5-2015

UAV-CLOUD: A PLATFORM FOR UAV RESOURCES AND SERVICES ON THE CLOUD

Sara Yousif Mohamed Mahmoud

United Arab Emirates University

College of Information Technology

Software Development Track

UAV-CLOUD: A PLATFORM FOR UAV RESOURCES AND SERVICES ON THE CLOUD

Sara Yousif Mohamed Mahmoud

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Science in Software Engineering

Under the Supervision of Dr. Nader Mohamed

May 2015

Declaration of Original Work

I, Sara Yousif Mohamed Mahmoud, the undersigned, a graduate student at the United Arab Emirates University (UAEU), and the author of this thesis entitled "*UAV-Cloud: a platform for UAV resources and services on the cloud*", hereby, solemnly declare that this thesis is an original research work that has been done and prepared by me under the supervision of Dr. Nader Mohamed, in the College of Information Technology at UAEU. This work has not been previously formed as the basis for the award of any academic degree, diploma or a similar title at this or any other university. The materials borrowed from other sources and included in my thesis have been properly cited and acknowledged.

Student's Signature_____ Date _____

Approval of the Master Thesis

This Master Thesis is approved by the following Examining Committee Members:

1) Advisor (Committee Chair): Dr. Nader Mohamed
   Title: Associate Professor
   Department of Networking
   College of Information Technology

   Signature _____ Date _____

2) Member: Dr. Imad Jawhar
   Title: Associate Professor
   Department of Networking
   College of Information Technology

   Signature _____ Date _____

3) Member (External Examiner): Dr. Rabeb Mizouni
   Title: Assistant Professor
   Department of Computer Engineering
   Institution: Khalifa University

   Signature _____ Date _____

This Master Thesis is accepted by:

Dean of the College of Information Technology: Dr. Shayma Al Kobaisi

Signature _____ Date _____

Dean of the College of the Graduate Studies: Professor Nagi T. Wakim

Signature _____ Date _____

Copy ____ of ____

Abstract

UAVs - Unmanned Aerial Vehicles – have gained significant attention recently, due to the increasingly growing range of applications. However, developing collaborative UAV applications using traditional technologies in a tightly coupled design requires a great deal of development effort, time, and budget especially for heterogeneous UAVs. Moreover, monitoring and accessing UAV resources using traditional communication media suffer from several restrictions and limitations. This research aims to simplify the efforts, reduce the time, and lower the costs of developing collaborative applications for distributed heterogeneous UAVs. In addition, the research aims to provide ubiquitous UAV resources access. A platform is proposed for developing distributed UAVs. This platform provides services to simplify application development. In this approach, UAVs are integrated with the Cloud Computing paradigm to provide ubiquitous access to their resources and services. Due to the limited capabilities of UAVs, a lightweight architecture is adopted. UAV resources and services are modeled in a Resource Oriented Architecture which is a new flexible web service design pattern with loosely coupled interaction between services. Hence, they are accessed as Representational State Transfer RESTful services using HTTP. Moreover, the research proposes using a broker architecture to increase efficiency by separating responsibilities. Therefore, it separates the requester's logic and functionalities from the provider's. It also takes the responsibility for allocating the issued request to the available and suitable UAV(s). To test the proposed platform, I first developed the UAV resources as a payload subsystem then provided them with Internet connectivity. Then, resource identifiers and uniform interfaces were developed using the RESTful Application Programming Interfaces (APIs). I also developed the broker service along with a database containing the information of the registered UAVs and their resources. The platform system components were tested using a requester interface in a browser by issuing a request for a resource to the broker to find and request the service from a suitable UAV. The test was done for retrieving data from UAVs as well as requesting actions from them. The main contributions of this research are proposing the UAV-Cloud platform for simplifying the development of ubiquitous UAV applications and its

perspectives, as well as a lightweight loosely coupled design for UAV resources. Another contribution is developing the broker architecture for separating responsibilities in this platform.

Keywords: UAVs, Cloud Computing, distributed systems, broker, client-server architecture, Resource Oriented Architecture -ROA, Representational State Transfer-RESTful.

Title and Abstract (in Arabic)

الطائرات بدون طيار السحابية: منصة لموارد وخدمات الطائرات بدون طيار على الحوسبة السحابية

الملخص

الطائرات بدون طيار - اكتسبت اهتماما كبيرا في الآونة الأخيرة، نظرا لتزايد مجال التطبيقات. ولكن تطوير تطبيقات الطائرات بدون طيار التعاونية التي تبنى باستخدام التقنيات التقليدية في تصميم المهمات المقرونة بإحكام يحتاج إلى جهود ضخمة في والوقت، والتكلفة. وعلاوة على ذلك، الرصد والوصول إلى موارد الطائرات بدون طيار باستخدام الأساليب التقليدية تعاني من العديد من القيود والحدود. لذا يهدف هذا البحث إلى تبسيط الجهود وتقليل الوقت والتكلفة اللازمة لتطوير التطبيقات للطائرات بدون طيار غير متجانسة الموزعة. بالإضافة إلى ذلك، يهدف البحث إلى توفير الوصل إلى موارد الطائرات بدون طيار من كل مكان باستخدام الانترنت. في هذا البحث، تم اقتراح بنية برمجيات للطائرات بدون طيار. توفر هذه البنية الخدمات الأساسية بحيث يتم بناء تطبيقات الطائرات بدون طيار عليها بسهولة. في هذا النهج، تتكامل الطائرات بدون طيار مع نموذج الحوسبة السحابية لتوفير الوصول للطائرات بدون طيار من كل مكان. نظرا للقدرات المحدودة لهذه الطائرات، تم اعتماد برامج خفيفة للتحميل عليها. صممت موارد الطائرات بدون طيار على نموذج ROA وهو نموذج مرن لتصميم برامج النت مع خدمات خفيفة الارتباط باستخدام منافذ RESTful. وعلاوة على ذلك، يقترح البحث برنامج وسيط لزيادة الكفاءة عن طريق فصل المسؤوليات. وبالتالي، فإنه يفصل الجانب الطالب للخدمة من الجانب المزود ويأخذ على عاتقه توزيع الطلب نظرا للطائرات المتاحة والمناسبة. لاختبار الهيكل المقترح، بنيت أولا الموارد للطائرات بدون طيار من خلال توفير الاتصال بالإنترنت لهم وتطوير معرفات الموارد "واجهات موحدة باستخدام واجهات برمجة التطبيقات (API). ثم تم تطوير خدمة وسيط جنبا إلى جنب مع قاعدة بيانات لاحتواء المعلومات من الطائرات بدون طيار. تم اختبار الوسيط باستخدام واجهة الطالب في مستعرض بإعطاء طلب مورد للوسيط، بحيث يقوم باسترداد القيمة من الموارد المناسبة. وقد تم اختبار لاسترجاع البيانات من الطائرات بدون طيار وكذلك الطالبة الإجراءات منه. المساهمات الرئيسية لهذا البحث هي تصميم منصة لتطوير الطائرات بدون طيار كل مكان الموارد

والخدمات إلى خدمات الإنترنت المتباعدة خفيفة مريحة، مع تحديد الاعتبارات والمتطلبات، فضلا عن نموذج وسيط لفصل المسؤوليات.

Acknowledgements

I would like to thank my advisor Dr. Nader for his time and efforts. He believed in me and my abilities. Also, I would like to thank my committee (Dr. Imad Jawhar and Dr. Rabeb Mizouni) for their guidance, support, and assistance during the preparation of my thesis. Furthermore, I would like to thank the Dean (Dr. Shayma Al Kobaisi) and all members of the College of Information Technology at the United Arab Emirates University for assisting me throughout my studies and research.

I am highly indebted to Dr. Abdulmutalib whose expertise and enthusiasm about broker and Cloud Computing guided me through my research. I am especially grateful to Dr. Yacine who introduced me to the concept of Internet of Things and Web of Things, and whose support and encouragement led to this research and most of the other studies in which I have been involved.

My gratitude is extended to the staff of the Engineering Lab in particular Dr. Hassan Noura who allowed me to use their equipment and materials and also for their invaluable guidance. My thanks too go to Dr. Jameela Al-Jaroodi for her kind co-operation. I likewise offer my thanks to the Library Research Desk for providing me with the relevant reference material. Also, I must not forget to mention that this work is supported in part by UAEU-NRF Research grant number 3IT045.

I want to express my deepest appreciation to my parents and friends. I am sure they suspected it was endless. Lastly, I would like to to express my profound gratitude to Tasneem Amin my best friend, for her encouragement and motivation whenever I felt despondent. She was so supportive and gave me the inspiration to continue this journey.

Dedication

To my beloved parents and family

Table of Contents

List of Tables

List of Figures

List of Abbreviations

API	Application Programming Interface
CC	Cloud Computing
DBMS	Database Management System
ER	Entity-Relationship diagram
FANET	Flying Ad Hoc Networks
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IoT	Internet of Things
IP	Internet Protocol
LED	Light-Emitting Diode
MANET	Mobile Ad-hoc Network
PaaS	Platform as a Service
QoS	Quality of Service
REST	Representational State Transfer
RF	Radio Frequency
ROA	Resource Oriented Architecture
RoI	Region of Interest
SaaS	Software as a Service
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol

UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
VANET	Vehicular Ad-hoc Networks
WoT	Web of Things
WSDL	Web Service Definition Language

Chapter 1: Introduction

This chapter gives a brief overview about this research. After that the problem statement is presented and the main objectives of this research are illustrated, followed by the scope that this research covers. Finally, the thesis outline is stated.

1.1 Overview

Unmanned Aerial Vehicles (UAVs) are aircraft without human pilots on board. UAVs are remotely controlled from the ground or autonomously by an on-board computer. A recent study estimated that in 2017, the civilian UAV market in the United States alone could reach $560 million out of a total of around $5 billion [1]. With recent advances in airframe, control, and communication technologies offered in UAVs, manned operations for many applications can be efficiently replaced with UAVs. UAVs have the potential to perform various important and repetitive tasks; they can do this in an automated efficient way. This is mainly a consequence of their high accuracy, mobility, and repeatability levels [2].

UAVs can be very useful in agriculture for spraying pesticides or seeds; in search and rescue operations in disaster areas; for capturing large areas for security and surveillance; in environmental monitoring; for large infrastructure monitoring; and in terrain mapping applications. Such tasks require repetitive, hazardous and/or tedious tasks. Although manned aerial vehicles can be used, such utilization requires long hours of repetitive, highly focused, and costly flights that place a heavy burden and high risk on pilots.

As they need to rely on some form of Radio Frequency (RF) communication, UAV applications need to establish direct links among themselves and with the ground station(s). Such links may either be single links or multiple hops through other communication nodes that may be other UAVs nearby or some intermediate ground stations [3]. However, this peer-to-peer RF communication between ground stations and UAVs is not suitable for many of the UAVs' dynamic distributed and heterogeneous operating environments.

Some UAV missions involve multiple UAVs working together to quickly achieve a specific task [4]. However, controlling and utilizing multiple UAVs that will effectively and concurrently operate and coordinate them for a certain problem area requires a huge number of man hours in design, development and testing [5]. This is mainly due to the lack of technologies that can be utilized to effectively coordinate the operations of multiple UAVs. Moreover, a collaborative mission usually consists of multiple tasks that are executed sequentially or concurrently by multiple UAVs to accomplish the mission. These tasks are allocated to UAVs and monitored by either a ground station [6][7], or autonomously [8][9]. Developing such missions is time consuming and costly due to the heterogeneity of UAVs' resources and systems.

The aim of this research is to provide ubiquitous UAV resources and services access through the cloud. As well as separating responsibilities for more efficient architecture, this eases the development of new client applications without the repetitive efforts for heterogeneous UAVs development.

The approach adopted in this research study is to utilize the Resource-Oriented Architecture (ROA) model by providing the UAV's resources and capabilities to other requesters through Application Programming Interface (APIs).

The ROA is a client-server architecture implemented in Representational State Transfer (REST) architecture. However, for distributed UAVs we utilize the broker architecture pattern for more efficient and scalable applications. Here, UAVs register their services and resources in the broker. Then, the requester sends the request of a service to the broker which allocates the appropriate available UAV that can perform the service. This model is implemented using the Cloud Computing (CC) paradigm. By integrating UAVs to the cloud, UAVs are accessed ubiquitously as cloud resources. CC has been expanded not only for computers and mobile devices but also for embedded systems [10]. Similarly, UAVs have embedded systems that conform to the concept of Internet of Things (IoT) [11] and Web of Things (WoT) [12], so that they can be connected to the Internet to be accessed and monitored through the Web. For example, a client application can monitor a mission's 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 the camera, sensors, and actuators using web services' protocols.

The contribution of this research includes: integrating UAVs not only to the Internet but also to the cloud computing paradigm that provides resources and services as a shared pool. In addition, the research proposes a UAV-Cloud platform for distributed UAVs. This platform focuses on their resources and services of the payload system. These resources are designed in a lightweight flexible ROA, providing APIs for each resource in a loosely coupled architecture to support reusability. Furthermore, a broker layer with a database is developed on the cloud for separating responsibilities to separate the UAV side from the client side.

The research assumes that UAVs are autonomous; hence, the control subsystem is responsible for the navigation process to the required location. In

addition, the user mission is assumed to be decomposed into set of tasks, where each task could be assigned to a UAV. Furthermore, the research assumes the availability of reliable network connections between the UAVs and the cloud. This is a valid assumption especially for environment such as smart cities.

The research is evaluated by developing a prototype using Arduino devices as an UAV payload subsystem with a Wi-Fi shield for internet connectivity. Each UAV will be considered as a server that provides its resources and services to be accessed through defined RESTful APIs. Then a broker will be implemented by NodeJS platform using JavaScript programming language. The UAVs will register their services, capabilities and identifications to the broker, so that the broker stores the information in a database that contains the data of the registered UAVs.

For the purpose of simplicity, the requester will be a RESTful requester plug-in on the browser. The request of a certain service will be sent to the broker, then the broker allocates the services to the available suitable UAV that matches the request considerations.

1.2 Statement of the Problem

At this point in time most UAVs rely on radio frequency communication. Most typical UAV operations need to establish direct links among themselves and with the ground station(s) through certain frequencies that both transmitter and receiver are tuned to. However, peer-to-peer communication and radio frequency transmission suffer from many restrictions such as a narrow range of communication which depends on the transmission frequency. Although the covered communication area increases proportionally with the transmission frequency, it leads to a high consumption of the limited UAV's energy source more rapidly for the transmission.

Moreover, the transmitter and receiver should be tuned to the same frequency to be able to communicate. Also, such common set-up systems suffer from the difficulty of programming and developing new applications depending on the UAV's 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. It restricts the location of the ground station to the mission's location and requires UAVs to be in direct line-of-sight (LOS) of the ground station to maintain communication and control. In addition, the control and monitoring of UAV missions have become more complicated and are limited to the specific devices that UAVs are connected to.

Most collaborative UAV missions' developments face difficulties when dealing with heterogeneous UAVs, where they have different resources, commands and operating systems. Therefore, the development of UAVs is specified for a certain mission and re-developed for each different mission using the same UAVs. That is due to the tightly coupled design of UAVs functionalities.

Due to the difficulty of task allocation for UAVs, tasks require specific resources that are available in some UAVs with specific conditions such as energy level and location. Moreover, real time monitoring for these UAVs throughout the mission is a difficult process for a human.

1.3 Objectives

This research aims to provide UAV platform architecture for developing ubiquitous UAV applications based on separating responsibilities, where UAVs are responsible for providing their resources and services to the given requester, while another layer- the broker- is responsible for monitoring and registering UAVs which

will be able to allocate the suitable UAV for requests. As a result, the requester of a service does not need to know the UAV services' providers and their resources. This facilitates the development of new missions because of the loosely coupled services and the responsibility separation.

Moreover, developing new applications becomes an easy process regardless of the heterogeneity of the UAV system and commands. Accordingly, adding UAVs becomes as easy as plug and play. This research proposes a broker architecture that keeps records of the registered UAVs and their services and resources with up-to-date information of the dynamic UAVs. This architecture is built in the cloud computing paradigm. In addition, it utilizes the cloud resources and the ubiquitous service, so that UAVs are accessed regardless the location of the user.

1.4 Scope

The scope of this research is the UAV payload subsystem that is, UAV resources and services. The research focus is to integrate UAVs to cloud computing paradigm and model their resources and services as Resources-Oriented Architecture that is implemented using RESTful web services. Resources and services are loosely coupled where there is no direct relation between them. They are utilized using broker architecture to separate the requester from the provider. The broker reserves the information of the registered UAVs in the database, and allocates the resource request to the appropriate UAV.

Decomposition of the mission from user requests into multiple assignable tasks is out of the scope of this research. Therefore, for testing purposes, tasks are simulated as an external request from a simple browser application. In addition,

controlling aspects and flight issues are beyond this research. The service assumes that the flight subsystem gets the destination parameters without specifying directions or path planning.

1.5 Thesis Outline

The rest of the thesis is organized as the following;

Chapter 2 is the literature review which examines briefly UAVs' missions and their importance. It then focuses on previous efforts toward multi-UAV communication and architectures. This is followed by a discussion of the IoT smart objects and their available platforms.

Chapter 3 proposes the Framework of UAV Cloud Computing starting by defining the layers of the framework and then determining the user types of the system. After that, the opportunities gained from this system are illustrated. Then a discussion of the technical considerations is specified for both the UAVs side and the platform. Finally, the components of the platform are detailed.

Chapter 4 focuses on designing the platform APIs using the ROA architecture that is implemented as RESTful HTTP. The design begins by defining the UAV resource types along with their APIs, then the database model for storing the UAV information as well as the operation information. After that, the broker's APIs design which allowed the interaction of both UAVs and application developers is discussed. Finally, a brief description of the user application is addressed.

Chapter 5 illustrates the implementation and testing of the architecture system components. The implementation includes the UAV side as well as the broker side. The UAV side is implemented as Arduino boards with connected sensors and LEDs

as resources and services accessed by their APIs. This is followed by the implementation of the broker layer in NodeJS connected to PostgreSQL database, the broker offers APIs to access UAVs through it.

Finally, Chapter 6 summarizes the research and makes suggestions for future area of further researches.

Chapter 2: Literature Review

This chapter begins by examining UAVs and highlighting the motivation for their usage and importance. Then, a summary is presented about UAV communication types and their limitations. This is followed by an examination of multi-UAV monitoring architecture, which focuses on previous efforts toward UAVs middleware and cloud computing. After that, a similar field of smart objects and IoT are discussed as my research is built upon this concept.

2.1 Motivation to UAVs and their Usages

UAVs are systems that include many subsystems such as flight and control, communication as well as payloads. They vary in size from High Altitude Long Endurance to Nano Air vehicles, with different speed capabilities and types of missions. Although UAVs have been known in military missions, they have recently been introduced into civilian missions and have had a great impact on the environment [13].

Most civilian missions use small UAVs that have limited capabilities and payloads. A UAV may have one or more payloads such as sensors and actuators. Sensors collect data from the environment, while actuators perform actions on the environment.

There are many applications for UAVs such as the example presented by Varela et al. [14], where UAVs are used for environmental monitoring such as collecting data on air quality in different layers of the atmosphere as some information cannot be collected by ground systems due to gasses or smoke from

fires. The main missions of these UAVs were to measure pollution and locate its sources. The swarm intelligence based strategy can be used as it uses a completely distributed approach. Another example, Fausto et al. in [15] proposed architecture for using UAVs and Wireless Sensor Network (WSN) in agriculture applications. Fausto et al. developed a collaborative UAVs system to spray pesticides and fertilizers in agricultural areas that can hardly be reached by humans efficiently without missing some areas in the spraying process, duplicating spraying areas or spraying outside boundaries.

Furthermore, Chmaj and Selvaraj [4] addressed a survey about collaborative and distributed UAV applications. They presented several applications, such as; object detection and tracking, where UAVs search and allocate a specific object then track it using a swarm of UAVs that communicate with each other. Surveillance is one of the most famous applications in UAVs, where multiple UAVs are distributed to monitor a large area. Another important application is data collection through WSN. This includes ground sensors as well as UAV sensors. Collected data can then be sent to the ground station to be monitored and analyzed. Environmental monitoring used to detect forest fires, storm and pollution has gained high interest in UAV applications.

Mohammed et al. [16] referred to UAV applications for smart cities. They addressed safety applications such as traffic and crowd management as well as urban security especially for big public events. They also discussed the business applications of UAVs such as in Amazon Prime Air for delivering products and their use for restaurant services. Also they proposed the development of UAVs in Dubai for small lightweight items delivery as well as documents and medicine [17].

These various application opportunities of UAVs have encouraged researchers and developers to focus on improving efficient frameworks to develop UAV applications easily, especially for multiple distributed UAVs that cooperate with each other. Therefore, they have developed different architectures and communication protocols for collaborative UAVs.

2.2 UAV Specifications

UAVs vary in size and specifications of their software and hardware according to their category. Categories depend on the communication range, UAV mass as well as their usages [18]. UAVs are categorized as shown in Table 2-1.

Table 2-1 UAVs categories according to mass, flight altitude, range of communication and endurance [19].

Category name	Mass [kg]	Range [km]	Flight Altitude [m]	Endurance [hours]
Micro	< 5	< 10	250	1
Mini	<25/30/ 150	< 10	150/250/ 300	< 2
Close Range	25 –150	10 – 30	3000	2 – 4
Medium Range	50 –250	30 – 70	3000	3 – 6
High Alt. Long Endurance	> 250	> 70	> 3000	> 6

However, most civil applications use only micro and mini UAVs. These categories have a limited endurance up to 2 hours due to their limited power supply. Furthermore, they fly in low altitudes with a short communication range not exceeding 10 kilometers. In such categories, the UAV is capable of carrying limited weight which restricts the hardware resources into certain boundaries.

Chao et. al. [20] compared the physical specifications of small UAVs, shown in Table 2-2. The comparison shows the limited processing and memory of UAVs. In addition, most of these resources are consumed for controlling, navigation and communication processes.

Table 2-2 A comparison of physical specifications of autopilots [20].

	Size (cm)	Weight (g) w/o radio	Power Consumption	Price (k USD)	DC In (V)	CPU	Memory (K)
Kestrel 2.2	5.08*3.5*1.2	16.7	500mA(3.3 or 5V)	5	6-16.5	29MHz	512
MP 2028[g]	10*4*1.5	28	140mA@6.5V	5.5	4.2-26	3MIPS	-
Piccolo LT(w.modem)	13*5.9*1.9	45	4W	-	4.8-24	40MHz	448
Unav 3500	10.16*5.08*2.03	42.45	100mA@6V	3/5(FW/HL)	5-7	40MIPS	256

Moreover, Chao indicated that open source UAV designed in Linux is useful for researchers to add and modify the source code and add their hardware. The industry provides open source hardware in which the developer has the freedom to design and program systems. Arduino and Raspberry Pi are the mostly used open source hardware. A comparison of these devices is shown in Table 2-3.

Table 2-3 A comparison of open hardware devices.

	Devices			
Platform	**Arduino**	**Propeller**	**Beagle Board**	**Raspberry Pi**
Variant	Uno	PropStick	Rev. C4	Model-B
Software				
Operating System	-	-	Android, Linux, Windows CE, RISC OS	Linux, RISC OS
Dev. Envrionments / Toolkits	Arduino IDE, Eclipse	Propeller/Spin	Eclipse, Android ADK, Scratchbox	OpenEmbedded, QEMU, Scratchbox, Eclipse
Programming Language	Wiring-based (~C++)	Spin / Propeller Assembly	Python, C, etc.	Python, C, possibly BASIC
Architecture	8Bit	32Bit	32Bit	32Bit
Hardware				
Processor	ATMEGA328	P8X32A-M44	TI DM3730 (ARM)	BCM2835 (ARM)
Speed	16Mhz	20kHz/12Mhz (Internal) or 4-8Mhz external	720Mhz	700Mhz
RAM	2Kbyte	32Kbyte	256MB	256MB
ROM	32Kbyte	32Kbyte	256MB Flash	SD
I/O (various protocols)	14	32	22 (on expansion header)	8
ADC	6	-	internally used	internally used
USB	-	-	1 x 2.0	2 x 2.0
Audio	-	-	Stereo In/Out	Stereo Out, In w/ USB mic
Video	-	VGA, NTSC or PAL	DVI-D, S-Video	HDMI, NTSC or PAL
Misc.	Many shields available for added capability	8 processors for parallel tasking	SD/MMC, RS-232, JTAG, USB OTG, LCD	SD, 10/100 Ethernet, JTAG
Cost				
	$29.95	$49.99	$199.95	$35.00

2.3 Radio Frequency (RF) Communication in UAVs

One of the main technical requirements of UAVs is the availability of communication facilities among them. A lot of research has been done on traditional radio communication. In [3] a Flying Ad Hoc Network (FANET) model was designed for UAVs. This model differs from traditional networks, Mobile Ad-hoc Networks (MANETs) and Vehicular Ad-hoc Networks (VANET) in terms of connectivity and routing capabilities.

The main challenge facing FANET is routing as the network topology changes dynamically and rapidly. UAV communications can be either UAV-to-UAV communication where UAVs communicate with each other or UAV-to-Infrastructure communication where UAVs communicate with fixed infrastructure locations such as ground stations. A MANET uses mobile nodes in random network topology that changes rapidly; therefore, it can be used in UAV FANET to make routing easier and to improve the performance of wireless communication systems. To increase FANET communication performance, transmission power needs to be decreased by communicating with the closer UAVs. As a result, MANET routing mechanisms are preferred in FANET but they are not directly applicable.

However, this short range peer-to-peer communication is not suitable for many of the UAV 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 to maintain communication and control. In addition, the control and monitoring of UAV applications become more complicated and limited to the specific devices that UAVs are connected to.

2.4 Collaborative UAV Architectures

Using multiple UAVs collaborating together decreases the time needed to achieve specific tasks. However, developing such applications for UAVs with heterogeneous devices; different energy levels, varying storage, communication, sensing and processing capabilities is a complex task [21]. Collaborative UAVs can be homogenous or heterogeneous in their communication, acting, sensing, storage, and processing capabilities as well as their energy levels. Although applications that rely on homogenous UAVs are easier to develop, heterogeneous UAVs can offer great opportunities for providing cost-effective solutions for complex applications that require different capabilities for the various tasks involved.

According to Mohamed's et al. work [22][23], there are six aspects of multiple UAVs collaboration; (1) collaborative sensing using distributed sensors; (2) collaborative acting to cover large areas faster; (3) collaborative communication to allow UAVs to interact with each other; (4) collaborative data processing which allows UAVs to process large data among the UAVs that have on-board high performance computers; (5) collaborative storage that organizes data storage among multiple UAVs depending on their capabilities; and (6) collaborative control of distributed components to achieve one goal.

2.4.1 Distributed Self-Allocation Architecture for Collaborative UAVs

In distributed collaborative UAV missions, a UAV interacts with all other UAVs to find the required service provider, and then interacts with it to request and get the service. In this scenario, all UAVs communicate to allocate tasks as specified in [6], [22], [24] and [25]. Following this approach, the mission is divided into tasks

and distributed to all UAVs then each UAV chooses a suitable task for itself. Next, they negotiate to ensure that all tasks are allocated to UAVs and no task assignment duplications. After that UAVs exchange messages to execute tasks in the right order.

When a UAV requires data or a service from another UAV it sends requests to all other UAVs, then the suitable UAV that provides that service replies to the requester UAV; next they exchange messages to complete the service. Another method is to broadcast information, where each UAV broadcasts its services and status to other UAVs such that the requester knows others' services and only sends the request to the provider UAV rather than broadcasting its request.

The self-allocation algorithm for a set of tasks was implemented in [6] for four UAVs and showed a conflict in allocating a task when having two UAVs had almost identical resources and capabilities. This showed the inefficiency of the algorithm for long collaborative service lengths and large numbers of UAVs.

The distributed self-allocation approach has many challenges especially for a situation where there is a large number of UAVs. This is because it consumes more energy in communications and negotiation for finding and requesting a service as well as updating all UAVs with new parameters, since each UAV needs to interact with all of the other UAVs. Also in such a scenario a lot of memory is used in UAVs to save the data of the services and information about other UAVs such as their energy level and locations. Furthermore, in case of re-planning a mission, UAVs interact with each other for rescheduling. This all leads to high communication traffic in collaboration, especially in the case of a mission with a huge number of UAVs.

2.4.2 Previous Efforts Toward Collaborative UAVs Middleware

Collaborative UAVs can be homogeneous or heterogeneous in their operating systems, commands, communication, acting, sensing, storage, and processing capabilities as well as their energy levels. While applications that rely on homogeneous UAVs are easier to develop, heterogeneous UAVs can offer great opportunities for providing cost-effective solutions for complex applications that require different capabilities for the various tasks involved. However, developing such applications for UAVs with heterogeneous devices, different energy levels, and varying storage, communication, sensing and processing capabilities is a complex task without middleware [21]. Middleware is the software layer composed of a set of services and functions to connect different components of a distributed system. It separates the operating system from the application side.

Distributed UAVs applications development, deployment, operations, and management are generally very complex tasks. One proposed approach to overcome these difficulties is to follow the Service-Oriented Architecture (SOA) [23] [26].

Earlier, de Freitas et al. [27] studied the UAVs sensing network specifically for surveillance applications through middleware. In surveillance applications, UAVs cooperate with ground nodes to cover the surveillance area. de Freitas et al. focused on providing an intelligent communication between: (a) UAVs and the ground station, (b) UAVs and ground nodes and (c) among each other, taking into account the limited resources and capabilities of UAVs. First, de Freitas et al. proposed breaking down the mission into a set of sub-missions that can be allocated to individual nodes. These sub-missions run over middleware. de Freitas et al. detailed the three layers of the middleware: At the bottom, the Infrastructure layer, in which

all hardware and resources are managed by the operating system. Then, the Common Services Layer, that are common in different applications, regardless of the mission such as networking management. Finally, the top layer, the Domain-Services Layer, to support application services according the domain, nevertheless, it can be reused among different applications. A minimal set of middleware services called a kernel was installed in UAVs and nodes to perform the basic services that support UAVs. Simulation results were provided to measure the efficiency of the proposed middleware. The simulation showed the distribution of nodes and the selected ones for mission. While their simulation demonstrated the number of engaged nodes, it did not show the discovery method and how to integrate them. In summary, there was no clear selection process or allocation approach.

The SOA model proposed in Mohamed's et al. [22] is based on the concept that every UAV has a global view of all other UAVs; however, it was reported that this concept has a poor scalability. As a result, Mohamed et al. discussed having a broker service in each UAV to maintain other UAVs' information regarding their services, capabilities, location, power level and other details. Then UAVs exchange their information through advertising and requests. Requests are invocations from the consumer to the provider to get a specific service. Mohamed et al. categorized invocation services into synchronous service and asynchronous service. The former maintains an active connection between the requester and provider until the provider returns a result. While in the latter, the connection may be terminated after the request is sent, then another connection is established when the provider responds back, which is more efficient in instances where the connectivity is unreliable. Finally, the Service-Oriented Middleware (SOM) services are integrated to develop

collaborative services so that applications can be reused without the need to implement them from scratch for every application [8].

2.4.3 Previous Efforts Toward Collaborative UAVs Cloud

Cloud computing is a new paradigm for hosting and delivering services over the Internet. Some research has been carried out to utilize the Cloud for some UAV applications. Chin et al. proposed connecting a UAV to cloud services such as Google Earth [28]. 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. However, the authors demonstrated the system for a single UAV, 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 suffers from inconsistent data.

Simanta et al. [29] developed four prototypes using the SOA and smartphones. The concept started by implementing a service that transmits Motion JPEG images from a wireless camera to a smartphone via TCP/IP. Due to the TCP delay, reimplementation was done using User Datagram Protocol (UDP). The first prototype was a UAV that tracked a vehicle and sent images as Simple Object Access Protocol (SOAP)-over UDP to a smartphone. In the second prototype smartphones were connected to the vehicle that sent messages to a fixed station as well as a UAV that transmitted video feed back to the station. The third prototype sent messages to both local and remote service consumers. The foruth prototype

focused on the video performance that was affected by the message overload due to its high size by using a binary format instead of SOAP.

Video Exploitation Tools is another example of a SOA application for UAVs as implemented by Se et al. [30]. 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 therefore it may not always be possible to achieve real-time interactions.

2.5 Cloud Computing for Smart Objects

On the other hand, smart objects such as sensors, actuators, and embedded devices are connected to the Internet through the IoT [31]. The main focus of IoT is establishing network connectivity between smart objects and the Internet, while the WoT builds the application layer on top of the network [32]. Accordingly, the Web tools and protocols can be used for developing and interacting with these objects.

Some efforts have 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, Guinard et al. [33] proposed the REST architecture by defining an object as a server that provides its resources in a ROA. Guinard et al. used the web tools as a solution for the WoT. Guinard et al. proposed two methods for accessing objects [34]. First, they connected devices to a smart gateway for measuring power consumption. The smart gateway is a web server that provides its resources for the clients to monitor and control electrical devices. In this approach, objects that have no direct Internet connectivity are connected to the smart gateway

through other protocols such as Bluetooth and ZigBee. This architecture allows Internet access to those devices through the smart gateway as well as calculating the overall consumption of all devices connected to it. The second method is a direct access to wireless sensor networks, where each node is considered as a web server that has a uniform interface that the client applications access.

According to the literature, IoT lacks standardizations and there is no commonly accepted layer architecture [35]. Therefore, there is a wide variety of platforms on the market. For example, Xively platform is one of the earliest IoT platforms [36]. It allows users to register their devices and monitor them using API keys. Another example is DeviceHive [37] that provides a common set of RESTful web services APIs for access from clients and devices. Also, 52North's Sensor Web provides access to sensor data encoded in SensorML [38]. The platform offers sensor registration, inserting observation and marking queries. Furthermore, ThingWorx is an application development platform with tools for model driven development of IoT applications [39]. It provides data models for storing devices' data and semantic query/ search.

My research is built upon these approaches and proposes a platform with a broker architecture for the UAV resources in a ROA implemented in a RESTful web service on cloud computing.

Chapter 3: UAV-Cloud Framework

This chapter presents a UAV framework on the cloud computing paradigm to enable the development of distributed UAV operations. Cloud computing has been expanded to include not only powerful computers and servers but also objects and embedded systems. Integrating smart objects to the Internet is IoT, while providing its resources and services is WoT. UAVs with limited capabilities and resources such as battery capacity, data processing and storage, may use the cloud resources for application development for distributed UAVs. As a result, UAVs do not need to be equipped with powerful capabilities and can be heterogeneous in their operating systems and resources, so that using this technology with standard communication protocols reduces the total time and cost of application development. UAVs can use the cloud's powerful services and resources while the cloud applications can use UAVs as a real world resource and service provider. Missions and task-allocation to UAVs depend highly on their locations and capabilities. Thus, Cloud Computing could provide a platform to manage mission planning and brokering services, while UAVs offer specialized services that are related to the physical world for certain tasks such as sensing and acting. This separation of responsibilities for each entity reduces the efforts needed to develop new applications on top of this platform. In addition, it allows the addition of more UAVs as plug-and-play to the system.

3.1 UAV-Cloud Framework Layers

Cloud Computing, one of the major IT revolutions, is defined as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with

minimal management effort or service provider interaction. This model can be used in UAVs to increase proficiencies and efficiency by collaborative UAVs. Cloud Computing consists of three service models: Infrastructure as a Service (IaaS) that includes hardware, virtual machines, storage, networks, and firewalls, then Platform as a Service (PaaS) to provide a set of APIs for functions for programmatic platform management and solution development, and finally, Software as a Service (SaaS) which is an online software application. UAVs can be mapped to Cloud Computing models to combine UAV resources with cloud features. The framework of the UAV-Cloud is shown in Figure 3-1.

3.1.1 UAV IaaS

First, the IaaS model includes UAVs and other components. UAVs' components include their payloads, sensors, actuators, internal memory, processor and other resources. Other components are any external entities that could provide resources or services such as ground node sensors or objects connected to the cloud, or the cloud computing resources such as storage servers and high performance servers and processors. These are managed through APIs to the PaaS.

3.1.2 UAV PaaS

Second, the PaaS is modeled as middleware to isolate the infrastructure layer from the application layer. It offers resources as services to the application layer. PaaS allows integrating cloud services with UAV services to implement powerful UAV applications. The platform includes UAV resources and services as well as cloud services such as collaborative services for mission planning and organizing resources. The development of collaborative UAVs implies the development of three

main decision-making abilities: mission planning, task-allocation, and coordinated task achievement [40][41]. In the proposed architecture, the Mission Planner is the service responsible for dividing the user mission from its application into a set of tasks, and then the Task Requester service coordinates these tasks by requesting a service from the broker according to the tasks' order. The broker is responsible for registering UAVs and it reserves their data in a Database Management System (DBMS). After that it allocates the requested task to the suitable available UAV. UAV resources and services offer specific data from sensors, or perform an action using certain actuators, for example, getting a temperature sensor or a gas sensor from UAVs or performing pesticide spraying and image or video capturing.

3.1.3 UAV SaaS

Third, SaaS is a lightweight software application available online and built on top of the PaaS through standard APIs. The developers implement applications for users to request certain UAV missions, for example, software that requests UAVs for spraying crops for a specific agriculture area. The user accesses the application to specify the location and size of the land then requests crop spraying by UAVs. It also offers monitoring interfaces for the user to follow up the progress and completion of the mission. Then, the collaborative services in PaaS manage the mission planning, scheduling and task allocation to suitable UAVs according to their statuses and resources such as cameras for monitoring, GPS for location, and fertilizer/pesticide tanks for crop spraying. These services are available by PaaS and are accessed through APIs. Another example is surveying forests to find the source of a fire. This mission is established and monitored through another software application that could use the same set of UAVs. Therefore, Collaboration Services are responsible for

allocating the suitable available UAVs with gas sensors and cameras to the surveying mission and managing the spread of UAVs over the forest to ensure they are covering the whole area efficiently. Then UAVs use customized services to sense temperatures, capture photos, update status and invoke other services requiring real-time information. These applications can be built easily on top of the PaaS for the same UAVs due to the separating of responsibilities of entities.

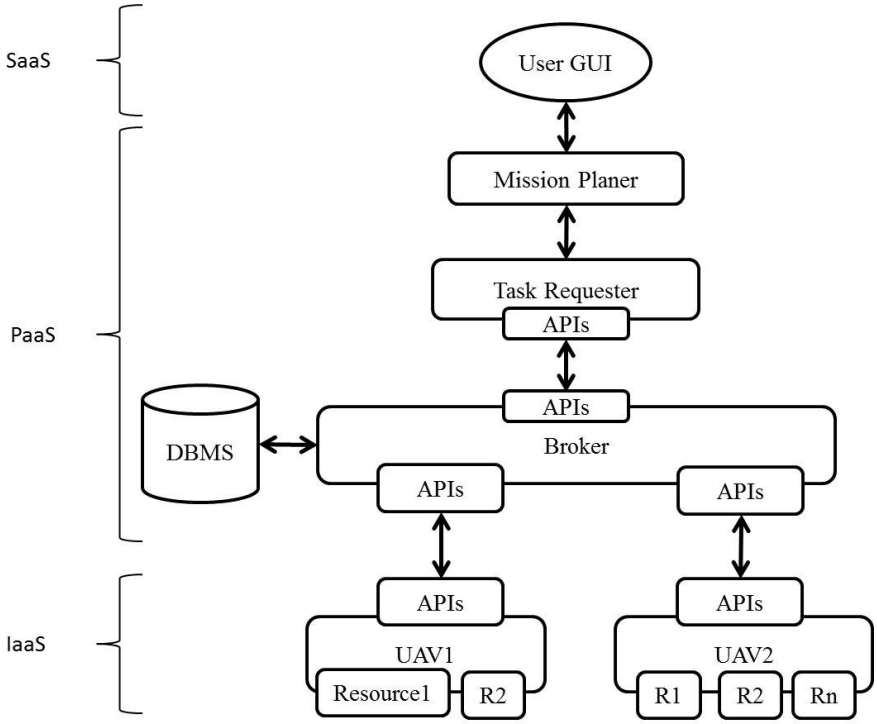


Figure 3-1 UAV-Cloud Framework

3.2 UAV-Cloud User Types

There are four types of UAV-Cloud framework users; End Users, Application Developers, UAV Providers and Administrators. These users access the Cloud Computing through APIs and identifications depending on the privileges given to each user.

3.2.1 End Users

These are the SaaS application users who establish the UAV mission. The end user accesses online application software through the browser to request the mission giving specific service parameters. The results and feedback are displayed in a user-friendly interface to the user with certain interaction capabilities. This application software is built by the application developer.

3.2.2 Application Developers

They develop the SaaS for the end users on top of the PaaS. The developers register to the platform to be authorized to access its services and APIs to develop new applications. The developers use the platform resources and services to integrate them through their APIs using the pre-defined formats and interaction protocols in order to build the application. Therefore, the developer defines the mission requirements and the UAV services required to perform that service. Also, the developer defines the parameters that the end user should specify to request the mission.

3.2.3 UAV Providers

These are the owners of the UAV who register them to the platform so that they can be accessed and used by the application developer for certain missions. The provided UAVs define their APIs according to standard interfaces, also they use the platform API to push their data and access the platform. The registered UAVs become part of the UAV cloud IaaS along with APIs to the platform.

3.2.4 Administrators

The administrators are the platform owners. They keep track of other users and resources. They operate and maintain the cloud services and UAVs. They use tools and APIs to manage and monitor the platform.

3.3 Opportunities of UAV-Cloud

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 of UAV data computations could be made on the cloud rather than in UAVs which reduces the UAV consumption of power and processing. Moreover, Cloud Computing provides large and scalable storage services that can be used rather than the limited UAV storage. As a result, storing data in the cloud increases reliability by ensuring data back-up thus offering access to previous log data even when the UAV is out of service. Cloud Computing provides ubiquitous services such as Google Earth 3D maps and computations that can be integrated with the UAV services to develop 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 application development easier for building heterogeneous UAVs in different programming languages that are used in web applications. Not only that, but also the standardized protocols affords the ability to integrate other nodes and components that use the same standards as the UAV application such as ground nodes and WNS. 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 in the run time of the operation. 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.

3.4 Considerations of UAV-Cloud

Although collaborative UAVs Cloud offers several opportunities for UAV operations and development, there are a number of considerations that must be taken into account for the UAV-Cloud framework. These considerations include UAV and platform development issues:

3.4.1 UAV 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.

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, i.e. 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.

3.4.2 Platform Development Considerations

On the other hand, there are several considerations in developing the UAV-Cloud platform. The platform should provide the ability to register UAVs and reserve information of their resources and services as well as the uniform interface to invoke them. This registration service facilitates the addition of UAVs to the platform. Furthermore, the platform is required to be scalable to large numbers of UAVs and should manage their distribution in real time simultaneously. Also, as the platform is responsible for integrating heterogeneous UAVs as well as cloud services, it should include services for (a) mission planning that divides the user's mission into sub-tasks to be executed sequentially, (b) decision-making of

performing services depending on the collected data of the environment, and (c) allocating tasks to the suitable and available UAV according to certain parameters. Additionally, the platform is responsible for tracking and monitoring UAV resources and their execution throughout the mission to ensure the efficiency of the operation. Moreover, UAVs collect a huge amount of data from the environment. These data should be stored in data stores and analyzed to support and enhance decision-making. Another consideration is security and privacy of data and resources. Data security is one of the important considerations in UAVs as the data could be critical and/or confidential, particularly if it is a military or political mission. The data should be secured such that only users with authorization can access it. Encryption and decryption processes can be used in data exchange. Other security mechanisms are required for data and resources protection. Also, user access such as establishing or canceling a mission could be authenticated by certain users under specific conditions, so that only authorized clients can control UAVs. In addition, for platform security issues, it authentication mechanism should be provided so that only registered and verified developers can access the platform services.

Another consideration is multi-tenancy, where users access the same set of UAVs. However, in a UAV environment, these UAVs are physical entities that perform real world operations. Therefore, the same resource cannot be used by multiple users at the same time. Nevertheless, they can be reused after a UAV has accomplished its operation. As a result, the platform separates the data and resources by having an operation database for each user to manage the assigned resources.

3.5 UAV-Cloud Platform Components

The focus of this research is the UAV-Cloud platform layer by integrating UAVs to the cloud and providing an efficient platform to build applications on top of it. In traditional development, applications are developed for specific hardware or systems and this usually means implementing all the component systems needed. This approach is inefficient and time and effort consuming. However, these components can be developed as services and integrated in the applications when needed. Services includes Collaborative Services that are required for any type of collaborative UAVs and UAV Services that are offered and used based on the UAV capabilities. Building applications on top of these services reduces the time and cost of developing collaborative UAV applications.

3.5.1 Collaborative Services

Collaboration services manage the distribution of UAVs to accomplish a mission. Using these services developers only focus on the main functionality of the mission rather than reinventing the wheel. Collaborative services include:

Mission Planner Service which is responsible for analyzing the mission then defining the resources needed to perform the mission according to the current and expected conditions. It decomposes the mission into tasks defining the functionality and parameters for the specified mission.

Task Requester Service which is responsible for requesting these tasks from the broker service. The task requester does not have knowledge about UAVs and their capabilities; however, it requests a certain resource giving its parameters according to the plan and schedule.

Broker Service, where all UAVs register their services and resources to be saved in a database. It has the knowledge about the available UAVs; therefore, it is responsible for allocating tasks to the suitable UAV. Hence, when a request is given, the broker service obtains the description of the request and searches for UAVs with those resources or services. Then it requests those UAVs to find the most suitable and available one with the requested parameters based on the resources available, locations, energy levels and other considerations, for example, to capture a specific location.

3.5.2 UAV Resources and Services

These are accessed according to the available resources in UAVs and the tasks that are required for the mission. UAVs may have one or more of them.

Sensing Services; Most types of payload can be considered sensors, such as temperature sensors, humidity sensors, radar, optical sensors and others. Sensing services collect data from these sensors and send them to the broker service. The request for this service could either be obtaining the value of that sensor, or setting a threshold to be triggered when the sensor meets that condition.

Actuation Services; some UAVs may have to take actions according to certain triggers. UAVs may have output devices such as lights or valves for liquid or gas for spraying missions. A set of actuation services can be provided in each UAV.

Camera Capturing and Video Recording Services; these are considered as separate services as more processes such as filtering and editing are used. Image and video capturing require higher internal memory in the UAVs than other sensors. They also may depend on the required resolution and environmental lighting conditions. Some enhancements can be added to those services such as object

recognition and tracking. However, sending real time images and videos to the user may require specific transportation protocols such as Real-time Transport Protocol (RTP) and Real-Time Streaming Protocol (RTSP).

Energy Monitoring Service that is, when a service is needed to request the UAV's energy. Many decisions are taken according to the energy level. The UAV may return to a specific location when it reaches a certain level. In addition, before allocating a task to a UAV, it must ensure that it has enough energy to complete the task. If a UAV reaches low energy levels during a mission, it can be replaced with a similar UAV or with a set of UAVs. The energy level of each UAV is tracked by the broker service.

Location Monitoring Service which is needed due to the mobility of UAVs. Their locations play an important role in allocating tasks. If a UAV is currently near the mission location, it should be chosen rather than similar UAVs which is located farther from the mission location. The location monitoring service is responsible for locating the UAVs in efficient method minimum power consumption. For example, a GPS consumes high power but gives accurate positions, while using Wi-Fi may give less accurate positions and consumes less power. These methods are managed by the location monitoring service and the UAV locations are saved in the broker service.

Status Service; UAV status could be monitored using the status service that returns the information about the resources. This is called housekeeping data. It includes the condition of the UAV resources.

When the UAV receives the request through the communication subsystem (i.e. Wi-Fi or 3G/4G), the request is then passed to the payload on-board 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 payload on-board computer. After that, the payload on-board sends the command to the requested resource which accordingly performs the service and returns the result to the payload on-board. Finally, it marshals the return message so that the communication subsystem sends it. This process is shown in Figure 3-2.

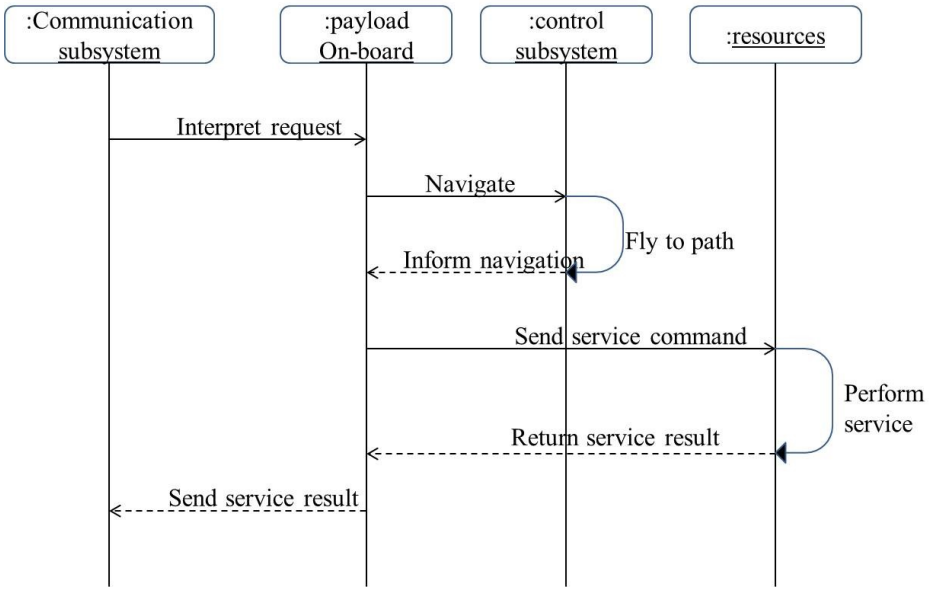


Figure 3-2 Service request sequence diagram for UAV subsystems.

The platform architecture consists of both the collaborative services as well as the UAV services that are accessed through web service APIs. There are different types of web services in different architectures; therefore, the platform should follow the requirements and considerations in the design of an efficient platform.

Chapter 4: UAV-Cloud Platform Architecture

This chapter narrows the research to the UAV side and the broker architecture of the UAV-Cloud framework. The purpose of this chapter is to illustrate UAV resources by presenting their interactions and models as well as the separation layer of the broker and its interactions. The chapter begins by comparing the SOAP and RESTful web services. This is followed by defining the UAV resource types. Then, the ROA model and its RESTful HTTP implementation are demonstrated for UAVs. After that, the broker architecture is proposed giving the process and interfaces with other components.

4.1 Web Service Architectures

There are different architecture styles for distributed computing. Thelin [42] defined them as Service-Oriented, Resource-oriented and Object-Oriented architecture styles. A comparison between distributed architecture is discussed for SOA, ROA and Object Oriented Architecture. The author concluded that the applicability of architecture depends on the application scenario and the system. In addition, he noted that using the single style is better than the combining styles.

There are two main web service architecture styles. First, in the standardized WS* web service architecture, the client requests and the service response objects are encapsulated using SOAP and transmitted over the network 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). Resources are represented in media types, such as JavaScript Object Notation (JSON).

Another comparative study was carried out for mobile hosts [43]. In this scenario, the author illustrated the preferences of REST architecture for mobile hosts because the RESTful services are loosely coupled, flexible and lightweight compared to the SOAP architecture that consumes more bandwidth and is considered more complicated. In addition, Markey and Clynch evaluated the size of a single payload; they found that the JSON Restful call was only 25% the size of the SOAP request [44]. Similarly, Guinard et al. [45] compared the two approaches (the standard WS* web services and the RESTful web services) for the WoT. They concluded that although SOAP is suitable for digital services that emphasize business architecture, the architecture is a complicated approach and it requires high computing power, bandwidth and storage. As a result, it is not suitable for physical-world embedded systems that have limited resources. On the other hand, the RESTful architecture is a reusable and loosely coupled set of web services. Moreover, they reported that it is easier to learn and use for developers [46]. Furthermore, the authors recommended the use of the RESTful web service for the WoT rather than the standard WS* web server unless the application has advanced security and quality of service requirements. The comparison is summarized in Table 4-1.

As a result I propose the use of the RESTful web services for implementing ROA for the UAV cloud. Due to the limited capabilities and resources of UAVs such as energy level and processing power, a simple lightweight web service architecture such as the RESTful is more suitable than a heavyweight complex web service like the WS*. Moreover, the broker layer provides its service APIs as RESTful web services to interact with the requester as well as the UAVs.

Table 4-1 A comparison of SOAP and RESTful web services.

SOAP	RESTful
For enterprise and business process	More suitable for simple services
Suitable for static infrastructure	Suitable for dynamic changeable infrastructure
Operation-centric	Data-centric
Tightly coupled interaction between client and server	Loosely coupled interaction between client and server
Heavyweight web service	Lightweight web service
Complicated coding and changes in server affects the change on the client side	Easy to learn and modify
Binary attachment parsing	Supports all data types directly
Not suitable for wireless infrastructure	Friendly for wireless infrastructure
XML messages	Support various message types
Large size messages that consume more bandwidth	Less message size and bandwidth consumption
Transport layer	Application layer
Old technology, supports standards (WSDL)	New technology, and lacks standards

4.2 Resource Oriented Architecture for UAV-Cloud

In SOA, a service is a functionality performed by a provider. However, in UAVs, the provided interaction is not just services but also data such as sensed data or housekeeping data. These entities are called resources. Therefore, SOA is not sufficient for UAV resources, while ROA is more appropriate to represent them.

4.2.1 REST Architecture

RESTful is the implementation of ROA. The central concept of RESTful web service [47] is that a resource is any component worth being uniquely identified and linked to the cloud. RESTful is described as:

Resource Identification, that is, the URI to identify the resources of each UAV.

Uniform Interface in which resources are available for interaction with well-identified interaction semantic, or HTTP, that has a set of operations to optimize the interactions with the resources.

Self-Describing Message: along with the HTTP interactions, the client and server exchange a set of 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 by both humans and machines; it is also lightweight and can be directly parsed to JavaScript in contrast to XML.

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

4.2.2 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.

Operations: The RESTful HTTP has four operation methods; *GET*, *POST*, *PUT* and *DELETE* are summarized in Table 4-2. In UAVs, the *GET* operation is used to retrieve the current value of a resource. For example, the *GET* method 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 initialize a service providing its required parameters if any are needed, for instance, requesting a *POST* method for a camera resource to take a picture of 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 picture. Then, the *PUT* method is used to modify the parameters of a requested service. For example, a request with *PUT* method is used for a sensor to change its threshold from one value to another, and the new value is determined in the body request. There is also the *DELETE* operation, which is used to cancel a UAV task or release it from the mission. Consequently, the *GET* method retrieves data without affecting the UAVs or resources. Therefore, it is safe to request, while the rest of the methods may change or affect some values or state of the UAVs. As a result, they should be used

carefully, taking into account that UAVs perform actions on the real world that could be irreversible.

Table 4-2 RESTful operations and their usages for UAVs

Operation	Usage
GET	Retrieving the current state of the UAV or its resources
POST	Initialize a service for the mission
PUT	Modifying assigned UAV resources and services
DELETE	Canceling or releasing a UAV from the mission

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 HTTP 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 acknowledged that JSON has widespread support in HTTP. Therefore, the HTTP header in a UAV is set to Content-Type: application/json.

Status Codes: the status of the response has standardized status codes in HTTP. These codes are well-known on the client side to represent the status of the client request. For example, a return code of 200 to the client represents the success

of the request while the 400 code is interpreted as a bad request, meaning that the client's request does not follow the server request rules.

4.2.3 RESTful Models

There are different model scenarios for real time accessing resources of embedded systems, the Pull and Push models [48] [49]. These models are compared for web applications in Table 4-3.

HTTP Pull Model:

In this model, the client pulls the data from the UAV by sending HTTP requests to it frequently using Asynchronous JavaScript and XML (AJAX) to refresh the content without refreshing the client page. This model has proven to be a good way of transferring some of the server workload to the client. The client requests the resource HTTP from the UAV so that it returns the value in the response JSON message. This is suitable for requesting the current value such as housekeeping data or the status of a service or requesting a service.

UAV Push Model:

On the other hand, in the push model, the UAV pushes its data in real time immediately to the client in an HTTP *PUT* request. In this scenario, the client first requests a resource with an event or threshold value. Then, the UAV pushes the data to the client when that event occurs.

This model is suitable for returning the result or notifying the requester at the end of a task that takes time such as sensing a certain location or spraying an area.

Table 4-3 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 event occurs
Client workload	Server workload
suitable for requesting the current value	suitable for notifying event occurrence
Monitoring slow changes	Monitoring sudden changes in server side
Fast changes require low time intervals	Requires client subscription to the event

4.3 Designing the UAV Layer

One of the concerns of integrating UAVs to the cloud is the connectivity. Most UAVs support Wi-Fi connection, therefore it can be used to connect to the Internet. In addition, recently, 3G/4G technology supports not only mobile devices but also embedded systems using external shields connected to the embedded system. By this connection, the UAV gets a unique IP address, so that it has a distinct identification over the Internet. The assumption of the connection availability is valid in many fields of application such as in smart cities. However, considering satellite connectivity opens broader application fields.

The IoT studies the connectivity of smart objects and provides them with addresses as well as applying the IPv4 or IPv6 for them. Integrating UAVs with the Cloud means that the UAV and its resources become available on the Internet to be accessed in a ubiquitous manner to a client user. The client could either be a human using web browsers and applications, a UAV accessing another one's resources, another system that collaborates with UAVs or any embedded devices that use the same protocol. Therefore, the most important step is to identify the resources and services that should be made available to the clients.

4.3.1 UAV Resource and Service Types

UAVs define their resources and services that vary from one to another due to the heterogeneity of UAVs. However, each UAV should have uniform interfaces to enable the client to achieve the following:

Monitor the UAV Housekeeping Data. This involves monitoring the UAV's current status (i.e. whether it is idle or on a mission), the current status of the UAV's storage, the UAV's flight conditions, the direction and orientation of the UAV, the UAV's speed, the energy level and the current position coordinates (altitude, the latitude and longitude values). Mostly, this is identified by the *GET* method with the resource URI.

Access UAV Services which is requesting a service to collect some data from real world, such as sensor readings (e.g. temperature, pressure, or humidity sensors), radar data, camera images or videos, thermal camera images. Services offering this type of information may require some parameters such as specifying the location or QoS. Other services may also generate some form of action by the UAV or the

devices on board. For example, a client request may require the UAV to spray gasses, pesticide or foam. These services are interfaced either as *POST* to initiate the service, *PUT* to change parameter values or *DELETE* to release the service along with the resource URI.

Monitor the UAV Resources, which is keeping track of the different resource payloads onboard such as finding out if a certain resource is available, currently in use or damaged. Another example is determining the remaining amount of liquid for spraying during the mission.

4.3.2 UAV Resource APIs

The UAV is the server back-end that provides its services and resources as web servers through RESTful APIs i.e. HTTP. These resources can be developed in different languages that support RESTful web services programing such as NodeJS, Ruby and Rails, Python or PHP. The variety of programming languages that implement the RESTful protocol facilitates the development of heterogeneous systems for easy collaboration.

For the UAV back-end development, first, it is necessary to identify the APIs for the UAV resource types. A resource is identified through its URI that is expressive and presents its meaning for human interpretation. Then, the exchanged message information is represented as a JSON object that could be easily parsed into JavaScript and be readable for humans. This can then be presented in the browser for the user in an HTML file.

UAV Housekeeping APIs:

The UAV housekeeping data has several resources, and these are modeled as HTTP pull APIs using *GET* method with the resource URI. For example, to retrieve the current energy level of the UAV, it provides the following HTTP request interfaced by the *GET* method:

```
http://.../energy_level
```

Then, the request reads the UAV energy level and returns it as a response in a JSON message:

```
HTTP/1.1 200 OK

Content-Type: application/json

{"id" : 1, "Name" : "uav1", "energy level" : 85}
```

This response indicates that the HTTP is version 1.1, the 200 is the success status code. Then, the `Content-Type: application/json` is to define the content negotiation type as JSON message. Next, the JSON object is the response message of this request that contains the value of the energy level as well as basic UAV information such as its name and ID. Other UAV housekeeping data are similarly designed.

UAV Service APIs:

The UAV provides its services according to the available resource payloads on it through *POST*, *PUT* and *DELETE* HTTP operation requests for each service. For example, for a temperature sensor resource, the UAV provides the following *POST* HTTP URI:

```
http://.../service/temperature
```

along with the JSON body of the request for the location parameters:

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

In this scenario the UAV checks if it is available to accept this request or it is performing another service. In the case where the UAV is available and ready to provide this service, it returns a confirmation response `HTTP/1.1 200 OK.`

Then it moves to the specified location to perform the service, i.e. measure the temperature for example. Then, it sends the collected data to the client HTTP API using the UAV push model. This HTTP contains the collected data in the body request as the following:

```
HTTP/1.1 200 OK

Content-Type: application/json

{"id:2, "name": "UAV2", "service": "temperature", "status": "available", "value": 28.5}
```

When the service is requested, the client may change the parameter value using the *PUT* method for the resource URI:

```
http://.../service/temperature
```

with the JSON body request of the new values defined as:

```
{"location": [{"latitude": 12.7025, "longitude": 45.4263, "altitude": 87.91}]}
```

Accordingly, the UAV modifies the service location to the new values.

Finally, the UAV provides HTTP API for releasing the service using the *DELETE* method for the URI resource. For example, releasing the spraying service using the *DELETE* method for the URI:

```
http://.../service/pesticide_spary
```

This request releases the spray service from the mission operation.

UAV Resourse Status APIs:

Similar to the housekeeping data, the resource monitoring requests the current status of the resource using the *GET* method, such as the URI:

```
http://.../pesticide_spray/tank_level
```

Then it reads the tank level and returns it as a response in a JSON message:

```
HTTP/1.1 200 OK

Content-Type: application/json

{"id" : 1, "Name" : "UAV1", "service" : "pesticide spray", "tank_level" : 40}
```

This response indicates that the pesticide spray tank level of UAV1 has 40% remaining.

The summary of the UAV resource APIs is shown in Table 4-4.

Table 4-4 UAV resources types and their RESTful interfaces

Resource Type	Description	RESTful HTTP interface
UAV housekeeping data	Collecting the status and internal values of the UAV resource	Mainly *GET* method along with the resource URI. *POST*/ *PUT*/ *DELETE* methods could be used for threshold and event feedback
UAV services	Requesting a service from a UAV	*POST* method is used to initiate the service, while *PUT* modifies the parameters. *DELETE* method releases the service
UAV resource data	Monitoring and follow the service status	Mainly *GET* method to the resource parameters to check the status or value of the resource. *POST*/ *PUT*/ *DELETE* with the resource parameter URI could be used for event or feedback notification

The HTTP is a client-server architecture, as shown in Figure 4-1. Therefore, the client application could be built using the UAV APIs. However, in this scenario the client application uses the UAV addresses directly by specifying the task for each UAV. This architecture suffers from limitations such as scalability of adding UAVs to the mission. Moreover, the application is developed for certain UAV resources where changing the UAV leads to modifying the resource address. As a result, I propose using broker architecture connected to a database to isolate the UAV side from the application side, so that the broker is responsible for registering then discovering and allocating the requested resources to the suitable UAVs.

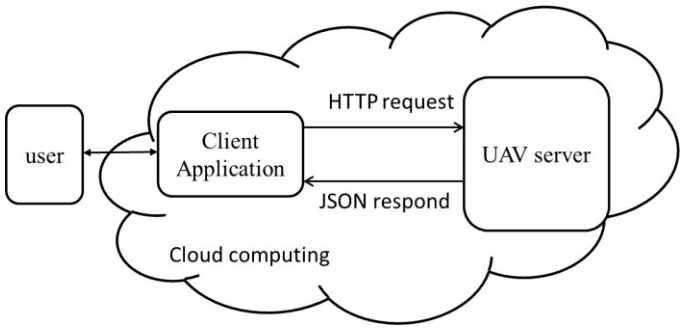


Figure 4-1 Client-Server Architecture.

4.4 UAV Database

UAVs reserve their information internally in their storage. However, due to their limited storage resources, I proposed storing their information and services in a cloud database along with a log track of UAVs identified by timestamps.

This database takes advantage of the cloud scalable resources to store the information. The database is useful for fetching UAVs' services and resources as

well as recording the log data and mission information. Furthermore, it simplifies monitoring the status and changes about UAVs through the mission time-line so that it can be retrieved later on.

The database consists of many tables that include records. A sample of an Entity-Relationship (ER) diagram is shown in Figure 4-2. It may basically have a UAV_info table to store the primary information of the registered UAVs. Most important is the UAV IP address in which it is requested. This information is inserted when the UAV registers itself to the broker. Next, a Resources table is needed to store the resources that UAVs provide. It contains the resource API information which is the URI of the resource, its method and the provider of that resource. These are the basic pieces of information required from a UAV when it registers to the broker. After that, the allocated UAVs for a mission are reserved in the Operation table or even in a separate database. The separation of user databases enables the multi-tenancy by having a distinct database for each tenant.

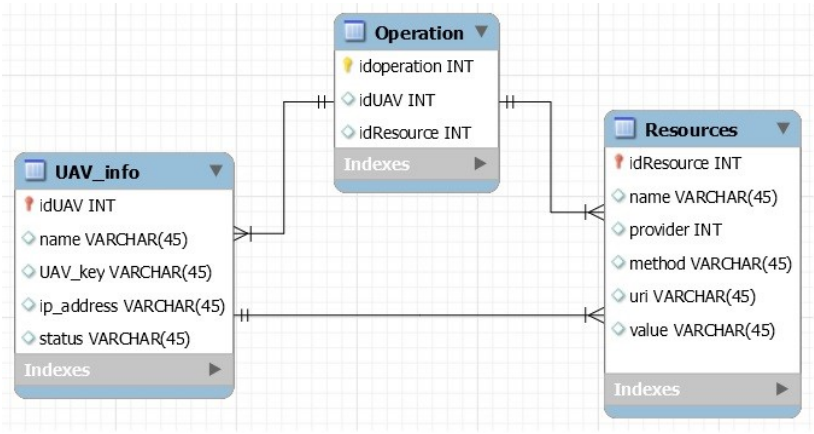


Figure 4-2 UAV Database Sample.

4.5 Designing the Broker Layer

One of the considerations in integrating UAVs to cloud computing is the distribution of UAVs and being scalable to offer their services and resources through APIs to multiple clients. Although ROA is client-service architecture, the RESTful implementation supports the loosely coupled, usability and flexibility services. Moreover, when the developer builds the application or the end-user establishes a mission, they are concerned with the UAV resource and service not a particular UAV. Therefore, the RESTful properties facilitate the cooperation between the required resources and services using a broker architecture to take the responsibility for allocating the suitable UAV for the request, as shown in Figure 4-3.

A broker is a middle-agent that receives advertisements from service providers regarding their capabilities and provision of services. After that, a requester asks the broker for a service specifying the service needed and its parameters. Then the broker compares the requested service against all available advertisements and determines the best match provider. Next, the broker contacts the provider and requests the service. If the provider accepts the request for the service, it performs the service and returns the result to the broker. Finally, the broker returns the result to the requester [50].

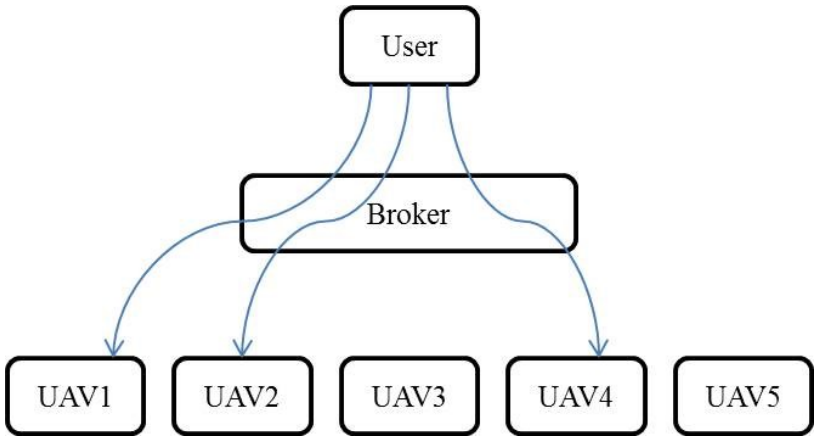


Figure 4-3 Broker layer to separate the application layer from the UAV layer.

For UAVs, the broker is a layer in the UAV-Cloud that is connected to a database, UAVs and requester as shown in the framework Figure 3-1. The broker is responsible for storing and retrieving UAVs' information to/from the database. The broker layer is one of the collaborative services. It manages the process of task-allocations, encapsulating UAV APIs for client requests and receiving UAV data and feedback. Therefore, the UAV back-end and the application front-end do not have direct interactions to request a service or to retrieve a resource information. This process is done through broker web service, as shown in Figure 4-4.

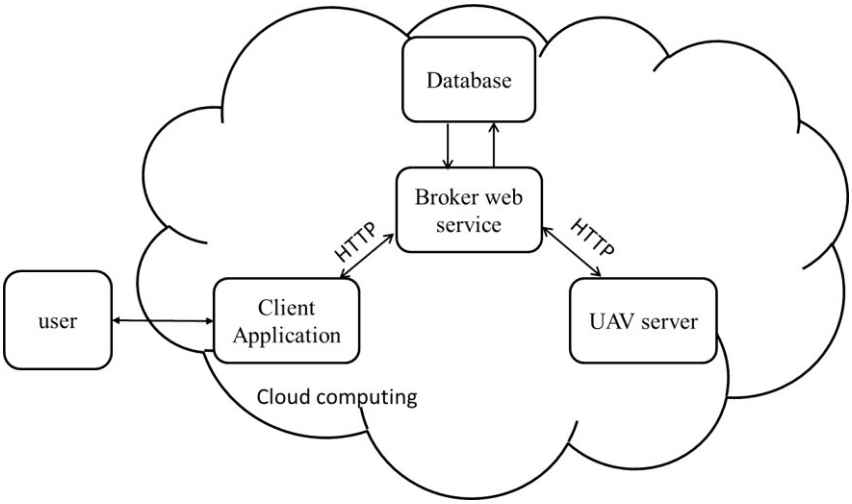


Figure 4-4 Client-Server Architecture with broker layer.

4.5.1 UAV Broker Process

The broker is responsible for the following:

- The broker registers UAVs i.e. information and services. Then, it adds this information to the database.
- The broker receives a resource or service request along with parameters if applicable.
- The broker identifies the suitable UAV from the UAV database to perform the requested task. This process depends on several factors. First, the broker discovers UAVs that are not assigned to other mission and have the resources to accomplish the request according to the request specification, for example, the camera resolution of spray gas quality. Second, in case of multiple available UAVs with the specified specifications, the broker narrows down the choice to the nearest UAV to the location with the highest energy level. In this case, the broker requests their locations and energy levels, and then calculates the distance between their locations and the target locations. Thus the most suitable UAV performs that service. Moreover, the broker may take into account the load balancing, to ensure that similar UAVs are assigned equally so that no one UAV is used more frequently.
- The broker requests the identified UAV using its APIs along with the suitable parameters.
- Then the broker may change the status of that UAV in the database according to the request type.

- The broker returns the request result to the requester.
- The broker receives the UAV push data and returns it to the client.

4.5.2 UAV Broker APIs

The broker provides APIs to the developer to build applications according to the available resources. Moreover, the broker provides APIs for UAVs to register themselves as well as updating their resources and send feedback.

Broker APIs for UAVs

The broker provides APIs to interact with UAVs. These APIs allow them to request several services.

First of all, in order to register a UAV, the broker provides API for UAVs to register themselves. This is a *POST* HTTP with JSON body that includes the UAV information as well as its service information. For example, the following HTTP with a *POST* method is used to register a UAV:

```
http://mybroker.com/register
```

with the JSON body:

```
{"name" : "UAV1", "address" : "176.205.68.244",
"energy level" : 85, "status" : "available",
"orientation": 61.5 , "location": [{"latitude": 36.872,
"longitude" : 140.0704, "altitude" : 260}], "services" :
[
{"name" : "power", "method" : "GET" , "uri" :
"/power"},
```

```
{“name” : "temperature", “method” : “GET”, “uri” :
“/temperature” },

{“name” : "temperature" , “method” : “POST” , “uri”
: “/temperature”}

]}
```

Then, the broker inserts this information into the database and returns an ID to the UAV to confirm registration:

The broker returns the following response to that UAV:

```
HTTP/1.1 200 OK

Content-Type: application/json

{"id": 5}
```

This indicates that the UAV is registered and added to the database with an id =5.

After all UAVs have been registered and recorded in the database, the broker is able to allocate a specific task to the suitable available UAV. Moreover, the client application monitors and tracks the process through the broker service.

Another API for UAVs is used to provide an interface to push their values when an event occurs or threshold is triggered. The broker provides the following URI with the *PUT* method and accepts JSON object:

```
http://mybroker.com/:service
```

where the `:service` is a variable of the service name that updates its value. For example, a triggered UAV to sense temperature in a certain location, the UAV returns the sensed data when it reaches that location using the *PUT* method for the URI:

```
http://mybroker.com/temperature

{"id:2, "name": "UAV2", "value": 28}
```

Where the UAV name is UAV2, id is 2 and the temperature value is 28.

Broker APIs for Application Developers

The broker provides APIs for developers to build applications on top of them, so that the broker receives a request of a resource or a service from the user application. Next, the broker searches the database for UAVs which had registered that service. Then, the broker requests these UAVs to check their availability, energy and location. When the broker obtains the information of these UAVs, it calculates the distance between the current location and the specified requested location. After that, the broker requests the nearest UAV with an applicable energy level to perform the service. Consequently, the requested UAV performs the service and returns the results to the broker using the broker API for UAV push data, which accordingly returns that information to the requester. These information and log data are stored on the database by the broker.

Therefore, the broker APIs are the gate between the application and the UAVs. Developers build the applications following the rules of these APIs to ensure the compatibility with UAV resources. The broker provides several APIs for developers to initiate services and access resources as the following:

For initiating a UAV service, the broker provides the following *POST* method with the following URI:

```
http://mybroker.com/service/:service
```

along with the JSON message that holds the required parameters according to the service requirements, for example, requesting the temperature sensor for certain location using the *POST* method:

```
http://mybroker.com/service/temperature
```

with the JSON body of the request defined as:

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

In this scenario, the broker searches the UAV database for UAVs that provides the temperature service, then checks the availability of them and allocates the task to the nearest one using the suitable UAV API. In the case of a successful task allocation, the broker returns a confirmation response to the application along with the name of that UAV; otherwise the broker informs the requester the unavailability of that service.

When a UAV is allocated to a mission, the UAV and its resources are added to the mission database for monitoring purposes which are accessed by its name and for determining the UAVs that are allocated to that mission.

Next, the broker offers API access to the allocated UAV resources, using *GET*, *PUT* and *DELETE* methods, for example getting the current location of the

UAV1 that is allocated for the temperature services, using the *GET* method for the following URI:

```
http://mybroker.com/:name/:parameter
```

in this case the `:name` is the UAV name that is UAV1 which is given when it is allocated, while the `:parameter` is the location. As a result, the request is a *GET* method with the following URI:

```
http://mybroker.com/UAV1/location
```

the broker searches the address, method and URI of UAV1 that is assigned for the temperature sensing task. Then it requests its assigned location using the UAV housekeeping data API. The response value is then returned to the application as a JSON message.

Similarly, the broker provides APIs for requesting UAVs by provided services rather than name using the *GET* method with the URI:

```
http://mybroker.com/:resource/:parameter
```

In this case, the broker searches the allocated UAVs that provides the `:resource` resource, then requests them to retrieves the `:parameter.`

An example is requesting the remaining tank capacity of the spraying service UAV. This is achieved by the GET method for the URI:

```
http://mybroker.com/pesticide_spray/tank_level
```

In this scenario, the broker requests all the UAVs that provide the pesticide spray service and gets their tank level values then returns them to the client. This API

is suitable for managing a group of UAVs that provides similar resources. The broker APIs are summarized in Table 4-5.

Table 4-5 Broker API interfaces for UAVs and application developers.

Broker APIs for UAVs	Register the UAV to the broker	*POST* method for the registration URI along with the UAV information in JSON message
	Push value according to an event or feedback	*PUT* method for the service URI along with the new value in JSON message
Broker APIs for Application Developers	Initiating a service	*POST* method along with service request URI containing the required parameters in JSON message
	Monitoring UAVs and their resources	*GET*/*PUT*/*DELETE* methods along with the UAV name or the provided resource URI and the JSON message if applicable

4.6 Front-End Application

The front-end application is online software on the client side. The client uses it to establish UAV missions. The application is built on top of the UAV-Cloud platform similar to web application development. It is then deployed to the Cloud and interacts with the Collaborative Service Layer. The application displays a friendly-user interface in a web browser. This interface provides the user with the ability to establish a mission, monitor and access the UAV resources easily (see Figure 4-5 for requesting a camera service). Due to the loosely coupled RESTful architecture, the application layer is built easily on top of platform services using the developer APIs. Therefore, different applications can be built for the same set of UAVs managed by the broker layer.

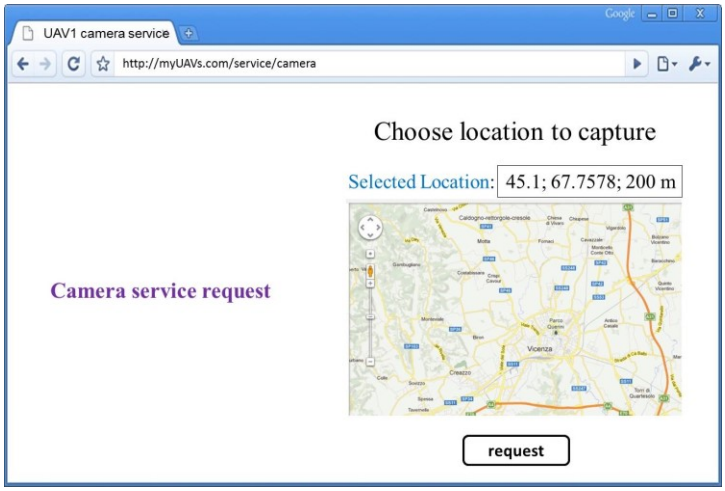


Figure 4-5 Requesting camera service for specific location.

Chapter 5: Implementation Experiment

This chapter illustrates the implementation and testing of the proposed UAV-Cloud architecture. The implementation includes building the UAV resources and providing their APIs. After that, the broker was developed to separate the requester side from the UAV resource side. The broker was connected to the database that store the UAV and resources information. The implementation covers the shaded components of the UAV-Cloud architecture, as shown in Figure 5-1.

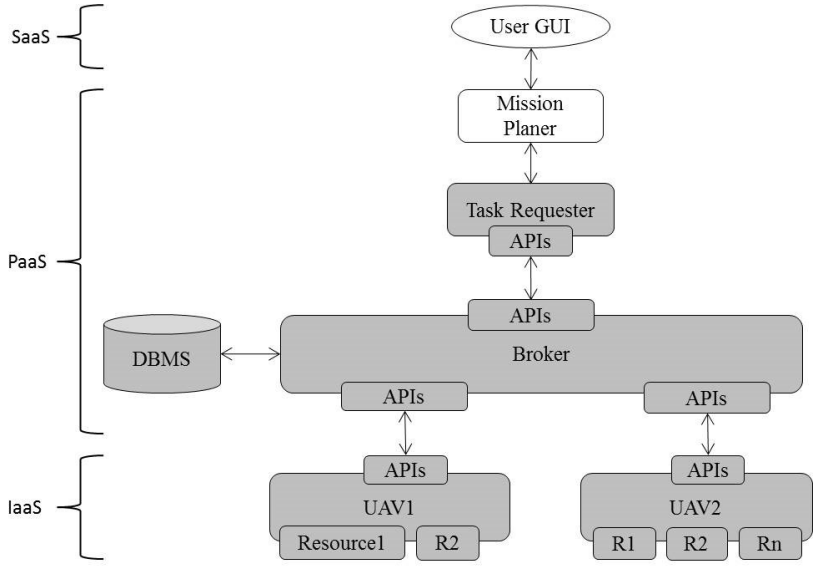


Figure 5-1 The implemented system components of the UAV-Cloud architecture are shaded in gray.

5.1 Implementation

5.1.1 UAV Resources Implementation

First for hardware part, the UAV was built using the Arduino board[1] which is an open source hardware for embedded systems. For this research, the Arduino was

[1] http://www.arduino.cc/

implemented as the UAV payload subsystem that is the on-board device for resources and services, and then sensors were connected to the Arduino such as DHT11[2] sensor for temperature and humidity and ultrasonic for distance measurements. In addition, a buzz and some LEDs were attached to represent actuators as shown in Figure 5-2. Moreover, for the Internet connectivity, an Adafruit CC3000 Wi-Fi board[3] was used to connect the Arduino to the Internet and get an IP address.

The Arduino was developed using the Arduino software[4] in the C language with the Adafruit CC3000 library[5] to read the request. Each resource was implemented with a RESTful API.

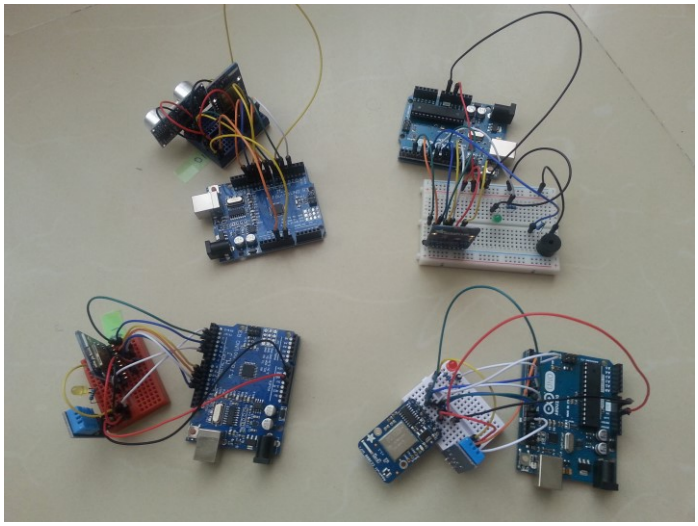


Figure 5-2 Four Arduino boards connected with Adafruito CC3000 boards as well as sensors and actuators representing UAV payload systems and their resources.

[2] https://github.com/adafruit/DHT-sensor-library

[3] https://www.adafruit.com/products/1469

[4] http://arduino.cc/en/main/software

[5] https://github.com/adafruit/Adafruit_CC3000_Library

The UAV resources were implemented for four UAVs. Each one has different resources, IP address and RESTful APIs. However, UAVs that have the similar resource, define their API interface in the same way. For simplicity, only the *GET* method was used for the implementation. The implemented UAV resources and services are summarized in Table 5-1:

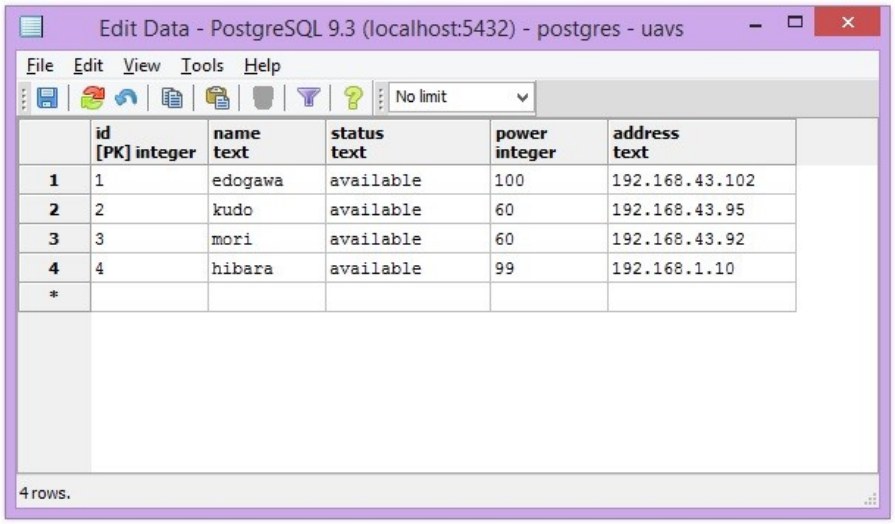
Table 5-1 Implemented UAV resources and their interfaces.

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	Refill 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	Return 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

5.1.2 Database Implementation

After that, database tables were implemented in PostgreSQL database[6] through PgAdmin platform[7]. The database was designed as in Figure 4-2 which includes three tables; UAV_info table for all registered UAV information such as ID, name, address and status (see Figure 5-3), second the Resources table for UAV services and resources (see Figure 5-4) and third the Operation table for allocated UAVs for a mission containing the requests log (see Figure 5-5). The database is accessed by the broker to retrieve, write and modify data through its configurations.

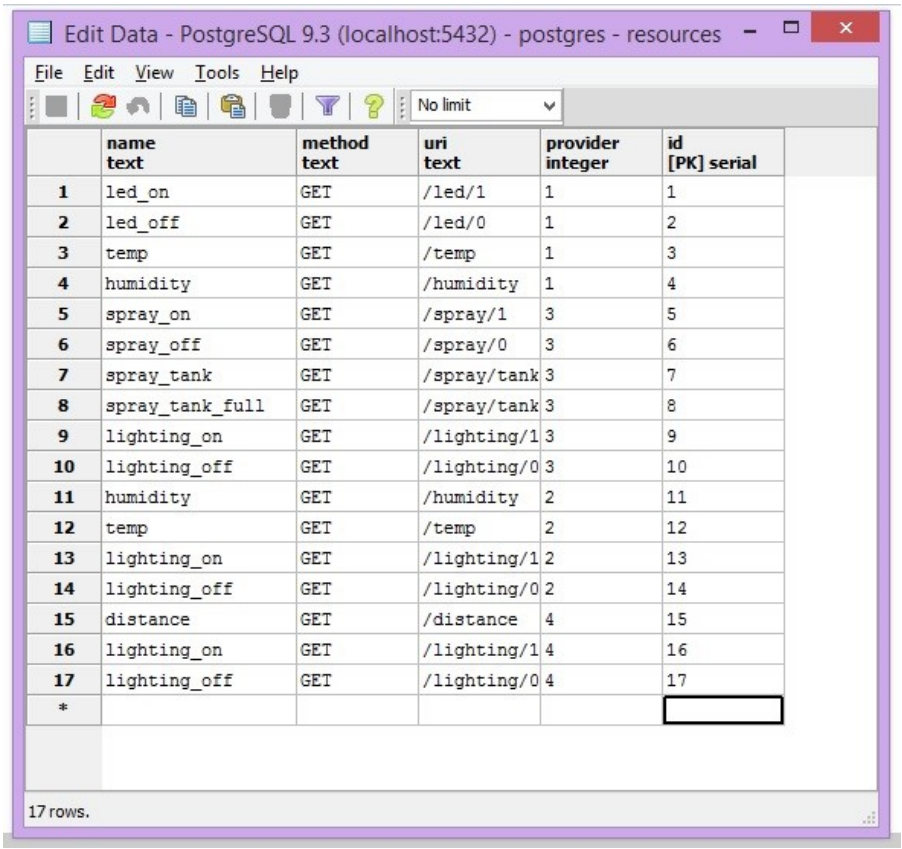


	id [PK] integer	name text	status text	power integer	address text
1	1	edogawa	available	100	192.168.43.102
2	2	kudo	available	60	192.168.43.95
3	3	mori	available	60	192.168.43.92
4	4	hibara	available	99	192.168.1.10
*					

Figure 5-3 UAV table in PostgreSQL database using PgAdmin platform.

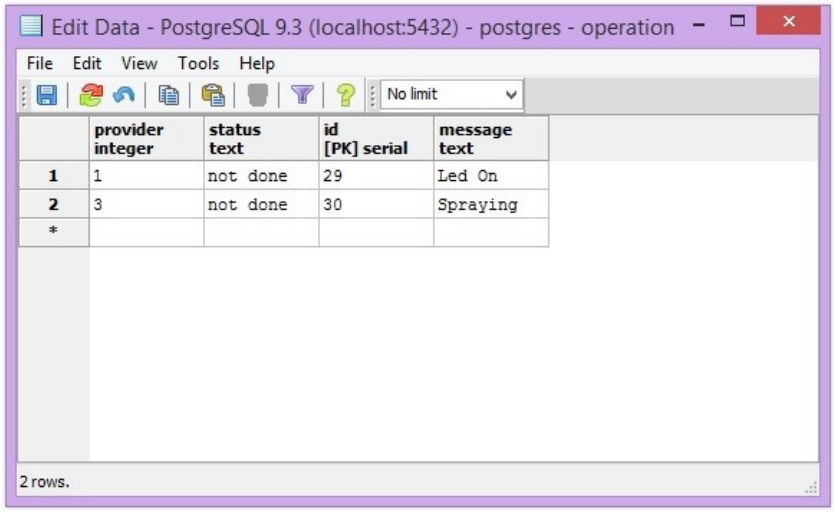
[6] http://www.postgresql.org/

[7] http://www.pgadmin.org/



	name text	method text	uri text	provider integer	id [PK] serial
1	led_on	GET	/led/1	1	1
2	led_off	GET	/led/0	1	2
3	temp	GET	/temp	1	3
4	humidity	GET	/humidity	1	4
5	spray_on	GET	/spray/1	3	5
6	spray_off	GET	/spray/0	3	6
7	spray_tank	GET	/spray/tank	3	7
8	spray_tank_full	GET	/spray/tank	3	8
9	lighting_on	GET	/lighting/1	3	9
10	lighting_off	GET	/lighting/0	3	10
11	humidity	GET	/humidity	2	11
12	temp	GET	/temp	2	12
13	lighting_on	GET	/lighting/1	2	13
14	lighting_off	GET	/lighting/0	2	14
15	distance	GET	/distance	4	15
16	lighting_on	GET	/lighting/1	4	16
17	lighting_off	GET	/lighting/0	4	17
*					

Figure 5-4 Registered UAV resource table in PostgreSQL database using PgAdmin platform.



	provider integer	status text	id [PK] serial	message text
1	1	not done	29	Led On
2	3	not done	30	Spraying
*				

Figure 5-5 Operation table of assigned UAVs in table in PostgreSQL database using PgAdmin platform.

5.1.3 Broker Implementation

Next, the broker service was built using the NodeJS platform[8] in JavaScript language. First, the broker was connected to the database using its configuration parameters such as host, database name, port, user name and password to retrieve and write values from certain tables. After that, RESTful APIs were built for the broker to allow users to request the required services or resources. The broker implementation focused on the developer APIs mentioned in Section 4.5.2. The implemented APIs for service requests are either allocating a new service by adding a UAV to the emission, modifying a service request, or retrieving a value of a parameter. The APIs were defined by the uniform interface operations summarized in Table 4-2.

The request of allocating a new service is the *POST* operation for the following API:

```
http://localhost:3000/service/:service
```

In this request, the `:service` is a parameter for any service name that the user defines, for example turning on the spraying service by requesting the *POST* method for the following API:

```
http://localhost:3000/service/spray_on
```

to allocate the suitable available UAV that has the spray resource and add this UAV to the operation.

[8] https://nodejs.org/

Moreover, to modify or retrieve a value of a resource, the broker provides the following *PUT* method API:

```
http://localhost:3000/:name/:service
```

In this situation, the `:name` and the `:service` are the parameters of the UAV name and the service to be accessed or modified. For example, requesting the following API by the *PUT* method:

```
http://localhost:3000/UAV2/spray_off
```

This request is to turn off the spray service of UAV2.

5.2 Testing

The implemented system was tested using the Postman Chrome extension[9] for each device and resource then for broker APIs. The test focuses on the pull data model of HTTP requests.

First, the test begins with testing the UAV resource APIs, by directly 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 resources and then returned the response of the requested service. The services mentioned in Table 5-1 were tested successfully with quick response.

Secondly the broker APIs were tested as the following; for requesting a service, the system was tested by sending requests of services for the *POST* API:

```
http://localhost:3000/service/:service
```

[9] https://www.getpostman.com/docs/requests

such as allocating a spraying service through the POST API:

```
http://localhost:3000/UAV2/spray_off
```

In this scenario the broker searches the database for the service spray in the services' table combined with the UAVs' table to find the available one that provides the spraying service. Then, the broker defines the UAV API components that are, the method, address, resource URI and the name of the allocated UAV to request it so that it performs the required service. After that, the broker changes the status of that UAV into allocated in the UAVs' table, to ensure that this UAV is not assigned again but could be modified and accessed through *GET, PUT* and *DELET* methods.

After the broker requests the allocated UAV, this UAV replies with a confirmation for performing the service. Next, the broker returns the response to the client as a JSON message to the requester containing the name of the UAV, the name of the requested service, and the UAV feedback message, as shown in Figure 5-6.

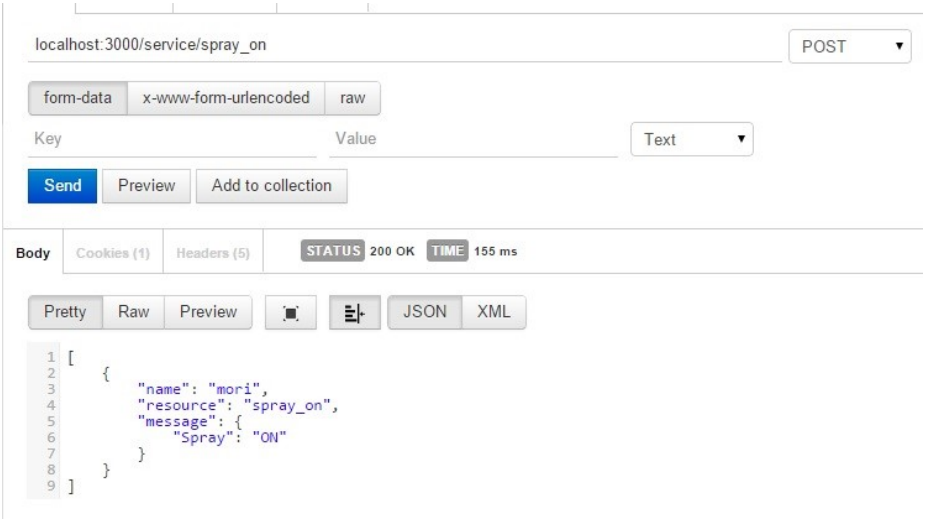


Figure 5-6 *POST* operation request and response for spraying service through the broker

Similarly, when requesting another service to be performed by second UAV, such as ‘led_on’ the *POST* method is used along with the resource name, as shown in Figure 5-7.

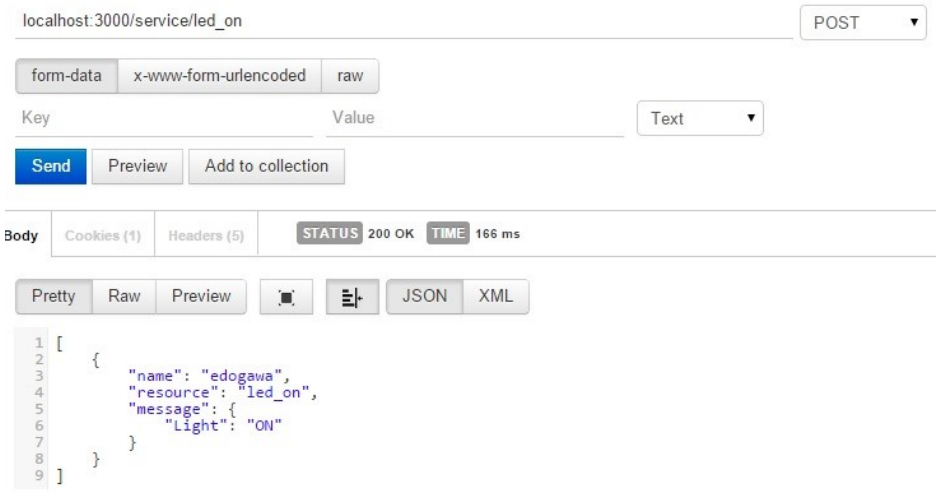


Figure 5-7 *POST* operation request and response for ‘led_on’ service through the broker.

Next, the allocated services are accessed through *PUT* APIs that specify the name of the UAV to be modified and the service name. This was tested for several services of different UAVs such as turning the spray off as well as turning the ‘led_off’ as shown in Figure 5-8 and Figure 5-9 respectively.

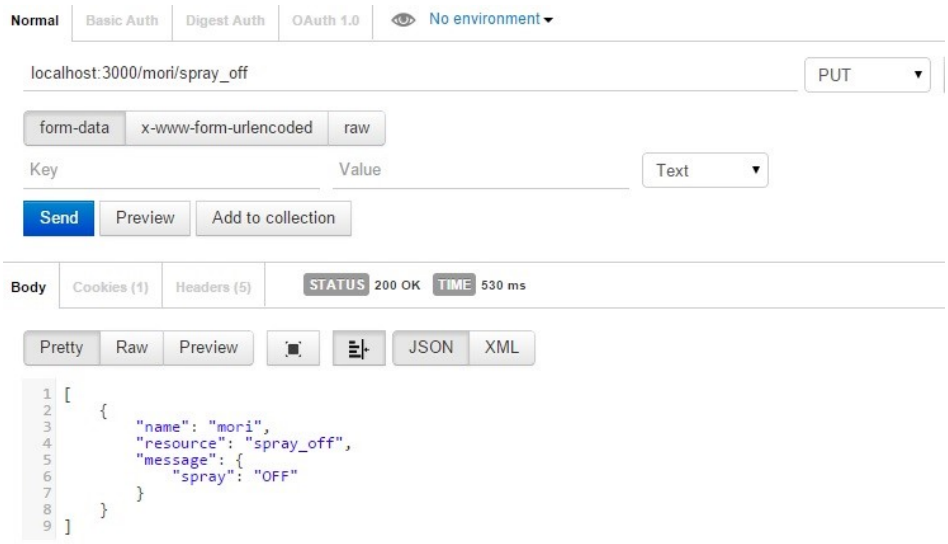


Figure 5-8 *PUT* operation request and response for turning spray service off through the broker.

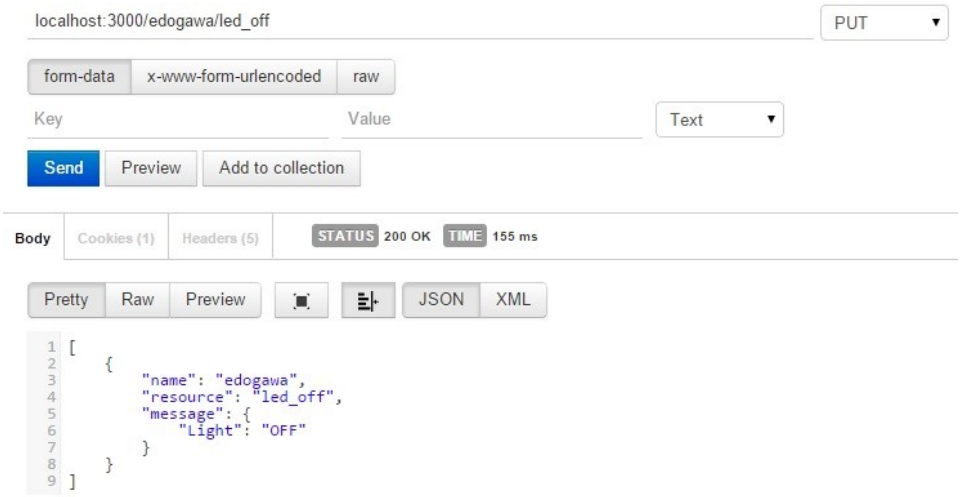


Figure 5-9 *PUT* operation request and response for turning 'led service off' through the broker.

In the same way, the sensor readings were retrieved by specifying the name of the UAV and its resource, as shown in Figure 5-10.

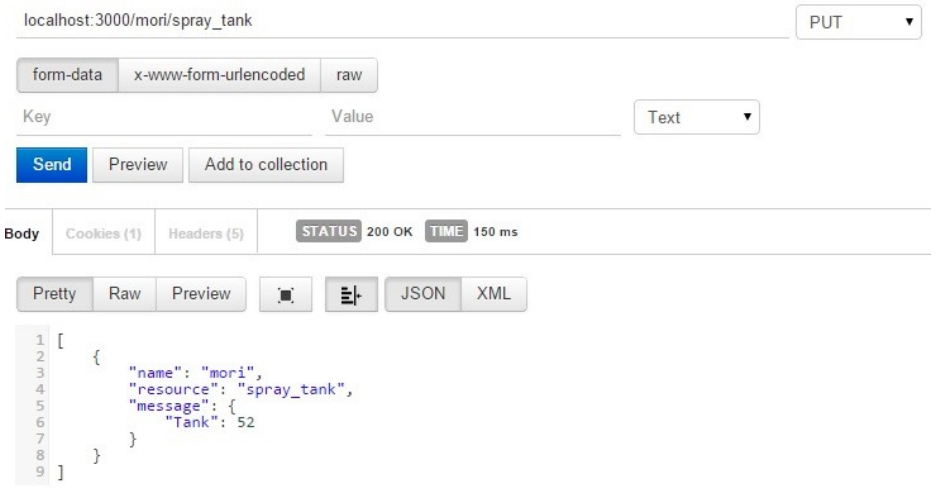


Figure 5-10 Reading the remaining tank capacity of the spraying service UAV

For these scenarios, the broker searches the allocated UAVs that provides the service from the Operation table and the Resource table. Then, it requests the UAV API using its address, operation and URI. With this, it will return the response to the client.

The architecture showed the seperation of the client side from the UAV side by the broker layer that allocates the suitable UAV to the operation from the set of UAVs. In case of no service provider or no available UAV for that service, the broker returns a not available message response to the client. Moreover, in case of requesting an allocated UAV, it returns a rejection response that it is not available.

5.3 Evaluation

For measuring the overload of the broker layer, the response times for the resources were compared in both direct access and through the broker.

First, the UAV resources in Table 5-1 were requested directly using their URIs and the UAV address. The response times were recorded ten times for each

resource and the average was calculated as shown in Table 5-2 and Figure 5-11. 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.

Table 5-2 Response times for UAV resources with direct accesses.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		response time in ms		2	3	4	5	6	7	8	9	10	**Avarage**
2			direct link										
3	Edogawa	/temp	460	475	571	440	458	460	466	465	450	435	**468**
4		/humidity	432	426	545	152	419	432	433	429	440	437	**414.5**
5		/light/1	256	174	169	170	160	185	246	184	170	187	**190.1**
6		/light/0	532	158	157	141	162	199	188	161	166	205	**206.9**
7	Mori	/lighting/1	166	170	187	200	467	208	167	167	166	160	**205.8**
8		/lighting/0	288	285	301	227	316	322	173	280	337	218	**274.7**
9		/spray/1	223	169	166	168	160	331	230	342	231	154	**217.4**
10		/spray/0	159	238	155	664	190	364	235	196	227	291	**271.9**
11		/spray/tank	161	363	156	142	279	153	163	185	152	161	**191.5**
12		/spray/tank/full	167	155	180	216	200	156	176	173	159	193	**177.5**
13	Kudo	/temp	445	163	489	493	499	477	492	153	426	458	**409.5**
14		/humidity	449	437	494	442	246	452	423	214	455	214	**382.6**
15		/lighting/1	198	259	164	158	175	157	155	159	158	181	**176.4**
16		/lighting/0	272	417	298	149	351	333	260	328	396	283	**305.7**
17	Hibara	/distance	161	145	201	201	173	165	174	228	196	252	**189.6**
18		/lighting/1	158	232	175	193	174	156	153	179	170	254	**184.4**
19		/lighting/0	272	378	371	397	199	259	191	155	215	188	**262.5**
20													**266.4117647**

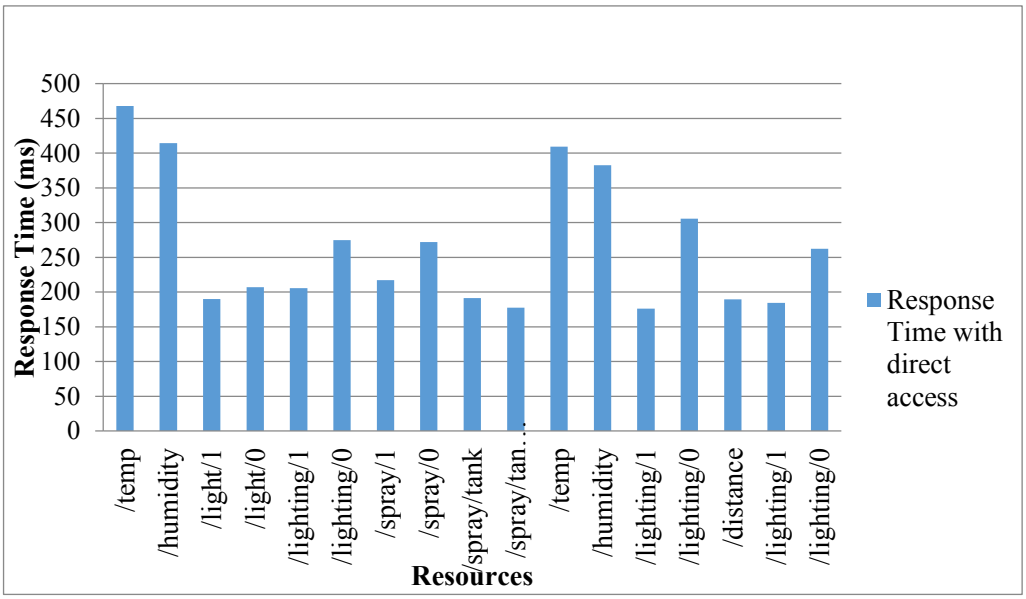


Figure 5-11 Response times of UAV resources with direct accesses.

After that, the UAV resources were requested through the broker layer by specifying the name of the service to the broker, so that it ensures the availability of the requested resource and requests the UAV according to its uniform interface and returns the results to the requester. The measurement was done ten times for each service of each UAV. The response time of requesting services through the broker varies between 200 and 500 milliseconds with an average response time of 310 milliseconds as shown in Table 5-3 and Figure 5-12.

Table 5-3 Response times of UAV resources through the broker.

	A	B	C	D	E	F	G	H	I	J	K	L	M
22		response time in ms		2	3	4	5	6	7	8	9	10	**Avarage**
23			through broker										
24	Edogawa	/temp	458	473	557	453	485	465	185	448	918	467	**487.9**
25		/humidity	476	461	481	442	461	498	482	478	456	440	**467.5**
26		/light/1	225	261	205	180	187	228	216	196	207	191	**209.6**
27		/light/0	302	198	203	504	182	170	457	198	192	336	**274.2**
28	Mori	/lighting/1	183	514	189	245	204	182	169	189	175	217	**226.7**
29		/lighting/0	210	1096	180	277	355	345	345	299	372	359	**383.8**
30		/spray/1	202	200	245	162	238	189	233	489	261	206	**242.5**
31		/spray/0	471	504	641	202	215	294	274	294	194	202	**329.1**
32		/spray/tank	209	164	216	251	196	326	192	202	185	221	**216.2**
33		/spray/tank/full	194	178	224	209	202	177	197	187	179	169	**191.6**
34	Kudo	/temp	483	480	455	481	533	612	501	495	464	520	**502.4**
35		/humidity	547	501	451	460	481	489	520	564	513	453	**497.9**
36		/lighting/1	169	169	179	173	163	167	435	169	171	223	**201.8**
37		/lighting/0	278	197	550	516	252	218	196	496	183	187	**307.3**
38	Hibara	/distance	271	470	175	219	186	197	166	159	165	171	**217.9**
39		/lighting/1	187	174	173	195	164	171	179	244	439	201	**212.7**
40		/lighting/0	598	475	205	216	248	229	251	196	304	242	**296.4**
41													**309.7352941**

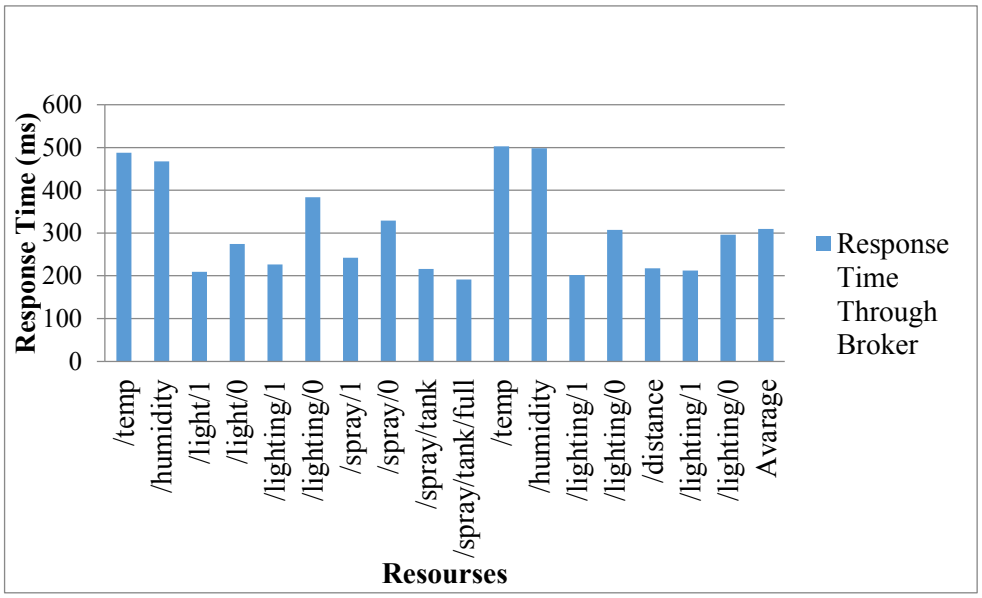


Figure 5-12 Response times of UAV resources through the broker.

Accordingly, the overhead of the broker layer is calculated for the resources as shown in Figure 5-13. The average increase of the response time is only 13%. This is due to the difference between the UAV and the computer processing capabilities.

Consequently, the cloud services transfer part of the processing from internal UAVs to the cloud and add more advantages with minimal overhead. This shows the high performance of the broker layer compared to the limited resources of UAVs.

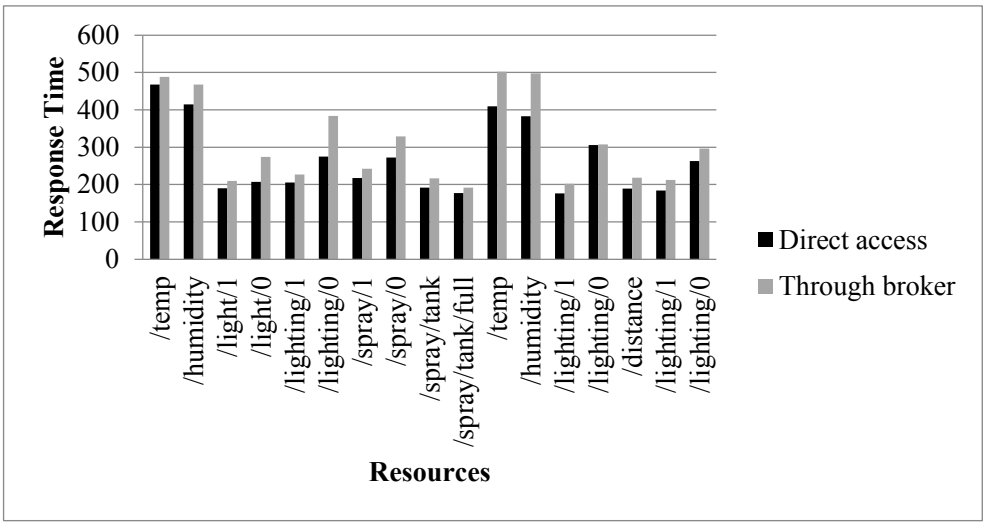


Figure 5-13 Response times of UAV resources with direct accesses and through the broker.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

In conclusion in this research, I proposed a UAV-Cloud platform for distributed UAVs. This platform offers several advantages for developing UAV applications easily, separating responsibilities of UAV services and integrating them. To facilitate this approach I proposed a ROA and described a broker layer to separate the application side from the UAV side.

The proposed UAV-Cloud platform overcomes the limitations of the traditional peer-to-peer RF communication that have showed numerous restrictions for operation and development. In addition, developing a heterogeneous UAV application using the traditional approaches is time and effort consuming because it requires the knowledge of each UAV programing language. The operation of UAVs is also limited to specific missions. Furthermore, in the RF communication scenario, the user location has to be within the mission area. Moreover, it restricts UAVs to be in a nearby area and to be in a line of communication with the ground station. This is unsuitable for the dynamic UAVs environments where UAVs have to be spread across large areas and may not have a direct line of communication with the ground station or between them. Besides, the development of heterogeneous UAVs becomes a difficult process for different UAV programing languages.

As a result, I proposed integrating UAVs to the cloud for ubiquitous UAV resource access. In this model, UAVs are considered as web servers that are part of the cloud so that they gain the benefit of the cloud computing ubiquity as well as facilitating the use of web tools and protocols for developing collaborative UAV

applications. Following the cloud web development opens the ability to develop not only desktop applications but also mobile applications for UAVs. In addition, these applications are accessed regardless of the user operating system.

UAVs provides not only service but they are also resources. The RESTful is the implementation of the ROA; it is a lightweight, reusable and loosely coupled web service. It is more suitable for UAV limited resources compared to the standardized heavyweight and complex WS* web services. Therefore, the UAVs were designed using RESTful web services to offer their resources and services using HTTP uniform interfaces. UAVs provide these HTTP APIs their resources and services which can be accessed and requested through the broker layer.

Due to the loosely coupled services and to gain the benefit of separating responsibilities, a broker architecture was proposed which is a web service on the cloud. The broker is connected to a database that holds the information about the registered UAVs and their resources, so that the user application is built upon it to request and monitor the process of the mission.

The research focused on the framework architecture and the functionality provided by the platform. On the other hand, there is a set of non-functional requirements provided by the framework which include reusability of the framework services due to the ROA design. Furthermore, the platform APIs support the usability for easy development as building blocks for implementing applications. Not only that, but also transferring the common services from the UAV side to the cloud side increases the efficiency of these services. In addition, due to the standardized communication and protocols, the platform supports interoperability where heterogeneous systems are able to exchange data and messages in an agreed-upon

format. This also allows compatibility with other systems that use this protocol. Besides, the system measured the performance of the services by direct access as well as through the broker and showed that the response time is slightly higher. This indicates that the platform layer does not lead to overheads for the system.

However, some other non-functional requirements were not addressed such as availability, recovery, failure management, safety and testability. Another important aspect is security and privacy. The exchanged data, platform access, UAV resources and database require security mechanisms for accessing them and the exchanged messages. The platform APIs enable having access tokens for access authentications. Also, encryption and decryption are preferable for exchanged data and messages.

The proposed architecture was implemented as a UAV payload subsystem. The implementation included a communication subsystem to connect to the network and get a unique IP. Then the payload for each device contained a resource that retrieved data and one to perform action. Each resource had its API to allow access for RESTful requests. This showed the separation of responsibilities and facilitated building applications and integrating services easily. This was followed by developing the broker layer which was connected to the database that contained the information of the registered UAVs, their services and the operation information. The broker APIs were used to assign a new UAV to the mission by defining the service. In addition, they were used for modifying selected services and retrieving values from the assigned UAVs. These were tested using a simple browser application to demonstrate the interfaces of UAVs and the broker for several UAVs and their resources. The overhead of the broker was measured and found that the

response time through the broker was only 13% higher than the direct access. This is an acceptable overhead for the added broker features.

On the other hand, the implementation has some limitations. It did not measure the scalability of the broker and how many UAVs it can deal with. This also includes the maximum number of requests that can be handled simultaneously. In addition, the impact of the concurrent requests on the response time and how the broker handles them were not investigated. The implemented prototype used fixed devices; therefore, the mobility factor and location considerations were not implemented. Only the pull model was implemented. The push model of registration was assumed available. Although the API supports heterogeneous devices, the implementation of UAV is based on similar Arduino devices with different resources.

From a business perspective, the requested architecture opens new opportunities to the UAV industry by using cloud pricing model of pay-per-use and resource sharing. The user operation does not have to go through the whole process of owning the UAVs, developing them as well as operating and using them. The cloud development models are (i) private cloud, (ii) public cloud, and (iii) hybrid cloud.

The private UAV cloud provides services and infrastructure only for its organization; this could either be managed by the organization itself or through a third party. In this situation, the UAVs are owned by the organization and the applications are developed according to its needs and operations.

On the other hand, the public UAV cloud provides services to an open network, this opens the field for business UAV applications for the public, where the user does not own or manage the UAVs but only gains the benefit of their usage. This is cost effective for public users who cannot afford the infrastructure and management process of UAVs.

Moreover, the hybrid UAV cloud is a combination of the public and private cloud, where the UAVs are owned and managed by a third party for a specific organization. This reduces the organization responsibilities of managing and maintaining the UAVs to focus on their usage and operation.

A comparison of the addressed features is compared to the literature review as shown in Table 6-1. Although some literature addressed part of these feature, no general platform was proposed for UAV resources using the reusability and cloud computing paradigm. Moreover, most of these researchers consider applications for a specific field. Therefore, the design is tightly coupled and not considered for other applications.

Table 6-1 A comparison among the UAV-Cloud and other related solution in the addressed features.

	Simanta [29]	Freitas [27]	Nadeau [30]	Mohamed [23]	UAV-Cloud
SOA	✓	✓	✓	✓	
Loosely coupled					✓
Power considerations		✓			✓
Location considerations		✓	✓		✓
Reusability					✓

Platform					✓
Business prospective					✓
Application independent				✓	✓
Lightweight architecture					✓
Separating responsibilities		✓			✓
Integrating with other system		✓			✓
Cloud resources					✓
Ease application development					✓
Multi UAVs	✓	✓	✓	✓	✓

6.2 Future Work and Open Issues

The proposed architecture does not cover the whole UAV-Cloud considerations mentioned in 3.4. The payload subsystem has a high dependency on the controlling aspects of flight path. Therefore, the control subsystem could have interfaces to link the UAV services with it. For example, the broker allocation for the nearest UAV depends on the flight path to the destination point.

In addition, another layer is required to decompose the user mission into a set of tasks to be requested by the broker. This decomposition highly depends on the operation of the mission. Therefore, it was assumed to be part of the application.

UAVs are not stand alone systems. They usually interact and exchange data with other systems. The proposed architecture can be expanded to open the ability for application to integrate not only with UAVs but also ground nodes and other systems that use the same RESTful protocol. Therefore, the application combines multiple resources to increase its efficiency and capabilities.

In addition, UAVs provide a huge volume variety of collected data, this opens the Big Data field to analyze this data for future decision- making in different operations.

Although the RESTful architecture is acknowledged to be suitable for the limited UAVs, it still lacks standards. For example, it lacks a standardized description format for representing UAV information and service details. Also, the push model is an open issue in this field that requires more enhancements.

Bibliography

[1] B. Wagner, “Civilian market for unmanned aircraft struggles to take flight,” *Natl. Def. Mag.*, 2007.

[2] G. M. Saggiani and B. Teodorani, “Rotary wing UAV potential applications: an analytical study through a matrix method,” *Aircr. Eng. Aerosp. Technol.*, vol. 76, no. 1, pp. 6–14, 2004.

[3] O. K. Sahingoz, “Mobile networking with UAVs: opportunities and challenges,” in *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, 2013, pp. 933–941.

[4] G. Chmaj and H. Selvaraj, “Distributed processing applications for UAV/drones: a survey,” in *Progress in Systems Engineering*, Springer, 2015, pp. 449–454.

[5] W. L. Teacy, J. Nie, S. McClean, G. Parr, S. Hailes, S. Julier, N. Trigoni, and S. Cameron, “Collaborative sensing by unmanned aerial vehicles,” 2009.

[6] A. Ryan, J. Tisdale, M. Godwin, D. Coatta, D. Nguyen, S. Spry, R. Sengupta, and J. K. Hedrick, “Decentralized control of unmanned aerial vehicle collaborative sensing missions,” in *American Control Conference, 2007. ACC’07*, 2007, pp. 4672–4677.

[7] P. Zhan, D. Casbeer, and A. L. Swindlehurst, “A centralized control algorithm for target tracking with UAVs,” in *39th IEEE Asilomar Conference*, 2005, vol. 13, p. 109.

[8] N. Mohamed and J. Al-Jaroodi, “Service-oriented middleware for collaborative UAVs,” in *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, 2013, pp. 185–192.

[9] A. Ryan, X. Xiao, S. Rathinam, J. Tisdale, M. Zennaro, D. Caveney, R. Sengupta, and J. K. Hedrick, “A modular software infrastructure for distributed control of collaborating UAVs,” in *Proceedings of the AIAA Conference on Guidance, Navigation, and Control, Keystone, CO*, 2006.

[10] P. Mell and T. Grance, “The NIST definition of cloud computing,” 2011.

[11] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[12] D. Zeng, S. Guo, and Z. Cheng, “The web of things: A survey,” *J. Commun.*, vol. 6, no. 6, pp. 424–438, 2011.

[13] R. Austin, *Unmanned aircraft systems: UAVS design, development and deployment*, vol. 54. John Wiley & Sons, 2011.

[14] G. Varela, P. Caamamo, F. Orjales, A. Deibe, F. López-Peña, and R. J. Duro, “Swarm intelligence based approach for real time UAV team coordination in search operations,” in *Nature and Biologically Inspired Computing (NaBIC), 2011 Third World Congress on*, 2011, pp. 365–370.

[15] F. G. Costa, J. Ueyama, T. Braun, G. Pessin, F. S. Osório, and P. A. Vargas, “The use of unmanned aerial vehicles and wireless sensor network in agricultural applications,” in *Geoscience and Remote Sensing Symposium (IGARSS), 2012 IEEE International*, 2012, pp. 5045–5048.

[16] F. Mohammed, A. Idries, N. Mohamed, J. Al-Jaroodi, and I. Jawhar, “UAVs

for smart cities: Opportunities and challenges," in *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, 2014, pp. 267–273.
[17] F. Mohammed, A. Idries, N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "Opportunities and Challenges of Using UAVs for Dubai Smart City," in *2014 6th International Conference on New Technologies, Mobility and Security (NTMS)*, 2014, pp. 1–4.
[18] H. Bendea, P. Boccardo, S. Dequal, F. Giulio Tonolo, D. Marenchino, and M. Piras, "Low cost UAV for post-disaster assessment," in *Proceedings of The XXI Congress of the International Society for Photogrammetry and Remote Sensing, Beijing (China), 3-11 July 2008*, 2008.
[19] H. Eisenbeiss, "A mini unmanned aerial vehicle (UAV): system overview and image acquisition," *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. 36, no. 5/W1, 2004.
[20] H. Chao, Y. Cao, and Y. Chen, "Autopilots for small unmanned aerial vehicles: a survey," *Int. J. Control Autom. Syst.*, vol. 8, no. 1, pp. 36–44, 2010.
[21] S. Hadim, J. Al-Jaroodi, and N. Mohamed, "Middleware issues and approaches for mobile ad hoc networks," in *The IEEE Consumer Communications and Networking Conf.(CCNC 2006)*, 2006, pp. 431–436.
[22] N. Mohamed, J. Al-Jaroodi, I. Jawhar, and S. Lazarova-Molnar, "Middleware requirements for collaborative unmanned aerial vehicles," in *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, 2013, pp. 1051–1060.
[23] N. Mohamed, J. Al-Jaroodi, I. Jawhar, and S. Lazarova-Molnar, "A service-oriented middleware for building collaborative UAVs," *J. Intell. Robot. Syst.*, vol. 74, no. 1–2, pp. 309–321, 2014.
[24] S. T. Patibandla, T. Bakker, and R. H. Klenke, "Initial evaluation of an IEEE 802.11 s-based mobile ad-hoc network for collaborative Unmanned Aerial Vehicles," in *Connected Vehicles and Expo (ICCVE), 2013 International Conference on*, 2013, pp. 145–150.
[25] T. Lemaire, R. Alami, and S. Lacroix, "A distributed tasks allocation scheme in multi-UAV context," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, 2004, vol. 4, pp. 3622–3627.
[26] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 7–18, 2010.
[27] E. P. de Freitas, A. M. Ferreia, C. E. Pereira, and T. Larsson, "Middleware support in unmanned aerial vehicles and wireless sensor networks for surveillance applications," in *Intelligent Distributed Computing III*, Springer, 2009, pp. 289–296.
[28] C. E. Lin, C.-R. Li, and Y.-H. Lai, "UAS Cloud Surveillance System," in *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, 2012, pp. 173–178.
[29] S. Simanta, D. Plakosh, and E. Morris, "Web services for handheld tactical systems," in *Systems Conference (SysCon), 2011 IEEE International*, 2011, pp. 115–122.
[30] S. Se, C. Nadeau, and S. Wood, "Automated UAV-based video exploitation using service oriented architecture framework," in *SPIE Defense, Security, and Sensing*, 2011, p. 80200Y–80200Y.
[31] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey,"

Comput. Netw., vol. 54, no. 15, pp. 2787–2805, 2010.
[32] S. Gustafson and A. Sheth, "Web of Things," *Comput. Now*, vol. 7, no. 3, 2014.
[33] D. Guinard, V. Trifa, and E. Wilde, "A resource oriented architecture for the web of things," in *Internet of Things (IOT), 2010*, 2010, pp. 1–8.
[34] D. Guinard, V. M. Trifa, and E. Wilde, *Architecting a mashable open world wide web of things*. ETH, Department of Computer Science, 2010.
[35] P. Balamuralidhara, P. Misra, and A. Pal, "Software platforms for internet of things and M2M," *J. Indian Inst. Sci.*, vol. 93, no. 3, pp. 487–498, 2013.
[36] "Xively by LogMeInXively." [Online]. Available: http://xively.com/. [Accessed: 28-Mar-2015].
[37] "Documentation | DeviceHive." [Online]. Available: http://devicehive.com/documentation. [Accessed: 28-Mar-2015].
[38] "52N Sensor Web Community - 52°North Sensor Web Community." [Online]. Available: http://52north.org/communities/sensorweb/. [Accessed: 28-Mar-2015].
[39] "ThingWorx - Internet of Things and M2M Application Platform | ThingWorx is the first application platform designed to rapidly build innovative Internet of Things and M2M applications." .
[40] B. P. Gerkey and M. J. Mataric, "A framework for studying multi-robot task allocation," 2003.
[41] T. Lemaire, R. Alami, and S. Lacroix, "A distributed tasks allocation scheme in multi-UAV context," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, 2004, vol. 4, pp. 3622–3627.
[42] J. Thelin, "A comparison of service-oriented, resource-oriented, and object-oriented architecture styles," in *OMG Workshop, Munich, Germany*, 2003.
[43] K. Wagh and R. Thool, "A comparative study of soap vs rest web services provisioning techniques for mobile host," *J. Inf. Eng. Appl.*, vol. 2, no. 5, pp. 12–16, 2012.
[44] P. Markey and G. Clynch, "A performance analysis of WS-*(SOAP) and RESTful Web Services for Implementing Service and Resource Orientated Architectures," 2013.
[45] D. Guinard, V. Trifa, T. Pham, and O. Liechti, "Towards physical mashups in the web of things," in *Networked Sensing Systems (INSS), 2009 Sixth International Conference on*, 2009, pp. 1–4.
[46] D. Guinard, I. Ion, and S. Mayer, "In search of an internet of things service architecture: REST or WS-*? A developers' perspective," in *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, Springer, 2012, pp. 326–337.
[47] A. Rodriguez, "Restful web services: The basics," *IBM Dev.*, 2008.
[48] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle, "The Web of Things: interconnecting devices with high usability and performance," in *Embedded Software and Systems, 2009. ICESS'09. International Conference on*, 2009, pp. 323–330.
[49] E. Bozdag, A. Mesbah, and A. Van Deursen, "A comparison of push and pull techniques for ajax," in *Web Site Evolution, 2007. WSE 2007. 9th IEEE International Workshop on*, 2007, pp. 15–22.
[50] M. Fasli, *Agent technology for e-commerce*. John Wiley & Sons Chichester,

2007.

List of Publications

[1] S. Mahmoud and N. Mohamed, "Collaborative UAVs Cloud," in "*Unmanned Aircraft Systems (ICUAS), 2014 International Conference",* Orlando, USA, 2014, pp. 365–373.

[2] S. Mahmoud and N. Mohamed, "UAVs Cloud Computing," in *UAE Graduate Students Research Conference 2015 (UAE GSRC 2015)*, Abu Dhabi, UAE, 2015, pp. 559–560.

[3] S. Mahmoud and N. Mohamed, "Broker Architecture for Collaborative UAVs Cloud Computing," at the "*The 2015 International Conference on Collaboration Technologies and Systems (CTS 2015)",* Atlanta, Georgia, USA, 2015. [ACCEPTED]