog voltage of the sensor pin, then converts it into voltage using a scale of 5, and then calculates the temperature value accordingly. This process requires more time compared to the digital output; therefore, the response time of the temperature request is higher than the response time of the LED. The implemented system was tested using the Postman Chrome extension (https://www.getpostman.com/docs/requests) for each device and resource. The test focuses on the pull data model of HTTP requests.

7. Conclusion

In this paper, we proposed a different way of monitoring and accessing UAVs' resources and services through the cloud. The traditional peer to peer communication and radio frequency transmission suffer from many limitations such as limited range of communication and difficulty in programing and developing new applications depending on the UAV's

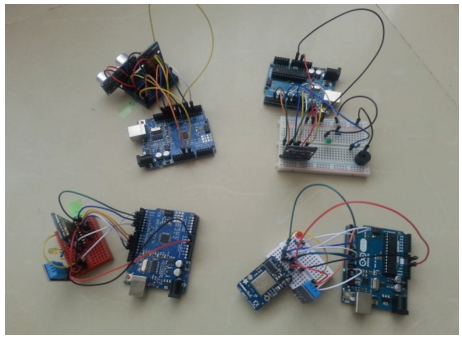

Figure 5: Four Arduino boards connected with Adafruit CC3000 boards as well as sensors and actuators representing UAV payload systems and their resources.

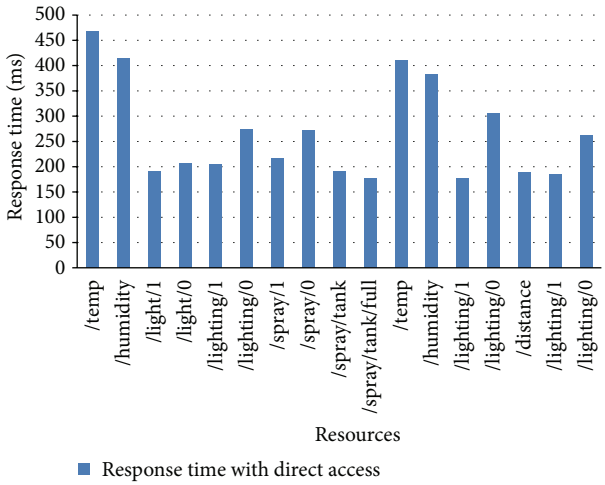

Figure 6: Response times of UAV resources with direct access.

language and commands. In addition, this approach does not support the heterogeneity of UAVs, where each UAV could have a different operating system and different command syntax and interfaces.

As a result, we proposed the Cloud Computing architecture to communicate with different UAVs, that is, a client-server architecture where the server provides its services to the client. However, in a UAV's environment, a UAV does provide not only services but also resources information such as the UAV's location and energy levels. This facilitates the monitoring task of multiple UAVs at the same time. The cloud supports two types of HTTP APIs.

One is the standardized SW* web service that is suitable for the digital services. It is considered unsuitable for the physical world and particularly for devices with limited resources. Also it is agreed that SOAP is heavyweight and difficult to learn. The other is the RESTful architecture which is a lightweight web service architecture and has numerous advantages such as its simplicity, ease of programing, being loosely coupled model of clients, and its readability for humans as well as the ease of parsing the JSON result into HTML or JavaScript.

These web services reside in the UAV's back-end server, and then the application in the front-end side can monitor and access the resources on the UAVs. The application could be either an AJAX model or a Comet model. In the Comet model the server pushes the feeds to the client whenever a change happens. Although this model seems suitable for monitoring, it has some limitations such as the programming language that supports this model. In addition, it limits the number of clients allowed to access the resources to those registered on the UAV's registry.

In the AJAX model, on the other hand, the client application receives the information of UAVs by refreshing the content without refreshing the page. This model transfers the load from the server side to the client side so that the client pulls the data from the server when needed. This model is suitable for UAVs because unlike traditional web servers, UAVs have very limited resources, while the client has much more resources and is capable of handling most of the required processing.

UAVs have a relatively well-defined set of services and models of access. However, building different applications that conform to how these services and devices interface is complex and involves huge efforts. Viewing UAVs as servers with well-defined services and resources accessible through a well-known set of interfaces helps resolve this issue. Therefore, integrating UAVs with the cloud offers several advantages and allows using the web tools for developing UAV servers and applications in a uniform manner. Accessing UAVs becomes as simple as accessing a web browser. Furthermore, it becomes more flexible and easier to build different applications that can take advantage of the services made available on the UAVs.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is supported in part by UAEU-NRF Research Grant number 3IT045 and UPAR Research Grant number G00001655.

References

[1] W. T. L. Teacy, J. Nie, S. McClean et al., "Collaborative sensing by unmanned aerial vehicles," in *Proceedings of the 3rd International Workshop on Agent Technology for Sensor Networks*, pp. 13–16, Budapest, Hungary, May 2009.

[2] O. K. Sahingoz, "Mobile networking with UAVs: opportunities and challenges," in *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS '13)*, pp. 933–941, IEEE, Atlanta, Ga, USA, May 2013.

[3] P. M. Mell and T. Grance, "The NIST definition of cloud computing," *National Institute of Standards and Technology*, vol. 53, no. 6, p. 50, 2009.

[4] 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.

[5] D. Zeng, S. Guo, and Z. Cheng, "The web of things: a survey," *Journal of Communications*, vol. 6, no. 6, pp. 424–438, 2011.

[6] S. Mahmoud and N. Mohamed, "Collaborative UAVs cloud," in *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS '14)*, pp. 365–373, Orlando, Fla, USA, May 2014.

[7] C. E. Lin, C.-R. Li, and Y.-H. Lai, "UAS cloud surveillance system," in *Proceedings of the 41st International Conference on Parallel Processing Workshops (ICPPW '12)*, pp. 173–178, Pittsburgh, Pa, USA, September 2012.

[8] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: state of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[9] S. Simanta, D. Plakosh, and E. Morris, "Web services for handheld tactical systems," in *Proceedings of the 5th IEEE International Systems Conference (SysCon '11)*, pp. 115–122, Montreal, Canada, April 2011.

[10] S. Stephen, N. Christian, and W. Scott, "Automated UAV-based video exploitation using service oriented architecture framework," in *Airborne Intelligence, Surveillance, Reconnaissance (ISR) Systems and Applications VIII, 80200Y*, vol. 8020 of *Proceedings of SPIE*, May 2011.

[11] D. Guinard, V. Trifa, and E. Wilde, "A resource oriented architecture for the web of things," in *Proceedings of the 2nd International Internet of Things Conference (IoT '10)*, Tokyo, Japan, December 2010.

[12] D. Guinard, V. M. Trifa, and E. Wilde, *Architecting a Mashable Open World Wide Web of Things*, ETH, Department of Computer Science, 2010.

[13] D. Guinard, V. Trifa, T. Pham, and O. Liechti, "Towards physical mashups in the web of things," in *Proceedings of the 6th International Conference on Networked Sensing Systems (INSS '09)*, pp. 196–199, IEEE, Pittsburgh, Pa, USA, June 2009.

[14] A. Rodriguez, *Restful Web Services: The Basics*, IBM DeveloperWorks, 2008.

[15] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle, "The web of things: interconnecting devices with high usability and performance," in *Proceedings of the International Conference on Embedded Software and Systems (ICESS '09)*, pp. 323–330, IEEE, Zhejiang, China, May 2009.