Department of Computer Science

Microsoft Research

MSc SSE - Gadgeteer Internet of Things API

Team Members

Pejman Aghili

Marios Constantinides

Sampath A Ramamuniappa

Zheng Gao

Product Owners

Dr. Steven Johnston (Microsoft Research)

Dr. Arjmand Samuel (Microsoft Research)

UCL Supervisors

Dr. Dean Mohamedally

Dr. Graham Collins

September 2013

This report is submitted as part requirement for the MSc Software Systems Engineering degree at University College London (UCL). It is substantially the result of the team's own work except where explicitly indicated in the text.

Acknowledgement

The team would like to express its deepest gratitude to Steven Johnston for providing his exemplary guidance, monitoring and constant encouragement throughout the course of this project. Also, we would like to thank Clare Morgan for providing the team with Gadgeteer hardware equipment.

The team would also like to especially thank Dean Mohamedally and Graham Collins for their excellent supervision and guidance regarding the technical and project management aspects of this project respectively.

Finally, we would like to appreciate the support and cooperation of the MSR1 group during the integration of two projects.

Executive Summary

Internet of Things (Gershenfeld, 1999) is emerging as a reality as we are moving into an era where technology and internet connectivity is becoming ubiquitous and pervasive. Internet of Things has received significant attention from 2008 when Cisco announced that the number of things connected to the Internet has exceeded the number of people on the world (Evans, 2011). Furthermore, Intel Corporation has predicted that the number of these things connected to internet will be 31 billion by 2020. Currently, many cloud based services are offered for Internet of Things, but these platforms do not have native support for the .Net Gadgeteer platform. Therefore, this project is aimed at providing Internet of Things API and services for .NET Gadgeteer prototyping platform to ease the development effort involved in creating IoT based application.

Gadgeteer (Microsoft Corporation, 2011) **Internet of Things API** (Application Programming Interface) is developed with having the vision of creating a network of globally connected devices and providing a rich feature set in the form of APIs to simplify device discovery, device registration, logging sensor derived data and data retrieval functionality for Gadgeteer and other hardware platforms. This project offers well-documented APIs and libraries to developers with medium level programming skills to get them started with developing applications based on Internet of Things.

Microsoft Research has sponsored this project to investigate the use of Microsoft .NET Gadgeteer Platform and analyse the difficulties involved in data management, visualization and privacy issues associated with projects related to IoT and .NET Gadgeteer Platform. The main deliverable of this project includes Internet of Things APIs which provides data logging functionality through cloud-based web service and APIs to retrieve the data for performing big data analytics and visualization. Additionally, another important deliverable of this project is the Gadgeteer library, which provides functionalities like data buffering in case of network connectivity failure, data logging and other library methods to facilitate easy and seamless communication with the cloud services; thus simplifying the process of data management and application development of a hardware developer. Moreover, the deliverables also comprise of some use case scenarios which will be discussed further in section 5.8 of the group report. These use cases demonstrate how to use the APIs and Gadgeteer library to implement different scenarios with ease. Another stand out feature of this project is the intuitive design of the website where a lot of effort has been put to make it as user friendly as possible.

This project contributes to Microsoft .NET Gadgeteer community with APIs and libraries which developers can use to create a network of connected devices with ease; this includes device management, device status monitoring, data management, notification management and library for .NET Gadgeteer platform. A number of students from UCL CS department work on .NET Gadgeteer platform every year and they spend significant time on setting up the infrastructure for gathering data from the hardware devices. This project will help in

addressing those difficulties by using our platform and API to reduce their development effort and let them concentrate more on their application. In addition to .NET Gadgeteer devices, this system also supports a wide range of other hardware devices such as Engduino, Arduino and Android by adopting a generic and scalable architecture using REST. This project is published as an open-source project in Codeplex (Aghili et al, 2013) through which other developers can contribute to this project and also make use of the services and libraries that we have developed.

Contents

1. Introduction 1
   1.1. Project Overview 1
   1.2. Business Case Overview 1
   1.3. High Level Report Overview 2
2. Background Study 3
   2.1. Internet of Things 3
   2.2. Challenges of Internet of Things 3
      2.2.1. Identity Management of Things 3
      2.2.2. Big Data 3
      2.2.3. Internet Connectivity 4
      2.2.4. Privacy and Security 4
      2.2.5. Hardware Prototype 4
      2.2.6. Summary 5
   2.3. Related Work and Existing Systems 5
      2.3.1. Microsoft Product 5
      2.3.2. Microsoft Competitors Products 5
3. Project Requirements 7
   3.1. Project Initiation 7
      3.1.1. Stakeholders Identification 7
      3.1.2. Main Goals 7
      3.1.3. Project Scope 8
      3.1.4. Risks 9
      3.1.5. Assumptions and constraints 10
   3.2. Requirements Elicitation 10
      3.2.1. Interview 10
      3.2.2. User Stories using Persona 11
   3.3. Evaluation and Prioritisation 12
      3.3.1. Evaluation 12
      3.3.2. Prioritisation 13
4. Architecture and Design 14
   4.1. Architecture Overview 14
      4.1.1. Architectural Patterns and Styles 14

4.1.2. Architecture Views 19
4.2. Technology Decisions 22
4.2.1. Azure Cloud Services 22
4.2.2. Windows Communication Foundation (WCF) 22
4.2.3. Databases 23
4.2.4. Hardware Platforms 24
4.3. UI Style Guide 24
4.4. Integration with MSR1 Project 25
5. Implementation and Testing 26
5.1. Brief Description of Components 26
5.1.1. Website 26
5.1.2. Web Service 26
5.1.3. Gadgeteer Library 28
5.2. Use Case Scenarios 29
5.2.1. Common Walkthrough 29
5.2.2. Use Cases Implementation 30
5.3. Software Measurement 31
5.4. Testing Overview 32
5.4.1. Strategy 32
5.4.2. Challenges 33
5.5. White Box Testing 33
5.5.1. Unit Testing 33
5.5.2. Integration Testing 34
5.6. Black Box Testing 35
5.6.1. System and Functional Testing 35
5.6.2. Performance Testing 36
5.6.3. Load and Stress Testing: 37
5.7. Static Analysis 38
5.7.1. JavaScript analysis with JSLint 38
5.7.2. Static Analysis with Code Analysis (VS 2012) 39
6. Project Management 40
6.1. Software Development Methodology 40
6.2. Sprint Planning 40

6.3. Task Distribution and Management 42
7. Project Results 44
7.1. Achievements 44
7.2. Comparison with Existing Systems 45
7.3. Critical Assessment 46
7.4. Further Work 46
8. Conclusion 47

List of Figures

Figure 1 Project Stakeholders 7
Figure 2 Project Scope 9
Figure 3 Personas 11
Figure 4 Comparison of WS* and REST for IoT applications 16
Figure 5 Web Service architecture using REST and Message Bus 17
Figure 6 Publish-Subscribe design pattern 17
Figure 7 Strategy design pattern 18
Figure 8 API façade pattern 18
Figure 9 Observer Pattern 19
Figure 10 Enterprise Architecture View 19
Figure 11 Layered Architecture View 20
Figure 12 IT System View 21
Figure 13 Integration with MSR LoT Analytic Engine 25
Figure 14 Web Service Components 27
Figure 15 Walkthrough of the User Registration and Device Addition 30
Figure 16 Software Metrics for Gadgeteer Library and Web service API 32
Figure 17 Testing Work Flow Process 32
Figure 18 Coverage Results of Web Service Components 35
Figure 19 Response Time for Data Logging API 36
Figure 20 Nth Percentile 'Response Time' comparison between data logging and data retrieval API 37
Figure 21 Load Testing Result for Data Logging API for varying load with simulated virtual users 38
Figure 22 Static Analysis Result with JSLint 38
Figure 23 Code Analysis Result for Web Service 39
Figure 24 Backlogs in TFS 43

List of Tables

Table 1 Primary risks for the project 10
Table 2 User stories 13
Table 3 Software Metrics 31
Table 4 Code Coverage Results of Data Access Layer Components 34
Table 5 Sprint Planning 41
Table 6 Scrum Roles 42
Table 7 Comparison of Gadgeteer Internet of Things API with Lab of Things 46

1. Introduction

This report outlines the problem statement, background study, existing as well as prospective solutions, software engineering practices that the team followed, important architectural decisions and some of the implementation challenges that our team faced during the course of this project. Our team had comprehended the knowledge gained through this 'MSc Software Engineering' programme and followed the various aspects of software engineering discipline throughout the course of this project to build a high quality software product. We adhered to scrum project management and standard software engineering practices while working on this project.

1.1. Project Overview

Internet of Things (see section 2.1) has been around in the internet world for a while and possesses endless opportunities. But these opportunities come at the cost of overcoming several technological challenges. This project has been sponsored by Microsoft Research for creating APIs to solve some of the existing challenges (see section 2.2) in realising the vision of Internet of Things (IoT). Furthermore, our project specification also promotes the inclusion of Microsoft's .NET Gadgeteer Platform for integrating with the API and help developers to exploit the benefits and developments in the IoT arena.

We followed the requirements engineering techniques (see section 3.2) to identify the goals and the requirements of the system. Some of the main objectives of this project includes creating Web APIs for connecting hardware devices to the Internet, establishing a platform for device management, addressing privacy issues associated with IoT, enabling communication between things in the Internet and delivering a generic solution to support a wide range of hardware devices apart from Gadgeteer. In addition, the objectives also include some non-functional requirements to make this project extensible and scalable to provision future improvements to this system. In addition to addressing the above requirements, this project also investigates the implementation of out of the shell features which is not in the existing systems. Some of the features are device discovery, custom data repository and notification management.

1.2. Business Case Overview

IoT is a truly empowering concept which can have a major impact on society and business. Currently, most of the leading technology companies such as Microsoft, IBM, Vodafone, Intel and Ericsson have IoT related research projects in their labs. Cisco (2013) has predicted that Internet of Things has a potential of $14.4 trillion in value at stake. Few examples of business use cases include efficient resource monitoring, usage pattern tracking, smart power grids, water management, and smart transport and traffic control, waste and recycling management.

Microsoft has already ventured into this field and has developed hardware prototyping platforms, home automation software and cloud services which take advantage of low cost

sensors to easily build connected devices. The main business use case of this system is to build an IoT cloud service platform which can be used by .NET Micro Framework (NETMF) developers to easily gather data from Gadgeteer devices and able to implement various use cases. Thus the project's business value is the promotion of Gadgeteer Platform by demonstrating that it is useful for IoT by taking advantage of rapid prototyping it offers.

In addition, this project includes some use case scenarios to explain the use of APIs to achieve business objectives. First use case focuses on IoT in education by demonstrating how students can use this system to conduct small scale scientific experiments such as temperature, humidity and atmospheric pressure logging and visualization.

Moreover, IoT has an endless impact on the society in numerous ways; to support this claim our project includes a use case scenario which is developed in collaboration with Met Police to ensure safer neighbourhood. These use cases are discussed further in section 5.2.2. Furthermore, this project introduces new features and concepts that can be used or integrated to extend the functionality of Lab of Things.

1.3. High Level Report Overview

This project work includes one group report and two supporting documents. The group report discusses the background study, need and purpose of the project, design and implementation of the solution, achievements and further work along with the project management aspects of the project. The first supporting document (Appendix A) lays down the user manual, technical user guide, a brief description of Gadgeteer library classes and how to use the library. Moreover, it documents the exposed APIs along with a brief description of how to invoke them. The second supporting document (Appendix B) solely concentrates on the testing and contains proper documentation of all the test cases that were created and executed. On the other hand, the results of testing are discussed in section 5 of this report.

2. Background Study

2.1. Internet of Things

"The Internet may already be huge, but it's about to get a lot bigger." (Intel Corporation 2011)

The internet has grown rapidly above all our expectations and has been evolving ever since it began as ARPANET in 1969 (Abbate 1994). Intel Corporation (2011) suggests that today internet has around 15 billion devices connected to it and estimates show that it is expected to have 31 billion devices by 2020. Internet of Things is a new evolution of the Internet and underpinning its development is the ever increasing proliferation of networked 'smart' objects to the Internet. A 'thing' is any physical object which can be given a unique internet address and is able to transfer data over the Internet. As more objects are being embedded with sensors and gaining the ability to send and receive data via the internet, these 'things' can make themselves recognizable to others and acquire intelligence by accessing information aggregated by other things.

2.2. Challenges of Internet of Things

Although IoT promises new business models, and reduce cost and risk; a lot of technical challenges have to be solved in realising that dream. Some of the existing challenges are briefly discussed in this section.

2.2.1. Identity Management of Things

It has been argued by Khan et al. (2012), Coetzee and Eksteen (2011) that one of the main features of IoT is the large number of things being connected to the internet. Therefore, as a fundamental requirement, an identity management system is vital in order to identify each thing across the Internet exclusively. These unique identities will also facilitate the bidirectional communication within IoT. For this purpose, IP addresses can be used to identify and communicate with these things. Due to the sheer number of these things in the internet, IPV6 seems to be the ideal solution which supports 128-bit addresses instead of 32-bit in the current addressing policy, IPV4 (Atzori et al, 2010). Though IPv6's huge address space is a great enabler of IoT, it is still a distant dream as there are a lot of challenges involved in migrating from IPv4 to IPv6. So an alternative way is needed to assign public static IP addresses in IPv4 to these things using for communication.

2.2.2. Big Data

IoT produces a tremendous amount of multisource and heterogeneous data from all the connected sensors and devices (Chen 2012). Lawson (2013) describes this data as mega big data and relates it to the challenges of big data. Therefore, the main dimensions of big data should be considered throughout the design and implementation phase.

1. Volume: As Lawson (2013) discusses, in IoT the amount of data that needs to be managed is extremely high. This requires a scalable and robust infrastructure.

2. Variety: In IoT the data is generated from different devices and sensors. Therefore, the data is in different formats and types. Thus, the architecture of the system should be very generic to support them all.
3. Velocity: the frequency of the data generation in IoT is extremely high. Therefore, the right tools and architectural patterns should be used to meet the high velocity of incoming data-- all in real time.

By considering these dimensions, we can conclude that the generated data in IoT does not fit in standard relational databases and alternatives should be considered. In addition, as discussed by Coetzee and Eksteen (2011), we should also consider how we can make this data useful for future use such as big data analytics. Otherwise, as Zhou et al. (2013) discuss, IoT offers insignificant useful benefits if it not able to produce any meaningful information from such great amount of data. The MSR1 group (Khandelwal et al. 2013) focuses on this challenge to extract knowledge from the raw data generated by IoT.

2.2.3. Internet Connectivity

Due to various network related reasons, the connection between the connected device or sensor and the Internet can disrupt at any certain point. Therefore, a unique mechanism is needed to buffer the data locally and send it to the Internet when the connection is established again. This is especially important in data sensitive applications which require a continuous stream of data.

2.2.4. Privacy and Security

Another inherent challenge with IoT is the security and privacy issues associated with data. In the domain of security, most of the communication channels are wireless, thus the transferred data can be simply eavesdropped. Additionally, because of the limits in energy and computing resources, developers cannot implement complex schemes, such as data authentication and access control, which normally support security on lightweight components. In the domain of privacy, IoT is in desperate need of specific technologies which can exert strict access control over the personal data. What is more, relevant protection laws also should be enacted (Bandyopadhyay et al, 2011).

2.2.5. Hardware Prototype

Also, there are several issues arising in the hardware development field to support IoT. Considering that the implementation environment for hardware-based application is still immature, it is fairly difficult for a developer to quickly build prototypes, thereby preventing IoT from becoming widespread. Moreover, standards for communication between electronic devices have been established, such as Constrained Application Protocol (CoAP) which aims to optimize the use of the RESTful web service architecture in constrained nodes and networks (Kuladinithi et al, 2011). However, little has been implemented for the various platforms of prototyping hardware including Gadgeteer, Arduino and so forth. The fact directly results in the lack of abstraction among devices from different manufacturers and the increasing complexity of integrating with these devices.

2.2.6. Summary

Facing the major challenges in realising IoT, we have managed to address some of the issues associated with privacy, internet connectivity, identity management of devices and hardware prototyping by making use of existing technologies and providing a robust solution in building a scalable Internet of Things application. This will be discussed more in detail in section 4 and 5.

2.3. Related Work and Existing Systems

As it was discussed in the previous section, in order to understand the difficulties of IoT in real world scenarios, we studied some of the existing systems that provide IoT based services. We tried to learn from these systems and designed our system in such a way to complement the existing systems and also tried to address some of the outstanding issues with these systems. For this purpose, we have chosen three different systems to discuss in this section, which provide different kind of IoT Services.

2.3.1. Microsoft Product

2.3.1.1.Lab of Things

Few weeks ago, Microsoft introduced Lab of Things (Beta Version 1), a new research platform to support real-world data in the cloud as Alex Wilhelm (16th July 2013) stated in his article. LoT is a research-device platform that boosts Microsoft's HomeOs on the Azure Cloud. In a nutshell, it allows researchers to get better field studies and conduct experiments on connected devices in homes by pushing all the gathered data to the cloud. It just simplifies the data logging and data management by maintaining a central repository that is easily accessible via Azure Portal. Digging into more depth of what LoT provides, we can discover a variety of features such as writing drivers for HomeOS to collect data from houses and store it in azure storage account. Taking into consideration that LoT is only available to Academia, someone could count it as a big drawback and wonders why Microsoft doesn't push it out of the academic realm. However, through all its capabilities, it lets you interconnect devices and scale up your field studies in diverse experiments and locations. Finally, it is worth pointing out that there are numerous of researchers among top universities that use Lab of Things (Microsoft Research Lab of Things) including UCL.

2.3.2. Microsoft Competitors Products

2.3.2.1.Xively

Xively (formerly Pachube) is a cloud platform that offers a commercial ground for connecting devices, or even better, letting sensor-derived data such as energy consumption, temperature, humidity and GPS, to be attached to the Web. It provides RESTful APIs that allow developers to build application based on the gathered data.

On the upside, Xively provides a broad range of features such as support for different software and hardware platforms, which establishes Xively as an open platform and offers end-to-end security across the entire platform. In addition, it provides frictionless developer

experience through web-based tools that simplify the complexities of IoT development. Moreover, it maintains a global IoT cloud infrastructure which leverages external cloud services such as Facebook and Twitter by enabling connectivity with them.

On the downside, Xively does not provide easy access to raw data as well as the data visualization is arguably poor. Despite the fact that Xively has an easy to use RESTful API, it does not include flexible visualization dashboards or a processing engine out of the box. Especially, the main drawback of Xively is the data policy, i.e. whatever data you publish on Xively's data cloud is open and thus is accessible from anyone. If we fall back to 2009 Richard MacManus (2011) described how its business model was based on the willingness of users to pay for privacy features.

2.3.2.2.IFTTT

IFTTT developed and launched in 2010 by Linden Tibbets with the aim to put the Internet to work for you. It is a service that enables people to create powerful connections that fit in one simple statement, the "if this then that". Basically, for those who have a Computer Science background IFTTT executes a "simple" IF-STATEMENT. In order to get more clear understanding how it works we should turn in to more technical aspects. The "this" part is the trigger whereas the "that" part is the action. Therefore, whenever something satisfies the trigger condition then the action is executed. An example that shows IFTTT's functionality could be the following one. Imagine you are tagged in a photo on Facebook and you have set your trigger to be Facebook photo tagging, then the action could be sending an SMS or updating the Facebook status. Few interesting statistics about IFTTT emerged and show its success through the 400,000 tasks that have been created since its launch (IFTTT Blog).

3. Project Requirements

3.1. Project Initiation

Project initiation is the initial and an important step in a project's lifecycle. It was conducted during the sprint 0 to define the project's vision and to study about the project's viability. This section discusses the different activities performed during project initiation.

3.1.1. Stakeholders Identification

The stakeholders have been categorised in 3 different areas; Operational Area, Containing Business and Wider Environment. This is based on the Onion model from requirement engineering practices (see Figure 1).

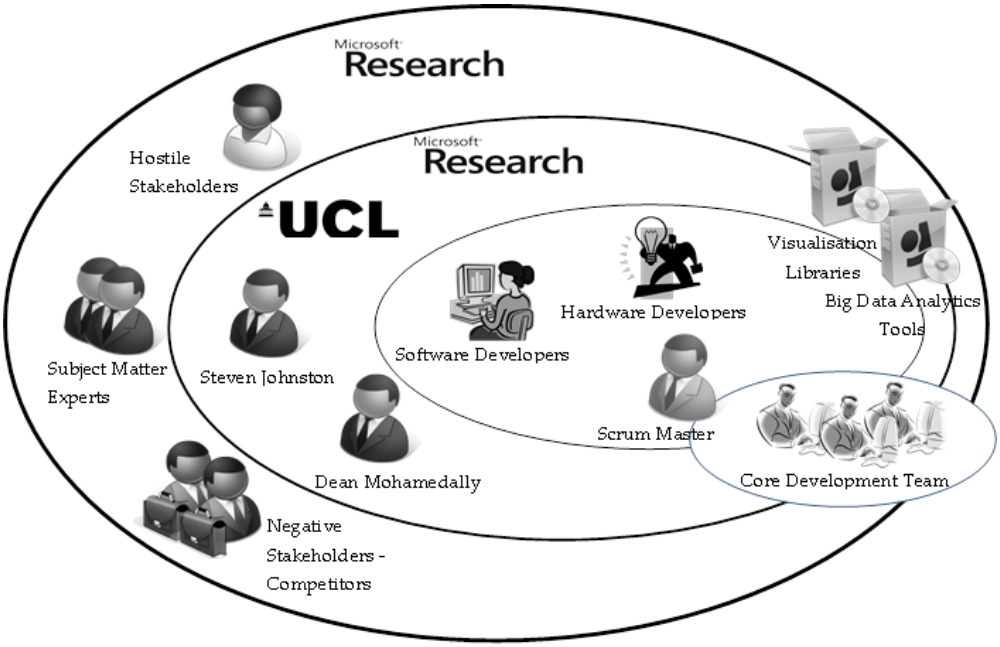

Figure 1 Project Stakeholders

Hardware and Software developers are the main users of the system.

- **Hardware Developers:** They use the Gadgeteer library to speed up their prototyping process for Gadgeteer platform. For other hardware platforms they can use the API directly to log data to the cloud.
- **Software Developers:** They use the API to retrieve the data generated from devices to create applications

The complete list of the stakeholders and their stakes can be found in Appendix A.

3.1.2. Main Goals

The team has followed KAOS goal modelling to identify and define the main goals of the system along with their objective functions.

Goal Achieve [Simplified Data Management]

Definition

Develop a web API which allows developers to log and retrieve data generated by the devices.

Objective Functions

- Accept data from dynamic data sources such as temperature, humidity and GPS.
- Be able to retrieve logged data within 0.5 second.
- Be able to store the data either in the cloud or in a remote repository based on user preference.

Goal Achieve [Hardware Platform Abstraction]

Definition

The API should be generic so that it can be used not only by Gadgeteer but also by any other hardware device.

Objective Functions

- Support majority of the hardware platforms such as Android, Arduino and Engduino and create use case scenarios to demonstrate it.

Goal Achieve [Simplified Hardware Prototyping]

Definition

Enhance developer experience by reducing the complexity for programming a Gadgeteer device to communicate with the API through a documented library

Objective Functions

- Reduce the required number of lines of code to communicate with the API by using library methods.
- Reduce the debugging time by using fully tested library methods.

3.1.3. Project Scope

Different hardware device can use the generic API to log their data into different repositories. However, hardware developers who are using Gadgeteer platform for prototyping can use our lightweight library in order to speed up their development process. In addition, the API can also be used by software developers to create IoT based applications such as health care applications. Outside the scope of this project, there are third-party libraries and analytics tools which can retrieve the data through the API and use it according to their needs, such as tools that perform big data analytics.

The figure (see Figure 2) below shows the scope of this project and its interaction with other systems and users.

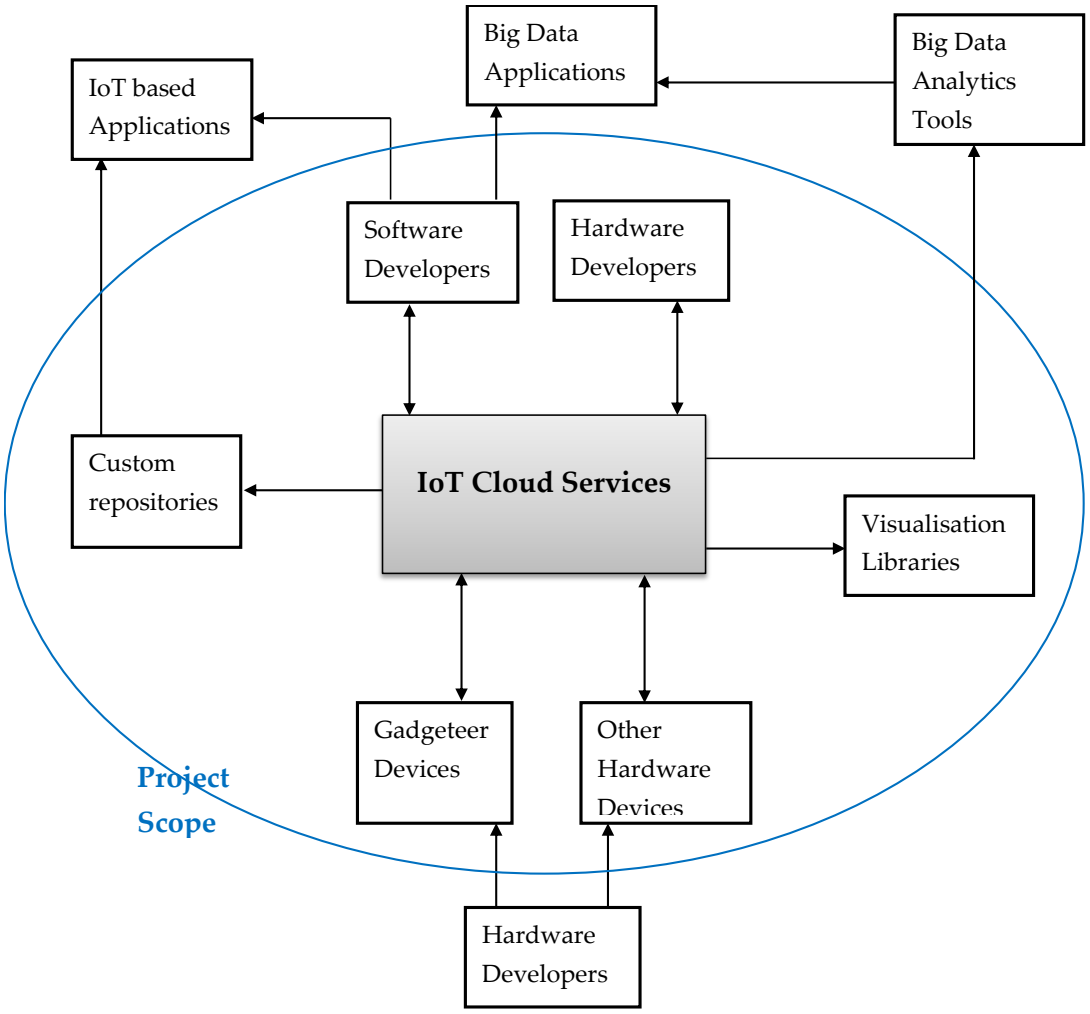

Figure 2 Project Scope

3.1.4. Risks

During project initiation the team identified the following top level risks associated to this project (see Table 1).

Risk Name	Risk Description
Azure Student Accounts	The project is hosted on Microsoft Azure Platform using student accounts. According to Microsoft Azure policies they can expire at any time.
Azure Cloud Infrastructure Failure	The project is mainly hosted on Azure. Therefore, when there is a failure or maintenance in Azure infrastructure our service is exposed to unavailability.
.NET Micro Framework	Gadgeteer is the main prototyping platform for this project. However, the Micro Framework library for this platform is still

(RTM) 4.3 Instability	buggy and there are some known and unknown issues.
Third-party Libraries Bugs	Since the project uses third-parties libraries such as high chart and pusher, the project is exposed to defects from the existing bugs of those libraries.

Table 1 Primary risks for the project

For the complete list of the risks and their associated resolutions please refer to Appendix A.

3.1.5. Assumptions and constraints

The communication between the hardware devices and the API is done through HTTP protocol. Hence, an assumption has been made that all the hardware devices can support HTTP request-response protocol.

Additionally, Gadgeteer is used as the main hardware platform for this project. Therefore, Assumptions have been made that data from Gadgeteer sensors are accurate and reliable and they do not produce unexpected data.

Azure accounts given by UCL only allow two cores per account. Our project completely relies on different cloud services such as Virtual Machines and Worker Roles requiring 8 cores in total, which is serious a constraint to this project. Therefore to overcome this constraint, services and virtual machines were distributed among various azure accounts which in turn affect the overall performance of the system.

3.2. Requirements Elicitation

3.2.1. Interview

Since the project description was really abstract, we held two face-to-face interviews with our product owners to understand the project in more depth.

The first interview was held with Dr. Steven Johnston to get the requirements for this project. He provided us with the general requirements and the expected approach that needs to be taken. However, he did not provide us any specific requirement and encouraged us to implement new ideas as long as the project fulfils the below general requirements.

General Requirements:

1. An abstract API should be created which can work with devices from different hardware manufacturers
2. Users' privacy of data should be considered
3. The API should not only transmit raw data. But, it should perform some processing on the data
4. A lightweight Gadgeteer library should be created to help the hardware developers with their implementation
5. Provide data visualisation by using third-party libraries

6. Provide a continuous stream of data by buffering the data locally when there is no internet connection
7. Use Cases
    - **Use Case 1:** High school students in a biology class logging temperature and humidity
    - **Use Case 2:** Specified by Dr Dean Mohamedally (using Gadgeteer)
    - **Use Case 3:** SDK Example use case of swapping to another prototyping platform (Engduino)

The second interview was held with Dr. Dean Mohamedally to get the requirements for a real world safer neighbourhood use case. His main requirement was to transmit live data from a device in an emergency situation. An example that he mentioned was to send the GPS coordinates of a bicycle when it goes outside a certain area to the web service and show the coordinates on the map (event based triggers). However, he also allowed us to come up with new ideas for this use case by considering assisting neighbourhoods with security and personal safety. Section 5.2.2 describes how we combined the requirements of our product owner with our ideas to create an innovative use case scenario.

3.2.2. User Stories using Persona

In order to tackle the challenges of wide scope, deal with multiple stakeholders and identify specific project requirements in the given time limit; the team decided to use an innovative user-centred approach known as persona to create user stories.

As it was discussed by Siricharoen (2012) and Calabria (2004), a successful product should consider the needs of the future users of the system throughout the development process and the features should be developed based on different types of end users.

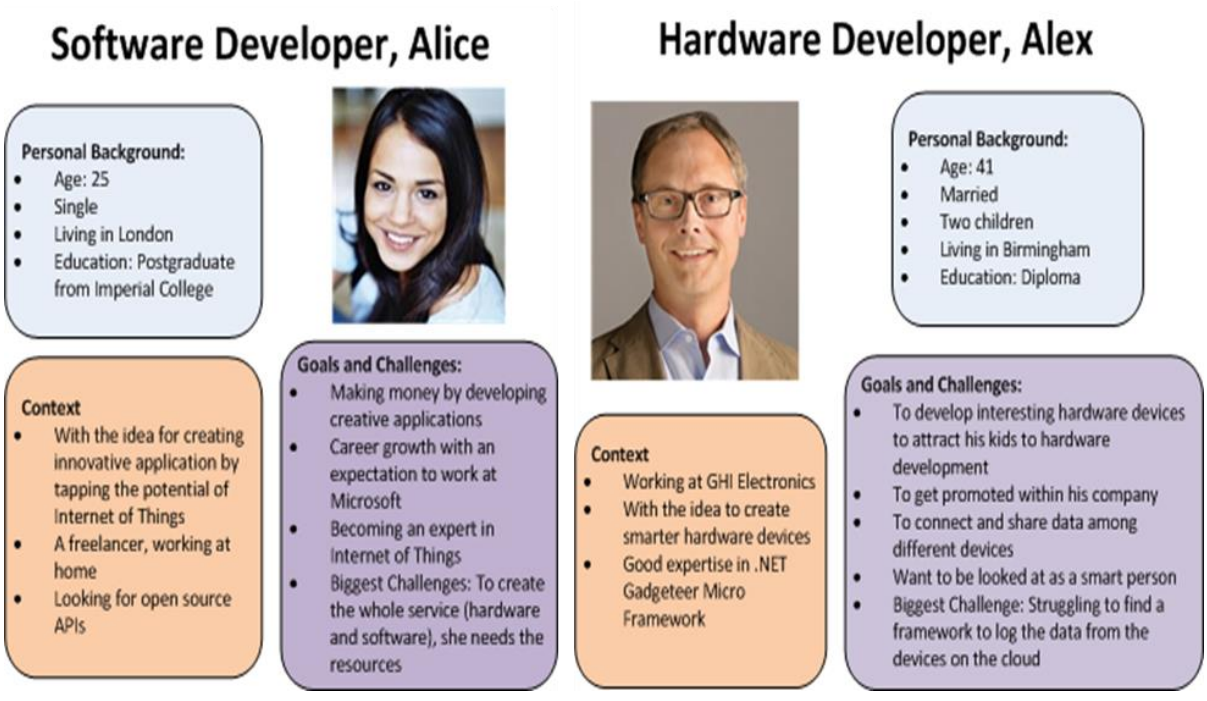

Figure 3 Personas

As Calabria describes, Personas represent archetypical users of the future system and embody the needs and goals of larger groups of people who will use the system. They enabled the development team to stand in users shoes and focus on one type of users at a time in order to come up with their requirements. Personas can be created fast; this characteristic makes this technique suitable for this project because of the short development time. For this purpose, two personas were created to represent the main users of the system. Alex represents hardware developers and Alice represents software developers. The Figure 3 describes the characteristics, main goals and challenges for Alex and Alice.

3.3. Evaluation and Prioritisation

3.3.1. Evaluation

By using Alex and Alice as our personas and RAG format (Role-Action-Goal) different user stories were created. However, as Leffingwell (2010, p.105) suggests , user stories should be evaluated against I(Independent) N(Negotiable) V(Valuable) E(Estimable) S(Small) T(Testable) features. The following table shows the identified user stories after applying the evaluation model.

ID	Title	Persona Name	User Story
1	Device and Sensor Registration	Alex	**As a** hardware developer, **I want** to register my devices and sensors **so that** they will have unique identifiers
2	Device Management	Alex	**As a** hardware developer, **I want** to have the ability to manage my devices **so that** I can update or remove them at any time.
3	Data Logging	Alex	**As a** hardware developer, **I want** to log data from my devices **so that** I can extract meaningful information from the data.
4	Custom Repository	Alex	**As a** hardware developer, **I want** to specify the repository to be used to store my data **so that** I can have more control over my data.
5	Data privacy	Alex	**As a** hardware developer, **I want** to set the privacy of my data for each device **so that** not everyone can have access to my data.
6	Data Sharing	Alex	**As a** hardware developer, **I want** to share my devices with my friends **so that** they can have access to my data.
7	Notifications	Alex	**As a** hardware developer, **I want** to set triggers for my devices **so that** I get notifications when they are triggered.
8	Data	Alex	**As a** hardware developer,

	Visualisation		**I want** to see my data visualised on a graph or map **so that** I can monitor the data easily.
9	Real Time Data Transmission	Alice	**As a** software developer, **I want** to get data from different hardware devices at real time **so that** I can gather data from different sensors and create applications based on them.
10	Historic data Retrieval	Alice	**As a** software developer, **I want** to have access to historic data of different devices **so that** I can gain knowledge by processing that data later.
11	Device Discovery	Alice	**As a** software developer, **I want** to discover all the devices that I can subscribe to **so that** I can get data from those devices and switch between them in my application.
12	Continuous Data Stream	Alice	**As a** software developer, **I want** a continuous stream of data **so that** my application does not suffer from missing data.

Table 2 User stories

3.3.2. Prioritisation

Effort estimation using Planning Poker

The widely used technique for the effort estimation in Scrum is planning poker where all members in the team are involved to reach to a consensus about effort for each user story in the product backlog. As Hartman (2009) described the advantages of using planning poker, the team also followed this technique by using the widely practised method of giving Fibonacci numbers (1, 2, 3, 5, 8, 13, 21, 34...) to estimate the effort for each user story. We decided to have 2 as least and 34 as the highest effort needed to complete a user story. For each user story, every member in the team used cards to write the effort needed to complete it. Afterwards, the team discussed the estimated efforts to come to agreement about the effort for each user story.

Business value / effort ratio

As Maurits (2011) discusses, prioritisation based on the ratio between the business value and the implementation effort (low effort, high value) provides the highest business value for the product owner within the least number of sprints.

By getting the business value through interviewing the product owner and estimating the effort by using planning poker, the team prioritised the user stories in the backlog throughout the development process.

4. Architecture and Design

4.1. Architecture Overview

Having done Advanced Analysis and Design as a core course in our master degree the team has developed a deep understanding regarding the importance and the benefits of having extensible and robust software architecture. Architecture and design decisions form an important aspect of any product development since they affect the success and acceptance of the end product. In this section we describe the architecture of our solution and discuss some of the important architectural decisions that have been taken and the reasons for making those decisions in order to address the challenges discussed in section 2.2.

4.1.1. Architectural Patterns and Styles

"An architectural style is a set of principles that lay down a coarse grained pattern that provides an abstract framework for a family of systems" (Garlan and Shaw, 1994). An architectural style helps to solve frequently recurring problems by promoting design re-use. We analysed the characteristics and benefits of the wide range of architectural styles (Meier et al. 2013) and followed the guidelines laid out by Bauer et al. (2012) for building a concrete architecture using the Architecture Reference Models (ARM) established for IoT.

4.1.1.1.Combination of SOA and Message-bus Architectural Styles (Mathew, 2007)

Trifa et al. (2010) and Spiess et al. (2009) suggest that the usage of SOA based services is appropriate for IoT since it reduces cost and effort by reusing real world services for different business situations. Adopting this argument, we have developed SOA based web service and a brief description of the services offered by this system is discussed in section 4.1.2.2. Though we have developed SOA based web service, the project's general requirement demands the architecture to be more scalable to support more use case applications which could plug in to this system effortlessly. Meier et al. (2013, p17) argues that building a complete application cannot be limited to just one style of architecture and recommends combining different styles to meet the objectives of a system. Therefore we have decided to combine SOA with message-bus architectural style (Mathew, 2007).

"A Message-bus architectural style describes the principle of using a software system that can receive and send messages based on a set of known formats, so that systems can communicate with each other without having to know the actual recipient of the message" (Meier et al. 2013, p19). An advantage of this style is that it is extensible, i.e. different applications can be added to the bus in the future without affecting the existing applications. This architectural style fits our requirements and likewise IoT in general includes different types of applications which process the sensor derived data according to the changing business needs.

In message bus style, usually all the applications are connected to a common message bus where applications interact through messages over the bus. Since this style provides a common interface through which all the devices can send and receive messages, it reduces

the complexity of sharing information generated by different devices. In addition, this architecture provides good performance due to the absence of intermediaries between communicating applications.

Windows Azure Service Bus

We use Windows Azure Service Bus as the common message bus which was discussed in the previous section. Moreover, Vasters (2012) from Microsoft has also demonstrated how the service bus is scalable for Internet of Things and how it can handle 250,000 devices and around 0.25 – 1.5 million events per hour for at least the initial ramp up target. The Service Bus has "brokered" messaging capabilities that support asynchronous, or decoupled messaging features for publish-subscribe and load balancing. The bus also provides a secure network service gateway for messaging which addresses the security requirements of the project. To be more specific, we use Azure Service Bus Topics of the service bus in this project. Topics provide the benefits of queues and in addition implement the publish/subscribe pattern. A topic can have many subscriptions (currently supports 2000 subscriptions for a topic, but can be increased by Auto-Forwarding) and these subscriptions have filter rules which filters the published messages and sends a separate copy to each of the subscriptions with matching rules. Azure Topics are a useful messaging solution for broadcasting messages to many consumers (Salvatori, 2013). As discussed in the earlier section, the extensibility requirement is addressed by adding more subscriptions to this service bus.

4.1.1.2.REST Architectural Pattern

REpresentational State Transfer (REST) is an architecture principle in which the web services are viewed as resources and can be uniquely identified by their URLs (Fielding, 2000). The fundamental characteristic of a RESTful Web service is the explicit usage of HTTP methods to denote the invocation of different operations offered by the web service.

As discussed in section 3.2.1, one of the general requirements of our project is to make our web service generic and scalable enough to support a wide range of devices. Trifa et al. (2010) discuss about the usage of SOA based Web Services for IoT and recommends the use of web service standards WS*. Whereas Guinard, D. (2011) study compares REST and WS* for IoT applications in different domains and the quantitative results (see Figure 4) indicate that REST is easier and better to be used for embedded devices due to various constraints such as memory, processing power in low cost devices.

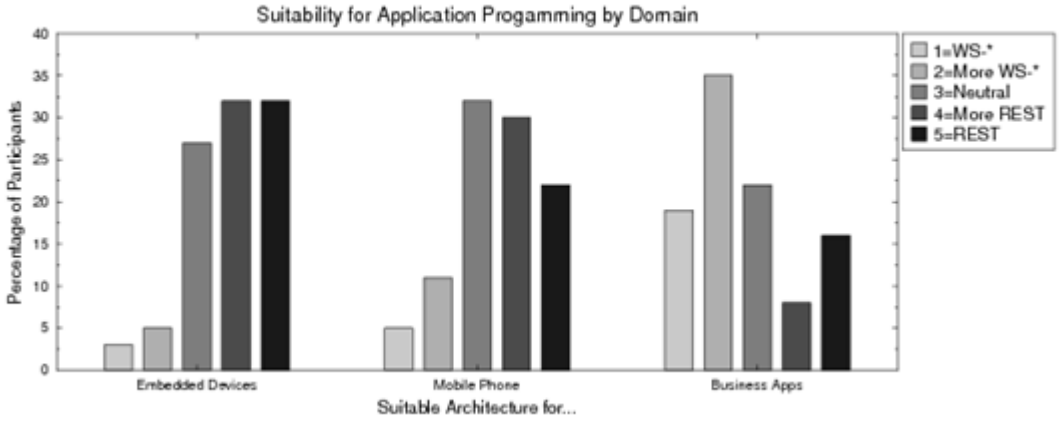

Figure 4 Comparison of WS* and REST for IoT applications

Furthermore, REST is considered to be generic since it is based on HTTP and most of the devices support the above HTTP methods (see section 3.1.5). Hence, we decided to make our Web Service to be RESTful in order to support a variety of hardware devices and prototyping platforms.

The web service was designed to be RESTful and at the same time include the message bus for exchanging messages between different applications. Although the devices can send data directly into Windows Azure Service Bus Topics instead of sending to a REST URI, the memory constraint of small hardware devices make it harder to initiate a straight HTTPS connection to the service bus where even HTTP can sometimes become heavy. Thus, we have utilised REST endpoints in combination with WCF services to receive events and data from the devices. Afterwards we transform the message from the device to a brokered message and push it into the service bus topic. The Figure 5 shows a WCF service which receives data from various devices and pushes it into an azure topic, which then has two subscriptions. One of these subscriptions is responsible for storing data in the database and another for pushing real-time data to the subscribed users. Hence, we have achieved the publish/subscribe functionality using azure topics. Moreover, the service bus also has a buffering ability which results in the additional advantage to buffer incoming data during components failure or scheduled maintenance.

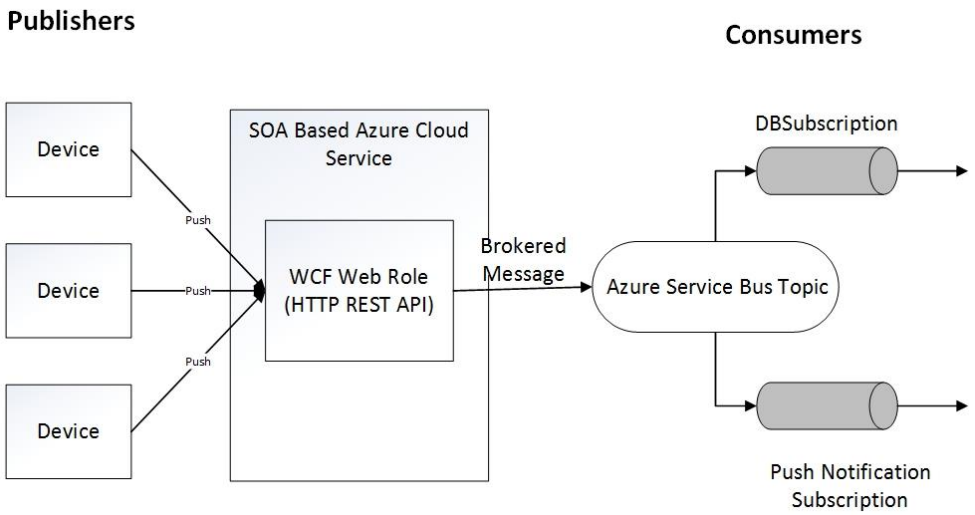

Figure 5 Web Service architecture using REST and Message Bus

4.1.1.3.Design Patterns

Design patterns have been great enablers for our project to deliver a highly scalable, maintainable and reusable system. The team adopted different design patterns to address well-established problems, which will be discussed further in this section.

Publish-Subscribe

The publish-subscribe messaging pattern fits in our architecture due to its loose coupling between publishers and subscribers and its scalability compared to the traditional client-server paradigm. Publishers could be any hardware devices pushing data into the data logging API, whereas subscribers are any third-party applications that subscribed to receiving updates-messages regarding to the context of the publisher. In our case publish-subscribe pattern has been implemented in the live feed of data into IoT website. The website subscribes to HTML5 Web Socket Channel and the pusher worker role is responsible for delivering publishers' messages on the fly.

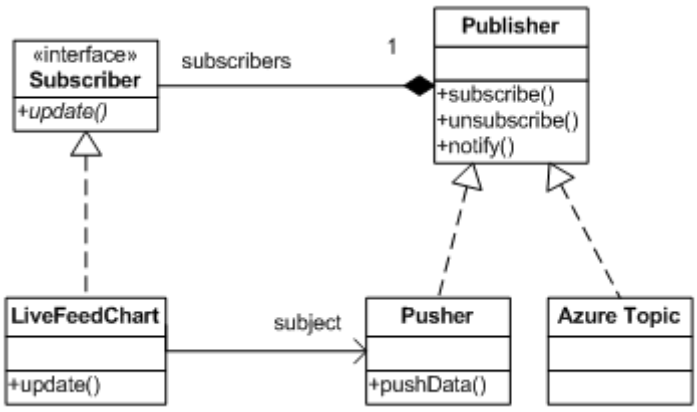

Figure 6 Publish-Subscribe design pattern

Strategy

A feature that distinguishes our project from similar ones is the flexibility provided for users to specify their own custom repositories. Obviously, this functionality indicates a definition of a family of algorithms which are interchangeable and encapsulated with each other. Hence, the strategy pattern has been utilised to switch between the user's custom repository and IoT Cloud Database based on his/her preference. Furthermore, another example of the usage of the strategy design pattern is dynamic routing to the nearest device in case of the device failure. As RTMA keeps track of the device's status, it can identify the disconnected device(s) and report the failure to the subscribed users and applications.

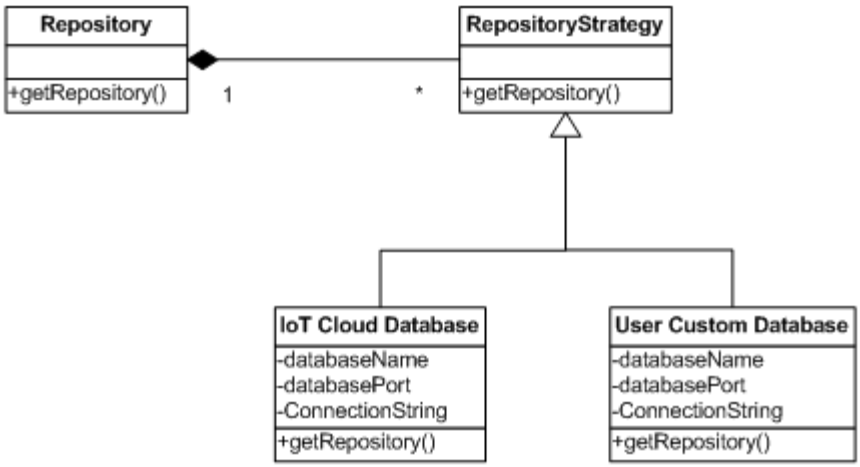

Figure 7 Strategy design pattern

Façade

A façade pattern allows designing a unified interface and provides an easy-to-use capability by hiding the complexity of the system and its subsystems behind the interface. Since the main deliverable of our project is a set of APIs, it was required to adopt this design pattern and moreover we used the guidelines for designing API Façade Pattern (Mulloy, 2012). Data Logging API provides a single interface, but hides the complexity of two different interfaces exposed by the pusher and Database worker roles. Thus a unified interface can be used to access to multiple sub systems based on the context of the incoming message.

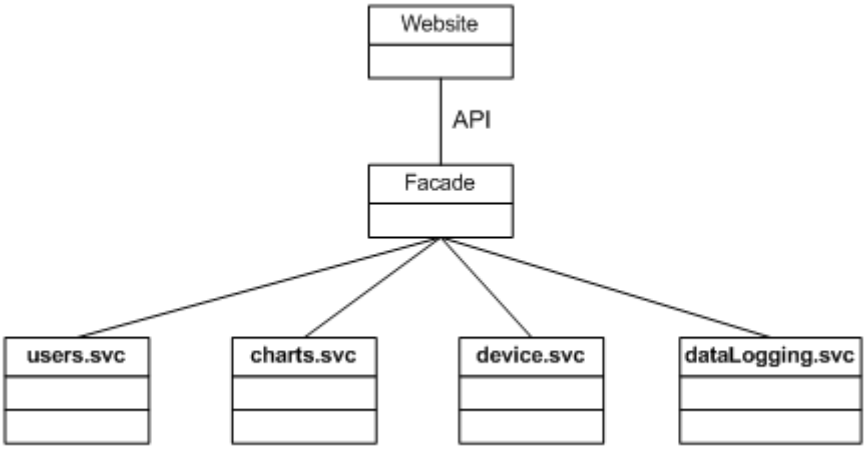

Figure 8 API façade pattern

Observer

This is a behavioural design pattern which delegates the change in data to all observers. This design pattern has been used by RTMA for observing the state of a device as offline or online and updating this information to show the current status of devices on a live chart, device info page and notification page. All the observers will be notified when the status of the device changes.

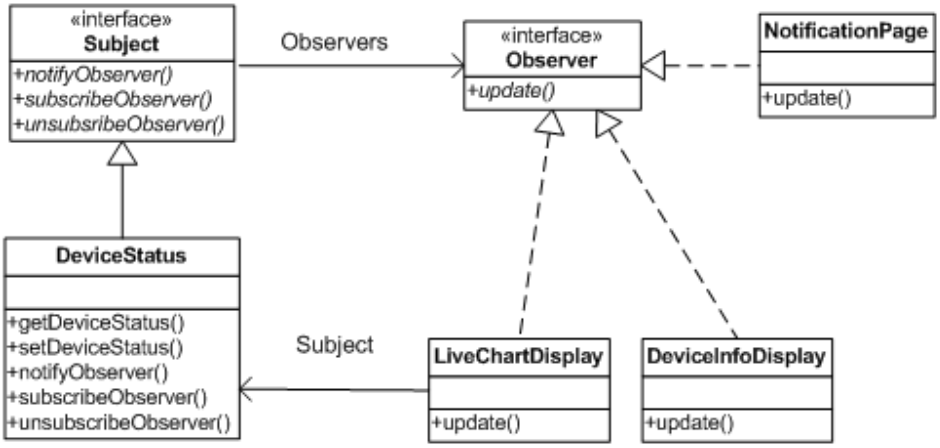

Figure 9 Observer Pattern

4.1.2. Architecture Views

4.1.2.1.Enterprise Architecture View

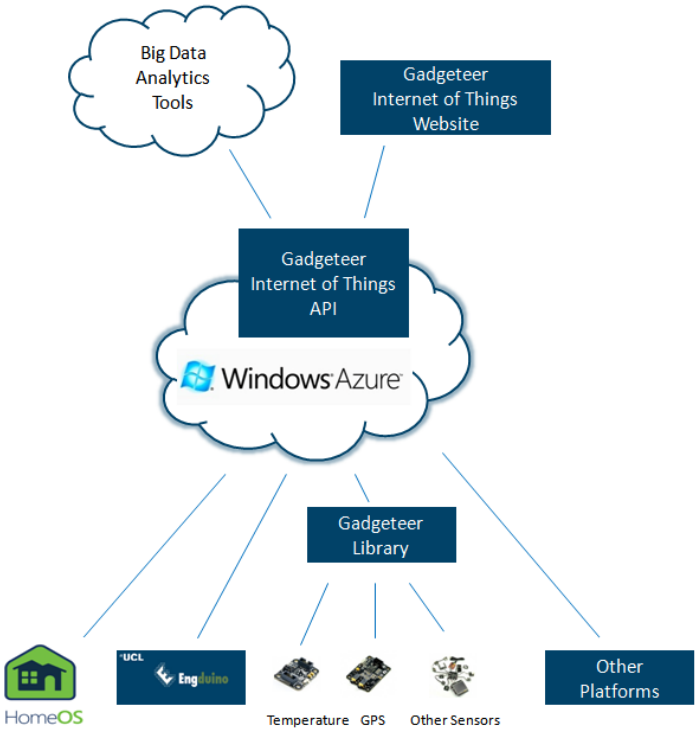

Figure 10 Enterprise Architecture View

The system is designed to be platform independent. The concise and well-defined RESTful API which is hosted on Microsoft Azure Cloud (see Section 4.2.4) makes the system extendable in the maximum extent. The Gadgeteer Internet of Things website and any other systems such as data analytics tools can easily interact with the API because of its simplicity and scalability. Taking the hardware's point of view, the Gadgeteer Library offers a lightweight communication with the API as well as the API is generic enough to be used by other hardware platforms such as HomeOS and Engduino.

4.1.2.2.Layered View

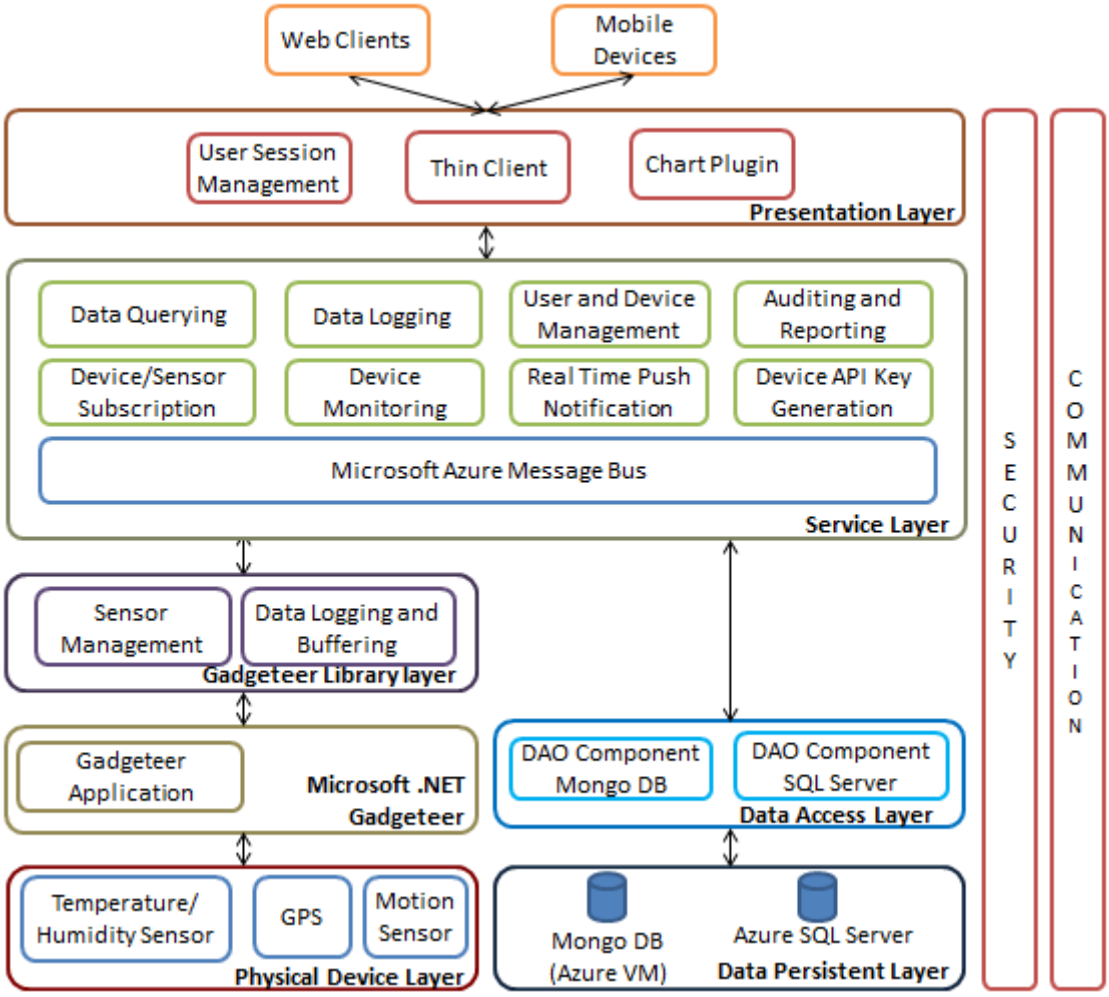

Figure 11 Layered Architecture View

The system is built on the basis of a multi-tier architecture which originates from the well-known three-tier architecture (Schulte 1997). Since the main goal of the project is to provide an abstract API to support different hardware platforms with RESTful API services, the need of multi-layering has emerged. Moreover, the multi layered architecture offers loose coupling between distinct layers and this significantly improves maintainability. At the top of the architecture is the presentation layer which is responsible for presenting data received from the service layer. The communication between the presentation and service layer is achieved through RESTful APIs and Microsoft Azure Message Bus. Furthermore, presentation layer includes the visualization components which are responsible to display data coming from the

service layer. The service layer includes a set of components to provide the following services:

- **Data Querying**: retrieves the data through the data access layer.
- **Data Logging:** transmits the sensor-derived data to data access layer.
- **User and Device Management:** sends and retrieves user and device information to and from the data access layer.
- **Auditing and reporting:** sends notifications to the users based on the defined triggers
- **Device/Sensor Subscription:** is responsible for associating sensors with devices
- **Device Monitoring:** tracks the status of the devices
- **Real time push notification: is** responsible for creating a virtual channel between the devices and the third party applications to feed live data
- **Device API key generation: is** in charge of identity management

Moving down the architecture stack is the Gadgeteer Library Layer. It defines a lightweight library that communicates directly with the service layer through HTTP protocol. Next is the data access layer which is responsible to access the data directly. This layer hides the complexity of accessing the data from the service layer and decouples it from the other layers; this improves the maintainability, reusability and scalability. At the bottom, we have the data persistence layer which is comprised of two distinct databases to deal with different type of data in the system.

4.1.2.3.IT System View

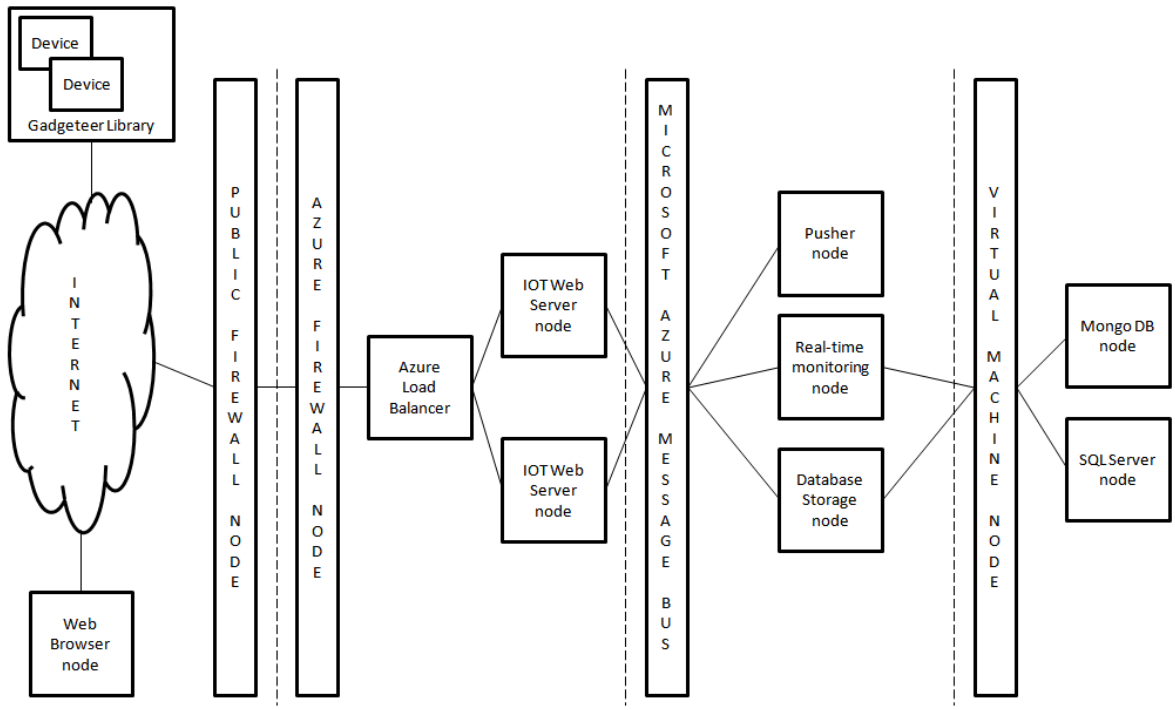

Figure 12 IT System View

IT system view shows the main nodes of the system in more details as it was mentioned by Mitra (2008). It was used as the starting point to create the component model as well as the entry point for our implementation (see Section 5). It contains the Azure Load Balancer which scales and balances the load between different IoT Web Server instances. In addition to that, the Pusher and the Real-time monitoring nodes (see Section 5) are worker roles that are running on the cloud. Finally, the Database instances are hosted on a Virtual Machine node which is also hosted on the cloud.

4.2. Technology Decisions

4.2.1. Azure Cloud Services

Gubbi *et al.* (2013) argue that internet services are the best solution for establishing Internet of Things and cloud centric approach is more viable than other alternative solutions. This approach utilizes the advancements in cloud computing to provide more reliable, scalable and cost efficient solution for our web service. Further, using a cloud service also avoids the need to maintain separate infrastructure to host our applications. We have chosen to use the Windows Azure cloud platform and obtained azure passes to register for Microsoft's Azure Cloud Services. Among various cloud computing services, we have utilised the Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) offering from Windows Azure Platform. Specifically, we have used PaaS to host the IoT web service application and IaaS for hosting MongoDB on a virtual machine. The web service application has been hosted using azure cloud service since it provides an easy way to package our applications and run it in the cloud. The IoT service APIs have been developed using WCF framework and created as Web Roles to be hosted as a cloud service. Apart from that, this project comprises of three more worker roles which perform background operations such as pushing real-time data, storing data to the repository and performing live monitoring of incoming data to flag malfunctioning devices respectively.

4.2.2. Windows Communication Foundation (WCF)

WCF is a framework for building service-oriented application and boasts the advantage of loose coupling, since it can communicate with clients created on any platform. The only constraint for this communication is that the essential contracts have to be satisfied. Yang *et al.* (2012) have shown how WCF can be used as IoT communication middleware for building dynamic, loose coupled, robust, scalable and flexible communication layer for supporting heterogeneous devices. As one of our main objectives has been to support a wide range of platforms, WCF was one of the obvious choices because of its loose coupling nature and the ease of hosting it as an azure cloud service. It is also capable of sending asynchronous messages between service endpoints and all the IoT service APIs are created as REST endpoints using 'WebHttpBinding'. WCF provides a simple and neat solution for exposing REST URIs. Furthermore, WCF is also considerably flexible and allows us to create our own security, transportation and authentication components which have proved to be extraordinarily useful while authenticating API Keys in the incoming messages to the web service. The WCF service application is hosted using Internet Information Services (IIS) which provides two unique features such as automatic activation, i.e. hosting the service only

on demand and process recycling when a service is unhealthy. This makes our web service fairly reliable and robust. WCF can also be programmatically self-hosted which simplifies the process of testing the service application by using MS Unit test framework for retrieving code coverage results.

4.2.3. Databases

There are two types of data that our system needs to deal with, first of which is the data required for the user and device management. This data is relational, therefore an obvious choice for the system was to use **Azure SQL Database** to store the data since it is a relational cloud database service and provides high availability and scalability. In order to provide a higher abstraction level in Data Access Layer in terms of database tables and columns, the team have used **ADO.NET Entity framework** which is an object relational mapper. The main advantage of using this framework has been the reduction of the amount and complexity of data accessing code; in addition to making the code easier to understand for others as it was discussed by Castro, Melnik and Adya (2007). Apart from the previous advantages, **LINQ** is also supported by Entity Framework. Thus, to query the database, LINQ query syntax was used (This is known as LINQ to Entities). The data model for this system can be found in Appendix A.

The second type is data generated by all the 'things' connected to the system. In order to tackle the challenges mentioned in section 2.2.2 we considered using a NoSQL database which has significant scalability and performance advantages over a relational database as discussed by Liu, Wang and Jin (2012). After researching about the available NoSQL databases the team has decided to use **MongoDB** mostly because of the following features that it offers.

- **Document-oriented storage:** the data is stored as BSON (Binary JSON) documents with flexible schemas. Hence, different hardware devices can send the data in their own schemas.
- **Replication:** as described in MongoDB Documentation Project (2013), the data can be duplicated on a different number of replica sets. One server becomes the primary server responsible for writing and the other servers are responsible for reading. This feature provides redundancy and raises data availability. In addition, it protects the data from a single point of failure and can be used in maintenance and backup procedures. More importantly, since in IoT the number of writing to and reading from the database is extremely high, it is possible to assign read and write operations to different servers and balance the load effectively.
- **Autosharding:** as described by Chodorow and Dirolf (2010, pp. 143-145), any database can be used for manual sharding. However, MongoDB handles it automatically which means it is abstract from the application layer. Sharding is about partitioning the data and storing each partition on a separate machine. The data growth in IoT is very fast, therefore, in order to make the data storage scalable, autosharding can be used to separate the data onto different machines and handle the load without having powerful hardware.

In this system, the MongoDB database is hosted on Azure Virtual Machine.

4.2.4. Hardware Platforms

4.2.4.1.Microsoft .NET Gadgeteer

Microsoft .NET Gadgeteer is a newly-produced hardware platform for rapid prototyping on Microsoft .NET Micro Framework within the concept of Internet of Things, upon which applications fulfilling the business value of specific scenarios are able to be implemented. Gadgeteer contains three critical elements: modular electronic devices which can be effortlessly connected to make sophisticated systems; object-oriented software library which conceals the complexity of low-level programming; and 3D design and construction tools supporting for quick physical form factor (Villar et al, 2012).

Gadgeteer has been chosen as the main prototyping platform partly because of the requirements specified in section 3.2 and also for the following reasons. (a) Gadgeteer is an emerging platform and this project can contribute to its development; (b) Compared to other platforms, Gadgeteer provides flexibility in using different modules and faster prototyping.

4.2.4.2.Engduino

The Engduino is an educational device designed in the Department of Computer Science at UCL, which is based on Arduino LilyPad in terms of hardware. The Engduino is programmed using the standard Arduino IDE through a normal USB connection.

The decision to use Engduino has been a result of the following reasons: (a) Engduino is built in the Department of Computer Science at UCL, therefore we can easily collect the devices and consult related staff; (b) The intention of Engduino is for teaching and IoT can be significantly used in this field; (c) The Engduino has a thermistor which satisfies the requirement of the third use case; (d) The Engduino can be used to prove the statement that our web service is generic enough to support different hardware platforms.

4.2.4.3.Android

Smart phones will undoubtedly play vital roles in the future of IoT. Among the popular mobile operating systems, we have handpicked Android, based on the following arguments: (a) According to research firm IDC's statistics between Q2 2012 and Q2 2013, Android accounts for 73.5% market share in the global Smartphone shipments; (b) Android is an open source platform which is compatible with the spirit of our project; (c) Android is another proof of the statement that our web service can support different hardware platforms.

4.3. UI Style Guide

The website has been developed upon a fully responsive and sophisticated modern UI bootstrap to deliver a great user experience. Harnessing the fundamental key properties of the modern UI style, the website meets all the standards to ensure secure navigation, stability, robustness and scalability. The fact revealed by Tami Reller, the CFO of the Windows division, that an estimation of 59M Windows 8 devices are in use as Gregg Keizer (9th May

2013) stated in his article reinforced our decision of building upon modern UI guidelines. As Microsoft's vision is to offer computer and mobile users the same interfaces, we have adopted the same strategy for our website. Therefore, users are able to visit the website either on their desktop or on mobile/tablet devices. In combination with that other up-to-date technologies such as jQuery and Ajax have been used to add responsiveness to the website. Moreover, user satisfaction and user experience have been our highest priority for UI design.

4.4. Integration with MSR1 Project

MSR LoT Analytic Engine team (Khandelwal et al. 2013) concentrate mainly on the data analytics part and uses Microsoft's HomeOS to gather data from Z-wave devices. HomeOS uses PC abstraction for the home technology and provides built in drivers for a very less number of hardware devices; therefore the other team used data from Z-wave device to perform the analytics. Even the newly released Lab of Things supports only Z-wave and Gadgeteer devices natively. To perform efficient data analytics it is necessary to gather data from a wide range of devices and this restriction can be overcome by using our system since it is generic and supports a wide range of hardware devices. Figure 13 describes how their data analytics application can integrate with our system. Therefore, their application can use our system to retrieve the raw data directly through an adapter to convert this raw data to the required format. The adapter can be written by developers who want to perform an analytic operation on their data. Hence these two systems can integrate with each other to provide an end-to-end functionality from device management, data logging to data analytics.

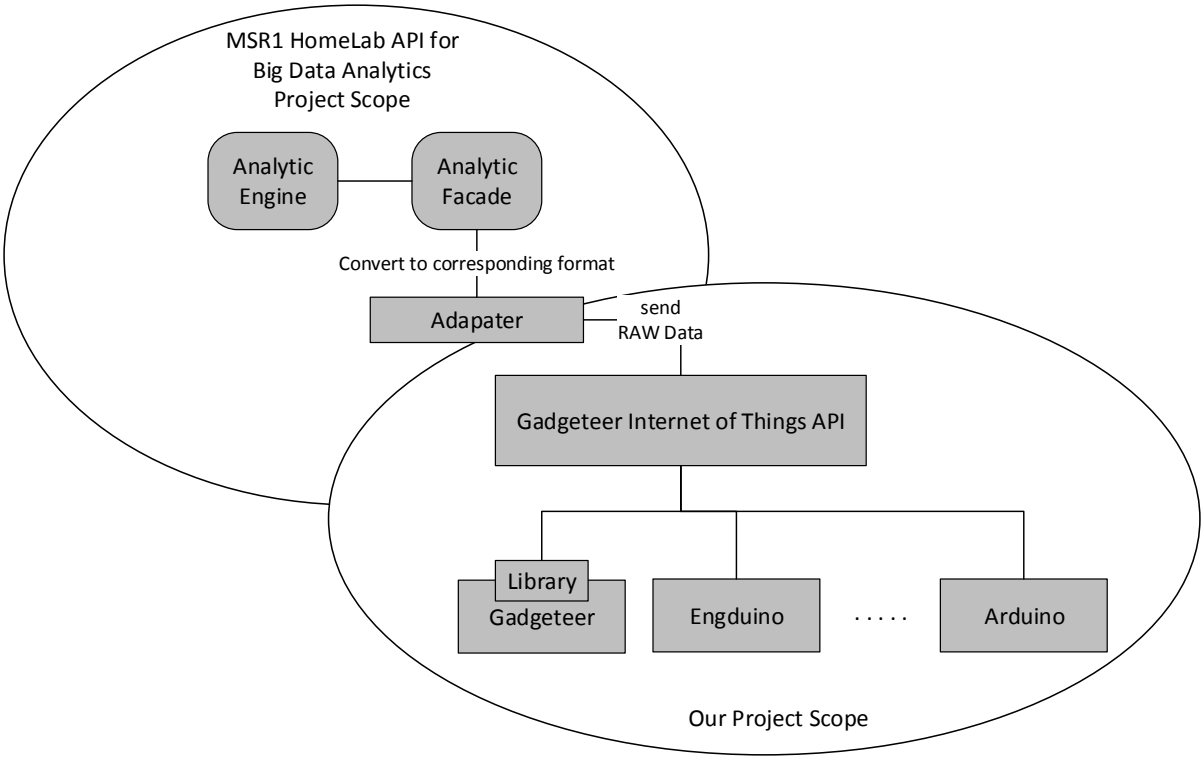

Figure 13 Integration with MSR LoT Analytic Engine

5. Implementation and Testing

5.1. Brief Description of Components

In this section, we will discuss about the key components of our system. Since this project is split into different decoupled layers, it is necessary to focus on all of the layers and the components it consists of. The three important layers of this project include the website, web service and the Gadgeteer library. A brief description of components included in these layers is described in the following sections.

5.1.1. Website

The Gadgeteer IoT website is developed to deal with user and device management. It also contains additional components such as visualisation and device discovery which can be seen as a third-party application that interacts with our API.

To start with, the user management component is responsible for managing the user's profile by providing common functionalities such as sign up, login and update/edit/delete user profile. In the same manner, device management deals with devices and sensors. It implements device administration functionality such as add new devices, update/edit/delete device's details and retrieving device's API Key (see Section 5.1.3) and download configuration file.

The visualisation component defines two types of data presentation, charts and maps. The former uses an off-the-shelf charting tool, the highcharts, which has been customised and extended to fit our context. The latter integrates Google Maps API v3 and provides features such as the device discovery page where the user is able to identify devices on a map. The live data transmission between the front end and the API has been achieved through Pusher, a third-party plugin. The following code snippet shows how the visualisation component subscribes to receive live data feed.

```
$.ajax({
    type: 'GET',
    url: "http://uclwebservicetest.cloudapp.net/Services/users.svc/subscribe?sensorid=" + sensorID + "&userid=" + getUserID(),
    contentType: "application/json; charset=utf-8",
    global: false,
    async: false,
    success: function (data) {
        var pusherConfig = JSON.parse(data);
        var pusher = new Pusher(pusherConfig.Key);
        var channel = pusher.subscribe(pusherConfig.Channel); //channel
        channel.bind(pusherConfig.Event, function (livedata) { //event
        });
    }
});
```

Code Snippet 1 Subscribing to real-time data feed

5.1.2. Web Service

The web service comprises of a number of components to provide abstraction between different layers. The component based implementation increases maintainability of the code

and improves easy extensibility in the future. Some of the important components are discussed in this section.

5.1.2.1.Real Time Monitoring Agent

RTMA (Real Time Monitoring Agent) is a specially designed component which inspects all the incoming messages and also monitors the status of every device in the system. When a device fails to send data over a time period defined by the owner of the device, this component will try to ping the device using the IP address of the device. Thus, RTMA can report the status of the device as online or offline, and if the device is offline it further verify whether it is the network connectivity issue or sensor malfunction.

5.1.2.2.Web Service API Controller

WSAC component is developed using WCF and exposes RESTful URIs for accessing various resources created in the system. This component routes the incoming requests to corresponding service components based on the URI and operations invoked by users. This component is easily scalable and extendable to include more operations in the future. A detailed list of APIs with their explanations is described in the Appendix A. Figure 14 shows the interaction of this component with other components in the web service layer.

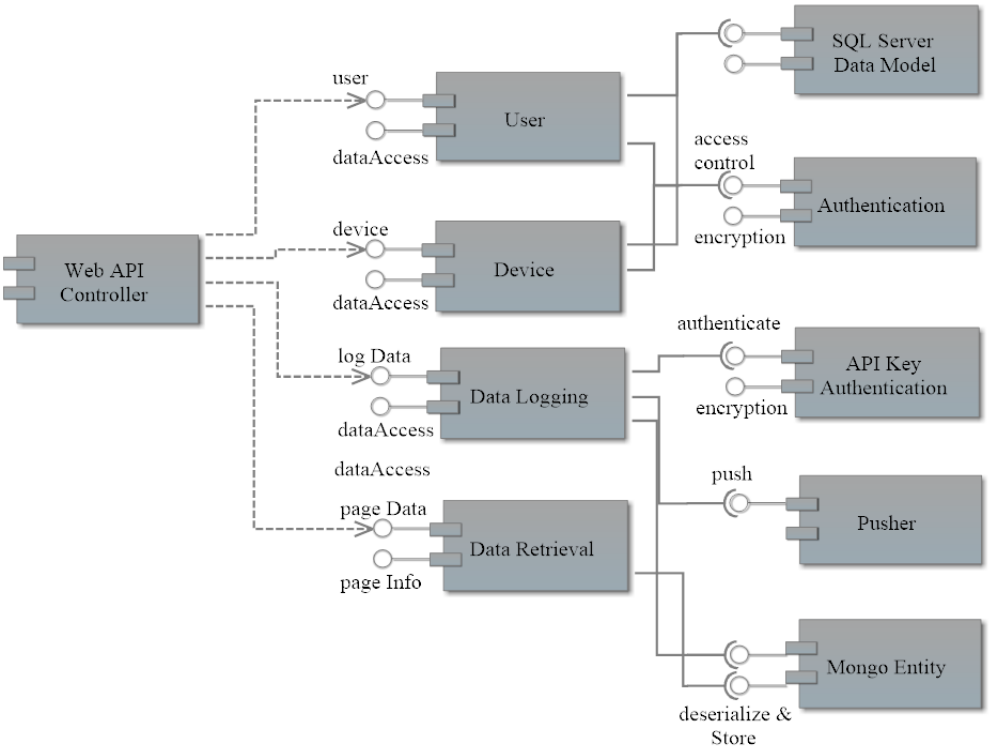

Figure 14 Web Service Components

5.1.2.3.API Key Authentication Component

This component extends the 'ServiceAuthorizationManager' class of the WCF framework and this class is triggered before calling the service method according to the sequence of

WCF lifecycle events. All the incoming requests will be checked and authenticated for a valid API key existence in the 'Authorization header' of HTTP messages.

5.1.2.4.Pusher Component

Pusher Component is an azure worker role whose sole responsibility is listening to the Windows Azure Service Bus and sending push notifications to corresponding channels based on filter rules. This component is implemented using publish/subscribe design pattern which observes to newly published data from the registered devices and send it across to subscribed users of these devices.

5.1.2.5.Database Storage Component

DB component in turn has two sub components which are responsible for SQL Server and Mongo respectively. The Mongo component is a worker role which processes incoming messages and store the data in the corresponding repository based on user preference. The default configuration stores data in public cloud but this configuration can be changed to support user's custom database by providing valid connection string for remote database server.

5.1.3. Gadgeteer Library

Gadgeteer library consists of the following three major components. The class level description of these components and explanation of library API can be found in section 2.1.3 of Appendix A.

5.1.3.1.REST Client Component

This component is responsible for handling the communications between devices and the RESTful web service. Given the intention of communication such as sensor registration, live data delivering and heart-beat notification, it creates HTTP request accordingly and also parses the response returned by the web service. Additionally, the REST Client Component is also in charge of distinguishing the sensor types and passing raw data to corresponding JSON serialiser, thereby constructing correct JSON data in the request payload.

5.1.3.2.Data Buffering Component

Considering the fact that existing network technologies need to be improved to support IoT, there is a high possibility that the devices encounter connectivity issues. Data Buffering Component is designed to store the data temporarily to the local storage such as SD Card, which guarantees that no data is lost when devices are disconnected to the network. This buffered data will be sent automatically when the network connection is re-established.

5.1.3.3.Configuration Manager

To register a sensor or send generated data, settings such as API Key, device ID and current UTC time are required. The major purpose of implementing Configuration Manager is to automatically read configuration file downloaded from the web site and synchronise the local

time with UTC time from an NTP (Network Time Protocol) server, which prevents developers from manually configuring these settings.

5.2. Use Case Scenarios

5.2.1. Common Walkthrough

Having briefly described the main components of the system in the above section, we would like to present a walkthrough of how these components are interacting with each other. To utilise the offered services, developers need to first login as valid users at the web site. Once logged in, at the device management page, developers can perform different actions such as addition, updating and deletion. Under the same user account, device names should be distinctive for easy identification. Each existing device is assigned a unique API key and a device ID, both of which will be used further in communication between sensors and the API. The device description is displayed in a tabular form on the web page. To proceed, the developers then have to register the sensors such as temperature and humidity, to the web service. In the sensor registration, developers need to specify the device ID and insert the API Key as an Authorization header in the HTTP request, which will be verified by the web service for authentication. The required information can either be retrieved automatically by the method ReadConfiguration() in the Gadgeteer library based on the associated XML file downloaded from the web site, or manually typed in the hardware application. By importing the library, developers simply invoke the methods such as RegisterSensor(), SendCurrentData() and SendHistoricData(), which hide the complexities of communicating with the API, to complete the registration and data logging operations. The following code snippet illustrates how a sensor can be registered easily by using the library.

```
// Configure the registration information
string apiKey = "e7cc056a-fac8-4f58-aa79-f51d9af9b37c";
Device device = new Device(1061, "GadgeteerDevice");
ArrayList sensors = new ArrayList();
sensors.Add(new Sensor("MyGPS20", SensorType.gps, "GHI", 5000, 20000, 60));
RESTClient restClient = new RESTClient(apiKey, device, sensors);

// Register the sensor list
restClient.RegisterSensor();
```

Code Snippet 2 Sensor Registration

Code snippet below demonstrates how to send real time data by invoking the method from the library.

```
// Construct the data to be sent
ArrayList gpsData = new ArrayList();
string currentTime = NtpClient.GetNetworkTime().ToString("dd/MM/yyyy HH:mm:ss");
gpsData.Add(new TimedData(currentTime, "23.1", "24.1"));

// Send the real time data
restClient.SendCurrentData(gpsData, currentTime, currentTime, "MyGPS20");
```

Code Snippet 3 Data Logging

Figure 15 briefly describes this work flow process through a walkthrough diagram.

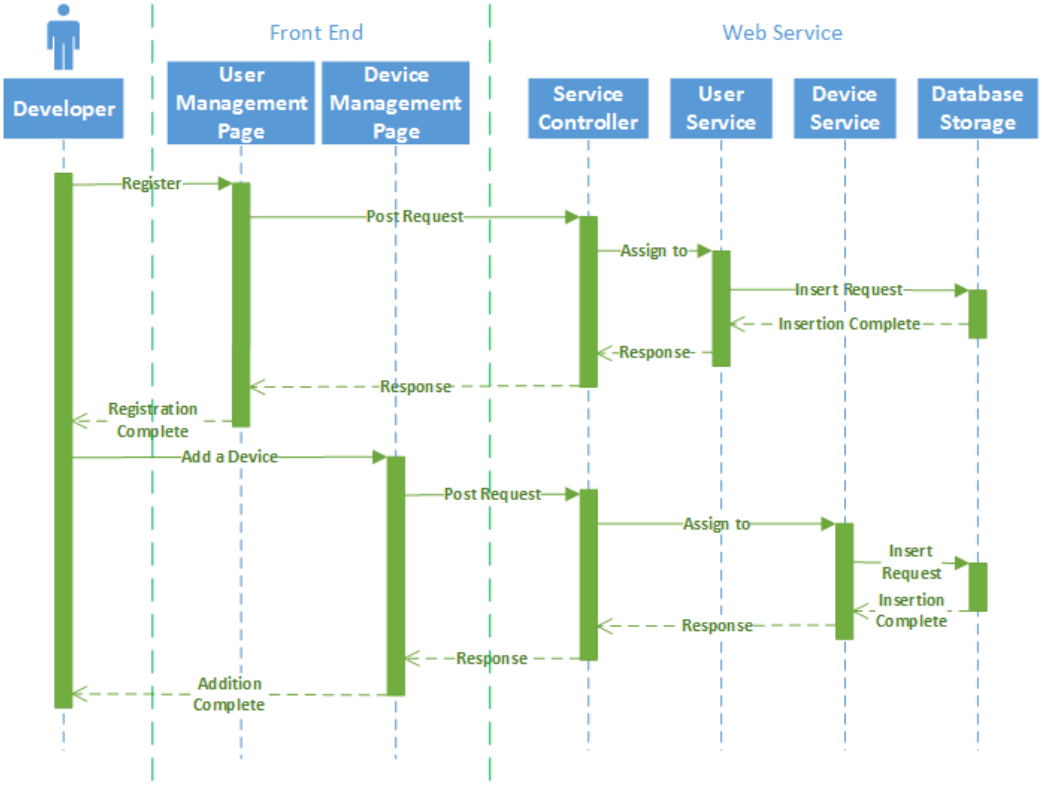

Figure 15 Walkthrough of the User Registration and Device Addition

5.2.2. Use Cases Implementation

We have designed and implemented three use case scenarios by making use of the API and library mentioned in the previous section. The main reasons behind these use cases are to show that our system meets the requirements of our product owners in addition to demonstrate how it can be used in different areas of the real word.

The first use case focuses on the applicability of our system in education and how it can be used by high school students. In this use case, science students log the temperature and humidity of their classes by using the Gadgeteer platform and through our provided API and Gadgeteer library. Afterwards, they use the website to share their devices with each other. Finally, by using the visualisation section of the website they see the data on a graph thus, they can compare the temperature and humidity of their classes with other classes or between different periods of time.

The second use case has concentrated on providing safety and security to people and the environment which was discussed under general requirements in section 3.2. The general idea of the use case is to create a network of devices that are physically close to each other, such as the devices that a person carries with him on a daily basis and make an alert when any device gets disconnected from that network. More importantly, provide the user the ability to see the real time location of the missing device on a map. Therefore, in this use case, we created a network using Bluetooth between 3 devices; the alerting device (Gadgeteer) which

is the Bluetooth host in the network; a mobile phone running on Android OS and another device (Gadgeteer) as the Bluetooth clients. When a client gets disconnected from the host, the host device starts to produce an alarm sound and sends the information of the lost device to the API. Then, through our website, the user logs in to his account and use the map to see the real time location of his missing device.

The third use case has focused on the device discovery feature of the API. This feature enables users to receive data from other devices with similar capabilities. In this use case we have considered a radioactive zone and a sensor that logs the data for that area. However, the sensor breaks down due to hardware failure. Since fixing the sensor by human intervention is not practical in this scenario, the system suggests the receiver of the data to switch to the nearest device that is sending the same type of data. Yet, since Gadgeteer does not have any sensor to measure the radioactivity level we have used temperature sensor to mock the scenario. In this use case the first sensor is on the Gadgeteer platform and the second one is an Engduino device.

5.3. Software Measurement

Because of our intention to develop software with high quality in terms of reusability and maintainability, along with implementation, we have been measuring the existing code to gain an insight into the current state of the project. Among various code metrics, according to the user guide (Microsoft Developer Network, 2012) focusing on the code metric tool embedded in VS 2012, we calculated the following ones:

Metrics	Definition	Indication
Maintainability Index	An index value on a scale of 0 to 100 which indicates the relative ease of maintaining the code	A higher value indicates better maintainability
Cyclomatic Complexity	The structural complexity of the code generated by computing the number of various code paths in the flow of the program	A higher value indicates less maintainability, which requires more test cases to achieve ideal coverage
Inheritance Depth	Maximum depth of a class in the inheritance tree	A higher value indicates greater difficulty to locate the definition or redefinition of particular methods or fields.
Class Coupling	The number of classes that a particular class is coupled to	A higher value indicates less reusability and maintainability due to the interdependencies on other classes
Lines of Code	The approximate number of lines in the Intermediate Language (IL) instead of the original code	A higher value indicates less maintainability, which requires the class or method to be split up

Table 3 Software Metrics

Hierarchy	Maintainability ...	Cyclomatic Com...	Depth of Inherita...	Class C...	Lines of C...
GadgeteerLibrary (D	76	273	1	39	619
WCFServiceWebRole	86	579	5	151	1,416

Figure 16 Software Metrics for Gadgeteer Library and Web service API

Based on the result shown in the figure above, we are able to assess the achievements of the goals and plans, build models based on relationships between observable attributes, identify potential risks and predict the complexity of subsequent testing. Moreover, by measuring these metrics, we have been able to refactor the code to achieve superior maintainability and reusability.

5.4. Testing Overview

5.4.1. Strategy

Every project should have a proper test strategy outlining about how and which artefacts will be tested along with test criteria and attributes. Since we followed Scrum project management methodology, all our testing efforts were aligned with values of agile manifesto. In Sprint 0, we laid out a master plan about how testing activities has to be carried out and the work flow to be followed. Figure 17 shows the testing process that we created and it was aligned with testing practices followed in Scrum. All the testing activities were carried out by following this process cycle.

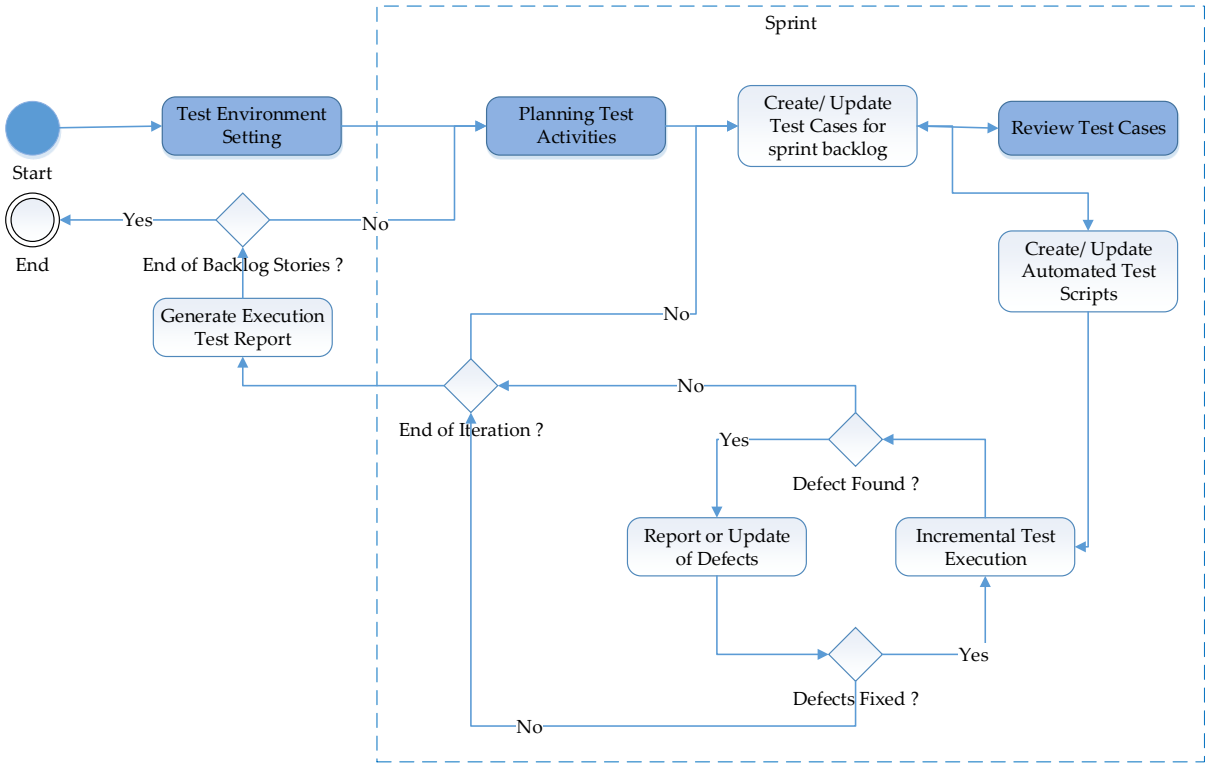

Figure 17 Testing Work Flow Process

Crispin and Gregory (2008) suggest that code coverage to be a good metric for testing in the Agile environment and furthermore Sommerville (2010) in his book describes that all-path

testing as one of the oldest structural testing method to achieve higher code coverage. Therefore high code coverage was selected as the test quality attribute and decided to achieve it through all path testing wherever possible. The software metrics described in section 5.3 were used to perform testing efficiently and Black and Mitchell (2011) describes cyclomatic complexity as effective metric for performing path testing. Moreover, according to Scrum practice, all the code should be unit-tested during development, so the test strategy included performing unit testing as and when an artefact was developed.

Since all the artefacts produced in a software system cannot be tested in a similar way, we have adopted different techniques for testing different layers of the architecture mentioned in section 4.1.2.2. Hence it was decided to perform unit testing on the data access layer and the Gadgeteer library; Integration testing by integrating the web service layer and data access layer components. The reasons for this decision are discussed further in the section 5.5.1 and 5.5.2 respectively. Fewster and Graham (1999) explain test automation is very critical to testing in their book. Therefore it was decided to perform all the above testing procedures through automated test scripts by making use of automation test tools such as MS unit test framework (Microsoft Developer Network, 2005). Further, it was decided to perform system or functional testing from the UI to test all the use case scenarios and user stories. In addition to above automated testing, we also decided to perform manual testing to test the use case implementations since it was not possible to create automated test scripts.

5.4.2. Challenges

We faced many challenges in obtaining the code coverage results and writing automated test scripts while testing the web service application. Most of the testing tools invoke REST URIs like a black box function and it is impossible to acquire coverage results and automate the testing process. Finally, we found a way around by self-hosting the service application within IIS programmatically and invoking the service methods just like any other C# library methods. Test cases were written using MS Test and code coverage results were generated by the MS unit test framework.

5.5. White Box Testing

5.5.1. Unit Testing

All path testing with k=1 has been used as the main approach to perform unit testing for data access layer. As it was described by Schligloffm and Roggenbach (2007), this type of testing is a white box structural testing that aims to find all the possible executable paths within the code.

The test suites have been created based on the classes that include the methods to be tested in different test cases. Each test case was defined by using the control flow graph. The team has set the target for code coverage to 90% in order to produce thoroughly-tested software.

The table below shows the code coverage for different test suits within data access layer. For the complete list of the test cases please refer to Appendix B.

Name	Not Covered (blocks)	Not Covered(%Blocks)	Covered (blocks)	Covered(%Blocks)
User Entity Suite	51	8.69	536	**91.31**
Device Entity Suite	57	5.25	1028	**94.75**
Sensor Entity Suite	56	8.55	599	**91.45**
Mongo Suite	17	6.88	230	**93.12**
Shared Entity Suite	6	6.12	92	**93.88**
Sensor Access Time Entity Suite	9	6.52	129	**93.48**
Notification Entity Suite	6	3.87	149	**96.13**

Table 4 Code Coverage Results of Data Access Layer Components

Owing to the limitation of .NET Micro Framework, the inbuilt testing tool in Visual Studio 2012, Unit Test Framework, cannot be executed. This indicates that there is no native support for unit-test on the Gadgeteer platform. Having realised that, in order to provide a thoroughly-tested Gadgeteer library, we decided to write test cases manually and execute them, because subsequent research revealed that no fully compatible external tools have been implemented for this relatively new prototyping platform. Additionally, considering that testing for the library requires extensive involvement of Gadgeteer devices and hardware-based applications, to build the test harness has been considerably time-consuming. (Please refer to Appendix B for the complete list of test cases over the Gadgeteer library)

5.5.2. Integration Testing

Integration testing is usually done by combining individual modules; test cases are executed on the integrated module. Although integration testing is generally black box, we have performed a white box testing to obtain code coverage. The web service component exposes REST URIs for accessing user, device and sensor resources; in addition allows CRUD operation on these resources through HTTP methods. As described in the layered architecture, these URI operations are wrapper around the data access layer operations. Therefore, there was no need to perform unit-testing for the web service layer since the data access layer was highly unit tested. Hence, the team decided to perform only the integration testing on the web service operations to test both the web service and the data access layer together. Since high code coverage is selected as the quality attribute, the WSAC component (see section 5.1.1.1) was tested using all-path testing with k=1 criteria and the figure 14 shows the code coverage results for this component. We were able to get only 71.73% code

coverage due to restrictions of WCF Self-Host in covering some parts of the code, the remaining parts of code could be executed only by running the application inside IIS.

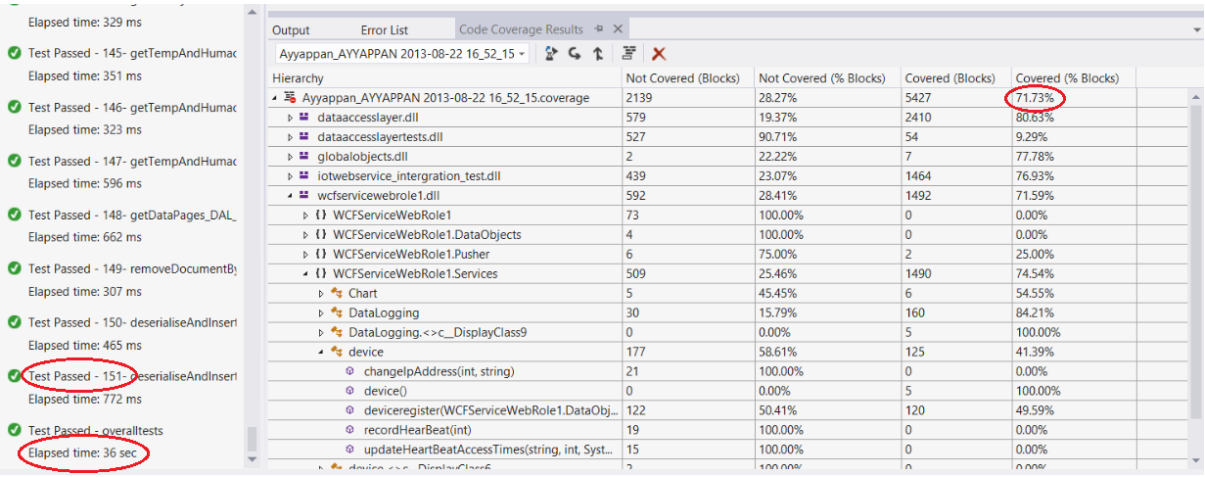

Figure 18 Coverage Results of Web Service Components

5.6. Black Box Testing

5.6.1. System and Functional Testing

It is obvious that our API could be invoked by any third-party application. Due to this fact, a functional testing which is classified as a black-box testing was the only option available to exercise our APIs in a third-party application. Our system, apart from the API also consists of a website which contains components that can be seen as implementation of the aforementioned third-party applications. Therefore, the website has been tested using automated tests through the user interface (UI) also known as Coded UI Tests by taking into consideration the acceptance criteria for all the user stories (see Section 6.2). Coded UI Tests are automated tests that drive the application through its user interface. They provide a mechanism to automatically execute and drive the application by simulating users' actions.

The UI Testing consists of four categories of test suites; users, devices, charts and others. Each category exercises different functionalities, for example, 'users' classification deals with every request to the API that is responsible for user management. The rest of the categories were also defined in the same manner. (Please refer to Appendix B for the complete list of the test cases).

Apart from the functional testing which exercised particular isolated functions of the system, we also considered system testing. In a nutshell, someone could call it as a "bigger Functional Testing" because functionalities are exercised in sequence or even better in a logical order as the user would perform them. Therefore, harnessing Visual Studio's Testing Framework which provides "ordered tests", system testing was achieved for the most common functionalities such as first login to the system, then move to my devices page, then add a new device and finally get the configuration file of a specific device.

5.6.2. Performance Testing

All projects have non-functional requirements and these requirements greatly influence the acceptance of the system. Performance is one of those requirements and this testing is usually done at the early stages and very often, in order to meet the performance requirements of clients by measuring the response time of critical functionalities and high frequency operations.

Carrying out performance testing is a mandate to our project since it encompasses a large number of users and devices. Any testing should have a test criterion, therefore we have chosen response time as ours since it is so far the most widely accepted and requested metric for performance testing (Meier et al. 2007).

In this project, one of the primary functional requirements is the ability to log data from a variety of devices and retrieve this data from the cloud, but this requirement has performance constraints for the system to be accepted. Data logging is a highly frequent operation as an enormous number of devices can send data at a single point of time to the API. Advanced REST Console tool has been used to measure the end to end response time (benchmarking) of data logging API.

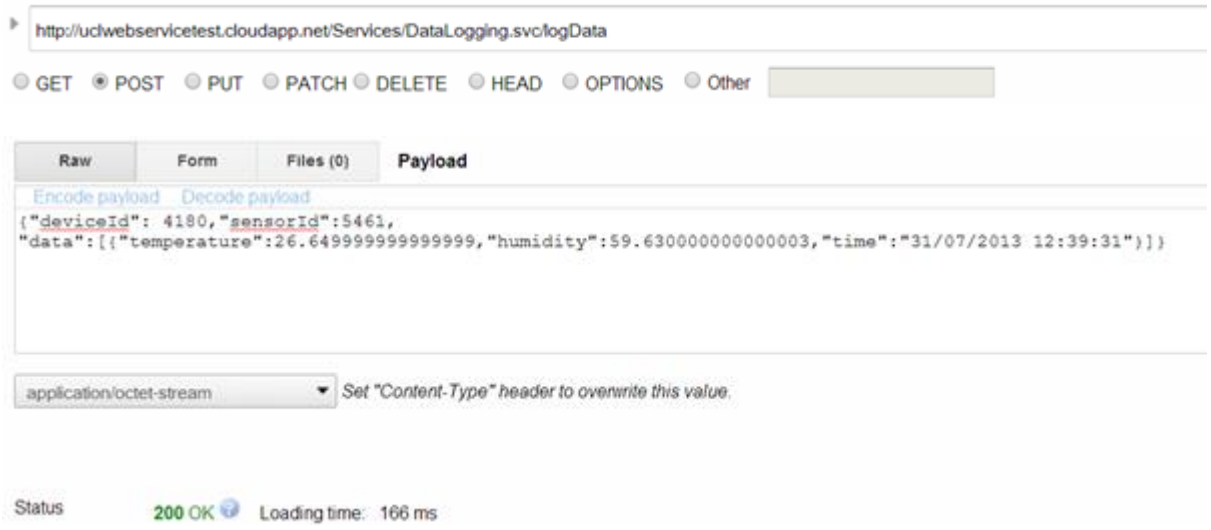

Figure 19 Response Time for Data Logging API

Figure 18 shows the response time of data logging API, but this response time is just one call or service invocation. Performance testing should usually consider different test cases with different data values and measure the average response time all the requests. But then Molyneaux (2009) states that average response time should be used in conjunction with *N*th percentile to get the best performance results. Percentiles are used to determine approximate response time at any given instance of time. Hence, we measured percentile for response time for both data logging and data retrieval APIs. The figure below shows the peak values at more than 90th percentile due to automatic on demand service activation of IIS discussed in section 4.2.2.

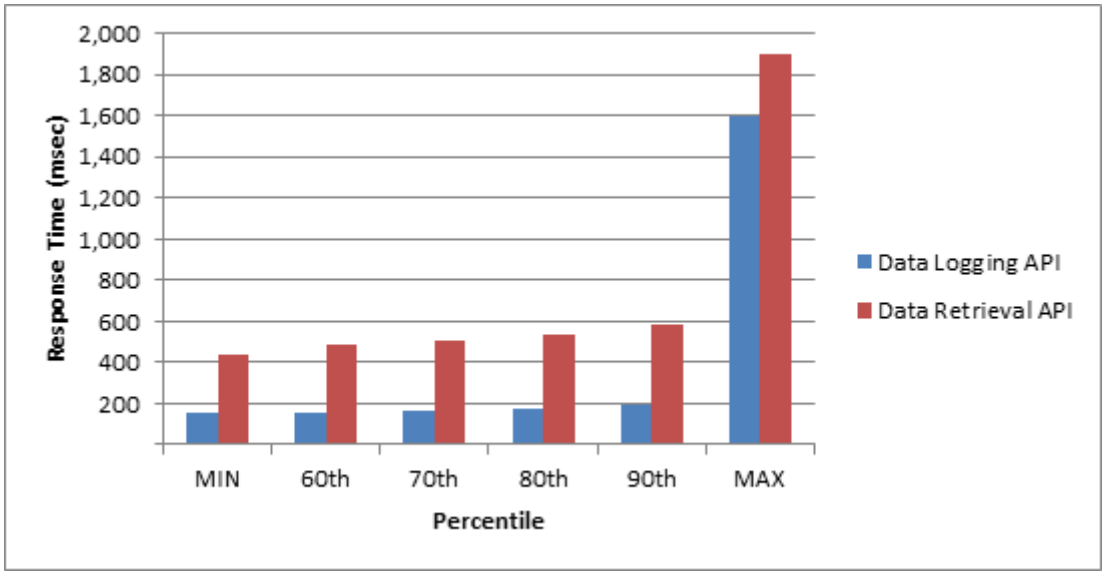

Figure 20 Nth Percentile 'Response Time' comparison between data logging and data retrieval API

5.6.3. Load and Stress Testing:

Load testing is all about testing the performance behaviour of a system under varying load conditions. Performance testing usually measures the response time at low load or varying load level for benchmarking purposes whereas load testing usually evaluates the performance at heavy load levels and checks if the system is still functioning properly. Load testing is usually performed by automated tools to increase the load to a system constantly till it achieves a peak load. We have used WCF Storm to simulate the virtual load on data logging API to see how many concurrent devices it can handle. Figure 17 shows how average response time increases and response rate decreases as the load increases on the system. Although the response time is 1278 msec. at 100 virtual users, it stabilised on that level for the given load and the system continued to function normally.

Stress testing usually tries to break the system and see how the system gracefully fails and recycles. It is usually performed by simulating an extreme load and removing resources away from the system and monitor how the system functions under these conditions. When we simulated extreme load levels, we received bad responses for the request and dropped further incoming messages. However, the web server continued to function and took some time to recycle in order to accept further requests. DB worker role (see section 5.1) was removed from the system to test tested if data logging works under resource failure condition. However, the API still worked owing to the queuing capability of the service bus and continued to accept a new request from the devices even under resource failure condition. When the worker role was started again, it continued to process the queued messages.

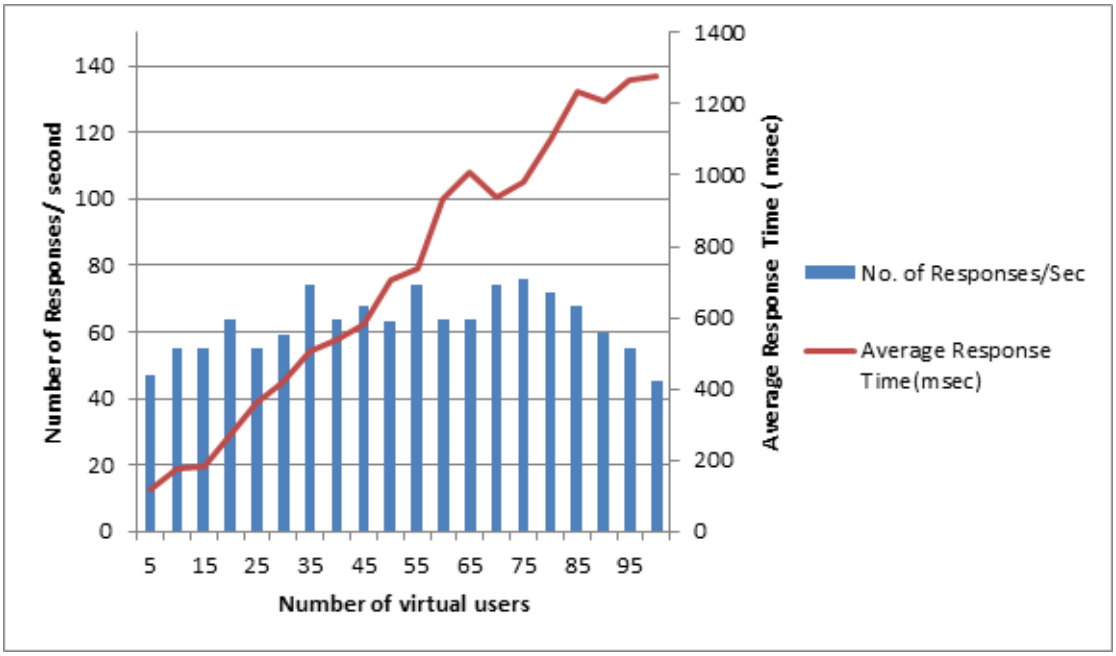

Figure 21 Load Testing Result for Data Logging API for varying load with simulated virtual users

5.7. Static Analysis

Static analysis is a critical part of software testing and complements the above tradition testing practices. This allows delivering a better quality code by identifying vulnerabilities in the code which may not be exposed by tradition test runs. Therefore it provides opportunities to improve the security and performance of the code, in addition to find the potential software quality issues that were not exposed by the above testing methods.

5.7.1. JavaScript analysis with JSLint

The website was built in JavaScript, CSS and HTML. Due to the weak typing nature of JavaScript, numerous errors and warnings are usually hidden. Therefore, the team used JSLint, a static analysis tool to assist us in revealing hidden errors in order to improve the overall code quality. Figure 22 shows an example of warning revealed by JSLint. An interesting warning related to equals operator is due to lack of strong data types present in high level programming. In this case, the variable is compared to 'undefine' data type, therefore, the correct usage should have included the directive typeof which the return type is a string literal. Thus, JSLint helped to improve code quality and readability.

	Description	File	Line	Column
47	JS Lint: Combine this with the previous 'var' statement.	charts_filters.js	123	14
48	JS Lint: 'jQuery' was used before it was defined.	charts_filters.js	123	20
49	JS Lint: Expected '===' and instead saw '=='.	charts_filters.js	125	30
50	JS Lint: Unexpected 'else' after 'return'.	charts_filters.js	126	14
51	JS Lint: Line too long.	charts_filters.js	135	106
52	JS Lint: Combine this with the previous 'var' statement.	charts_filters.js	144	14

Figure 22 Static Analysis Result with JSLint

5.7.2. Static Analysis with Code Analysis (VS 2012)

Code Analysis feature in Visual Studio 2012 provides static analysis functionality based on different inbuilt rules (Esposito, 2011). The code analysis rules engine, analyses the code statically without executing the project. We performed this code analysis on Data Access Layer in order to find the security vulnerabilities. This layer interacts with the databases directly therefore, finding security issues such as buffer overflow was critical. In addition, code analysis was performed on the web service components to analyse for any hidden security vulnerabilities that were not revealed in the unit and functional testing. Figure 21 shows an example of warning generated by this tool and it also suggested the corrective action to preserve stack details in order to avoid buffer overflow vulnerability. We used these suggestions from the tools to fix the vulnerabilities in the system.

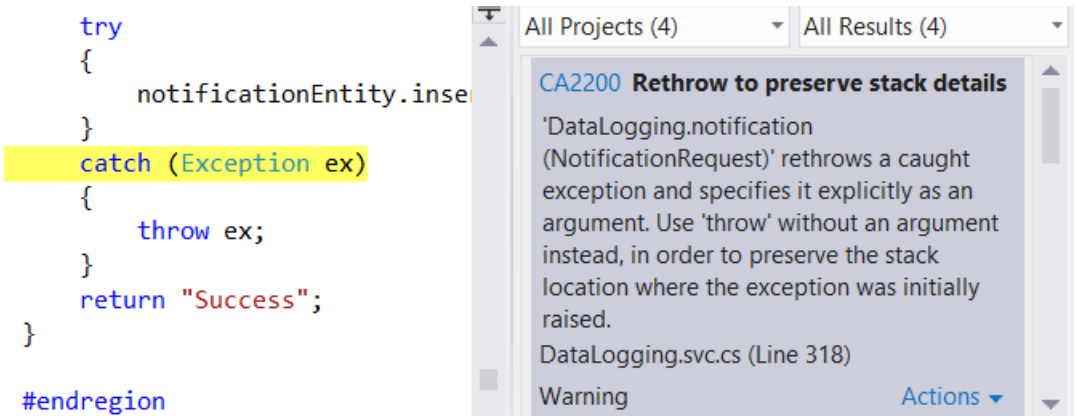

Figure 23 Code Analysis Result for Web Service

6. Project Management

This section briefly discusses the software development methodology that we have adopted to carry out this project and the reasons behind it. Likewise, how this methodology was useful to approach our project demands and distribute work among the team members is described.

6.1. Software Development Methodology

Typically, the software development can be represented as three models: the waterfall model, the incremental model and the reuse-oriented model (Sommerville, 2011). These generic models are abstractions of the process that can be practiced by different approaches to software development. Among all these methodologies, we have chosen Scrum (Takeuchi et al, 1986) for the following reasons:

(1) Part of the requirements was unclear at the beginning of the project. The second use case was sponsored by the Metropolitan Police, but the team was informed that we could not meet them in person until late July, which was the middle of the development.

(2) The requirements for the project were subjected to change. During the development, a similar platform from Microsoft Research named Lab of Things was announced. Due to its release, we modified the requirements after thorough research, because it was beneficial and critical for us to integrate with LoT.

(3) Prioritisation of the features we decided to implement initially kept changing while the development was in process. Based on client's feedback, we increased the priorities of some features, such as continuous data streaming and device discovery.

(4) Compared to traditional waterfall development methodology, Scrum tends to be more efficient. Features such as daily Scrum meetings, Sprint retrospective, intense collaboration with Product Owner and Burndown charts significantly enhanced the team's productivity.

(5) Scrum encourages developers to communicate frequently with others. According to the instructions of Scrum, we developed a communication plan and strictly followed it, which helped the team to maintain a shared understanding of the project throughout the development and also massively reduced the time wasting on waiting for other team members to complete their task.

6.2. Sprint Planning

The project development consists of four sprints, the first of which was spent in project initiation. The team focused on planning for the Scrum process, related system study, requirements elicitation, high-level design and setting up development environment (see Appendix A).

After the initial sprint, the team has started the actual implementation of user stories (see Section 3.2.2), which lasted for three sprints and Table 5 illustrates that.

Sprint	User Story	Acceptance Criteria
Sprint 1: June 20 - July 10	• Device and Sensor Registration • Data Logging • Real Time Data Transmission • Historic Data Retrieval	• Device and sensor registration cannot be completed without all the mandatory fields filled • Device with an existing name cannot be added • Information from the registration is stored in the database • Logging data with missing information cannot be completed • Data sent by devices is stored in the database
Sprint 2: July 11 - July 31	• Device Management • Data Privacy • Data Visualisation • Continuous Data Stream	• Information from the updated device is changed accordingly in the database • Information from the removed device is deleted in the database • Data from private devices cannot be retrieved by anyone but the owner • Time shown on the visualisation graph remains in accordance with the current UTC time • Historic data visualisation cannot be completed with ending time earlier than starting time • Historic data visualisation cannot be completed without data stored in the database for the selected timespan • After the device starts to log data, there cannot be missing records in the database
Sprint 3: August 1 - August 21	• Custom Repository • Data Sharing • Notifications • Device Discovery	• The specification of repositories with missing mandatory fields cannot be completed • Data sent by devices is stored in the personal repository • Data from private devices cannot be retrieved by unauthorised users • An Email is sent to the user after condition for the trigger is satisfied • Private devices cannot be discovered

Table 5 Sprint Planning

6.3. Task Distribution and Management

In Scrum, each team member, along with the client, has to be assigned a specific role. Table 6 shows the roles of different individuals involved in this project.

Name	Roles
Pejman Aghili	**Scrum Master**
Steven Johnston and Dean Mohamedally	**Product Owner**
Marios Constantinides, Ayyappan Sampath, Pejman Aghili and Zheng Gao	**Core Development Team**

Table 6 Scrum Roles

Throughout 12 weeks, the development team has strictly adhered to the discipline of Scrum. Team members came to the university at dedicated time and fully participated in the development together. Each sprint starts with a sprint planning and features to be implemented for this project have been registered as backlog items. The daily Scrum meetings have been conducted consecutively in a stand-up style, in which team members coordinated their work and specified what they had done yesterday, what they would be doing today and what problems were blocking them. Typically, for the team to proceed in development, the impeding issues would be addressed later by the Scrum master, Pejman Aghili. A sprint review meeting was held routinely after implementation, where the team reviewed the work that had been completed or not completed and presented the work to the Product Owner, Steven Johnston. Feedback was collected based on the prototype version which resulted in the potential change of feature prioritisation and creation of new backlog items. At the end of every sprint, the Scrum master would facilitate a retrospective meeting, which typically concentrated on inspecting how the last sprint went regarding to people, relationships and process; identifying the major items that went well and potential improvements; specifying the activities or habits that should be terminated; and creating a plan to enhance the team which would be enacted during the next sprint. Please refer to Appendix A for the list of retrospectives and minutes.

Team Foundation Service (TFS), as shown in the figure below, has been selected as the tool guiding the team to use Scrum for the following reasons: First, TFS provides native support to Microsoft projects, especially in cooperation with Visual Studio. Secondly, TFS offers an easy-to-use capability in the practice of Scrum such as sprint planning, backlog item management and remote feedback. Apart from team collaboration and Agile planning, features such as Git supporting are highly appropriate to the project. Furthermore, TFS has great accessibility and usability without infrastructure to manage.

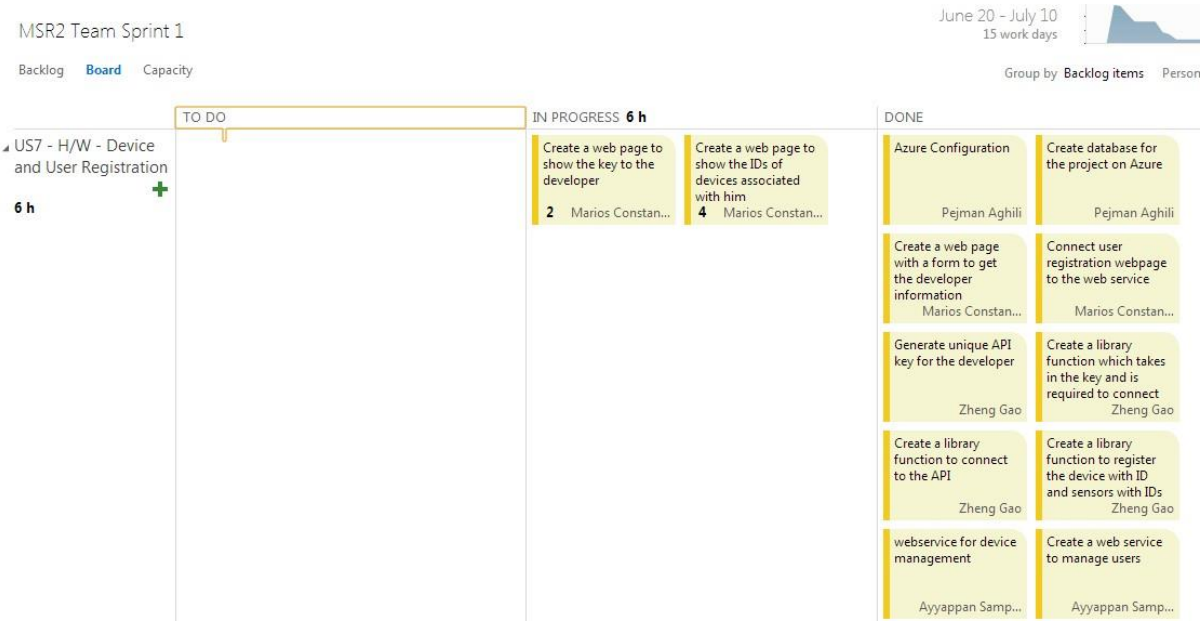

Figure 24 Backlogs in TFS

In terms of implementation, based on the architecture, the project has been split into four separate parts: the front-end, the web service, the data access layer and the Gadgeteer library, which have been assigned to Marios Constantinides, Ayyappan Sampath, Pejman Aghili and Zheng Gao respectively, according to personal interest and individual competence.

7. Project Results

7.1. Achievements

This project helps users and developers who are trying to experience the potential and opportunities within IoT by providing a platform to simplify the data management. As it was discussed in section 3.1.2, simplified data management was one of our top level goals for this project. The team has achieved this goal along with extended and innovative features to provide a better experience for the users. As it was mentioned in section 2.2.4 the privacy of the data generated in IoT is one the main concerns currently challenges IoT. Therefore, we have introduced innovative features such as the custom repository as well as extended features such as sharing and privacy status for every device. In addition, as it was described in section 5.2, we have also considered people who are not experts in programming such as high school students to explore the opportunities in IoT. Therefore, we have developed a responsive web site not only to deal with user and device management, but also to provide data visualisation and comparison based on different time ranges and sensors. Moreover, notification management has been included in this system to help the developers to set triggers on their data.

As it was discussed in section 5.2 we demonstrated how our API can be used in different use case scenarios and real life applications. Through these scenarios we have proved that the implemented API is generic and can be used by other platforms such as Android and Engduino which effectively shows our second main goal has also been achieved.

The device discovery feature is one of the highlights of this project since it does not exist in any other IoT platform before and has been designed and implemented by our team. This feature makes use of the fact that IoT connects more and more devices to the Internet every day; therefore, among all these devices, there are many sensors that log similar type of data such as temperature and humidity. Therefore, in situations where a sensor is broken or has lost connectivity to the Internet this feature can become handy by switching the data source from the broken device to the nearest sensor which is logging the same type of data. We have showed an example of the applicability of this feature in the last use case scenario in section 5.2. In addition, device discovery feature also enables the developers to explore all the available and accessible devices on a map and make use of their data by subscribing to them. It is worth mentioning that this feature has been implemented by considering the privacy of each device and sensor.

Along with the API we have also developed a lightweight Gadgeteer library as our third main level goal to ease the communication with the API. This includes sensor management and data logging library methods. In order to make the platform even easier to use for Gadgeteer developers, we have also provided the capability for the developers to download their device configurations as an XML file and use it in the library to prevent any hard-coding in their program.

7.2. Comparison with Existing Systems

Undoubtedly, Our Gadgeteer IoT API and libraries extend and improve aspects that existing systems lack to do by providing robust message bus architecture, discovery features and so on. Equally important is the fact that our API provides features that existing systems also offer without losing its uniqueness. Thus, this section will examine these features that distinguish our system from related and at the same time will point out functionalities that our system tackle in a better way.

To begin with, the option of choosing your own custom repository for data storage is one of the powerful features that differentiate our system from others. Compare it to Lab of Things which only allows storing data in your Azure Cloud Storage; our API gives you flexibility regarding your database choice which also might expose privacy issue if the data is stored in the cloud. Moreover, by storing data in your own private database you are able to perform any operation in that database at a later point and implement any business case using that data.

Furthermore, Gadgeteer IoT API provides high extensibility and can be easily integrated with other systems as shown in the example in Section 4. It allows the developers to subscribe to the Gadgeteer IoT Azure service bus directly and plug in more applications to their projects. The table below shows, some of the main features that our system provides in comparison with Lab of Things. Due to the fact that our project has been sponsored by Microsoft Research, this section is dedicated in the comparison between both systems.

	Gadgeteer IoT API	**Lab of Things**
Who can use it	Everyone	It is only available to Academia
Data Storage	Either on our database which is hosted on Azure or can be stored in a custom repository	Data is stored only to your Azure Cloud Storage account
Hardware Support	Gadgeteer as the main platform but it supports any hardware platform	They provide drivers for Z-Wave and Gadgeteer platforms but for other platforms you should write your own drivers
Big Data analysis	It does not provide natively, but it could be integrated with existing big data analytics tools easily	Not built-in functionality but you can integrate any third-party big data analytics tool.
Device Discovery/Switching	It provides discovery feature by identifying devices' failure and automatically switch to the nearest one	No built-in functionality
Notification Channels	It provides a generic API for setting/getting Notifications such as triggers when the temperature	Yes, it provides such as sends email after an event occurred.

	exceeds a threshold.	
Device Monitoring	Yes	Yes
Create Apps	You can directly use the device to send data	You should write drivers for HomeOS that use these devices.

Table 7 Comparison of Gadgeteer Internet of Things API with Lab of Things

7.3. Critical Assessment

Having discussed the achievements in the previous section, we critically assess the results of this project work in this section. Although data logging is made easy through APIs and libraries provided, it still enables only one way communication from the device to the web service. Although the requirement was just to address data logging problem, but in a wider sense it lacks two way communications which is important to send commands to devices such as power on/off, control thermostat etc. The main reason behind this drawback is the dynamic IP addressing mechanism in IPv4. As discussed in section 2.2.1, IPv6 will solve this problem; but currently in IPv4, the internal IP address and IP address couldn't be resolved while communicating to device using its IP address.

In addition, we have used HTTP protocol to create RESTful web service in order to support a wide range of devices. In general, high level protocols such as HTTP is frequently used for request/response type of communication, but they consume more power resources in embedded devices. Protocols such as MQTT consume less power than HTTP and offer publish/subscribe pattern using a central broker. Collina, Corazza and Vanelli-Coralli (2012) introduced QEST which is a combination of REST and MQTT for solving this issue.

Although this system provides privacy for the generated data by storing it in custom remote repositories of user's choice or allowing users to set privacy status for their sensors, but it lacks to provide security to this data. Data in transmission can be eavesdropped or altered and this system will not be aware of these attacks. Some of the new protocols such as light weight Constrained Application Protocol (CoAP) which use Datagram Transport Layer Security (DTLS) can ensure security to this data by providing extra encryption even in low cost embedded devices (Raza et al., 2013).

The Gadgeteer library provides some useful functionality like buffering data generated by the device in case of network connectivity failure. However, this additional feature comes at an expense of featuring a SD card in the device.

7.4. Further Work

Having in mind the time allocated to this project, undoubtedly there are improvements that can be considered in extending and enhancing its functionalities.

As an immediate work, this project will be used by the next academic year students under Dean Mohamendally's supervision. In general, the idea is to create a hierarchy of hardware

devices and manage them through the Gadgeteer API along with the implementation of some business use cases.

With regards to long term improvements we have considered ideas such as including Machine Learning models in our APIs to enable data prediction and creating libraries for other hardware platforms especially for Arduino which is popular and more mature. The former could be used to keep the models learning about the gathered data through the API. By harnessing the knowledge gained from the models the device discovery feature could be enhanced by predicting the future data. The latter idea can speed up the development process for other platforms as the Gadgeteer library does. By doing so, it adds extra value to the project and establishes Gadgeteer IoT API as a universal platform that can be integrated with any hardware device platform.

8. Conclusion

Gadgeteer Internet of Things API is the result of 3-month painstaking work by four bright software engineers, proved to be useful in different areas such as education and safer neighbourhoods. Microsoft Research can publish it in their website and integrate it with their existing systems.

Within the time and budget constraint, the team has managed to meet all the requirements proposed by the stakeholders, achieve great client satisfaction and deliver a completely-working system which will be extended by the students from UCL in the following years. Every team member has fully participated in the development with great passion and equally contributed to the outcome of this project.

Given this opportunity, by strictly following the discipline of software engineering, team members have been able to critically assess and actually apply what they have learnt through the master programme, from development techniques to project management methodologies, from requirements engineering and advanced design to implementation and testing. Considering that the majority of the team had not implemented API-based systems in .NET, we have enhanced the skills and gained rich experience from this project. More importantly, we have the chances to attend workshops and access to cutting-edge technologies, such as Internet of Things, Windows Azure and MongoDB, which has considerably broadened our vision and kept us in pace with the current technology trends.

Acronyms

Term	Definition
IoT	Internet of Things
LoT	Lab of Things
IFTTT	If this then that
UI	User Interface
NTP	Network Time Protocol
HTTP	Hypertext Transfer Protocol
SDK	Software Development Kit
GPS	Global Positioning System
ARM	Architecture Reference Model
SOA	Service-oriented architecture
URL	Uniform Resource Locator
WS*	Web Service Standards
WCF	Windows Communication Foundation
HTML5	Hypertext Mark-up Language 5
SQL	Structured Query Language
IIS	Internet Information Services
LINQ	Language-Integrated Query
IDE	Integrated development environment
CFO	Chief financial officer
DB	Database
IP	Internet Protocol
JSON	JavaScript Object Notation
UTC	Coordinated Universal Time

CRUD	Create, read, update and delete
CSS	Cascading Style Sheets
TFS	Team Foundation Service
NETMF	.NET Micro Framework
SD Card	Secure Digital Card
IPV6	Internet Protocol version 6
IPV4	Internet Protocol version 4
NoSQL	Not Only Structured Query Language

References

Abbate, J. E. (1994) From ARPANET to Internet: A history of ARPA-sponsored computer networks, 1966--1988, Ann Arbour: University of Pennsylvania.

Aghili. P., Constantinides. M., Ramamuniappa. S.A. and Gao. Z. (2013) MSc SSE - Gadgeteer Internet of Things API, Available at: https://gadgeteeriot.codeplex.com

Alex Wilhelm (16th July 2013) Microsoft releases Lab of Things, a new research platform to support real-world data in the cloud, Available at: http://thenextweb.com/microsoft/2013/07/16/microsoft-releases-lab-of-things-a-new-research-platform-to-support-experiment-data/?fromcat=all (Accessed: 19th August 2013)

Atzori, L., Iera, A., & Morabito, G. (2010). The internet of things: A survey. Computer Networks, 54(15), 2787-2805.

Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. Software Engineering, IEEE Transactions on, 22(10), 751-761.

Bauer, M., Bui, N., Carrez, F., Giacomin, P., Haller, S., Ho, E., Jardak, C., Loof, J. D., Magerkurth, C., Nettsträter, A., Serbanati, A., Thoma, M., Walewski, J.W. and Meissner, S. (2012) Introduction to the Architectural Reference Model for the Internet of Things, Barcelona, Spain: Internet of Things - Architecture (IoT-A).

Black, R. and Mitchell, J.L. (2011) Advanced Software Testing, 3 edn., : Rocky Nook.

Boehm, B. W. (Ed). (1989). Software Risk Management, DC: IEEE Computer Society Press.

Calabria, T. (2004) An introduction to personas and how to create them, Available at:http://www.steptwo.com.au/papers/kmc_personas/index.html (Accessed: 21 June 2013).

Castro, P., Melnik, S., Adya, A. (2007) 'ADO.NET entity framework: raising the level of abstraction in data programming', Proceeding SIGMOD '07 Proceedings of the 2007 ACM SIGMOD international conference on Management of data, (), pp. 1070-1072.

Chen, Y. K (2012) 'Challenges and opportunities of internet of things', Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific, (10.1109/ASPDAC.2012.6164978), pp. 383 - 388.

Chodorow, K., Dirolf, M. (2010) MongoDB: The Definitive Guide, : Reilly Media,January.

Cisco (2013) Embracing the Internet of Everything To Capture Your Share of $14.4 Trillion, Available at:http://www.cisco.com/web/about/ac79/docs/innov/IoE_Economy.pdf(Accessed: 19th August 2013).

Coetzee, L. and Eksteen, J. (2011) 'The Internet of Things - promise for the future? An introduction', , (978-1-905824-26-7), pp. 5.

Collina, M., Corazza, G.E., Vanelli-Coralli, A. (2012) 'Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST', IEEE 23rd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC), (), pp. 36 - 41.

Dingsøyr, T., Hanssen, G. K., Dybå, T., Anker, G. and Nygaard, J. O. Developing Software with Scrum in a Small Cross-Organizational Project. In Proceedings of the 13th European Conference on Software Process Improvement (Joensuu, Finland, Oct, 2006). Springer Verlag, 5-15.doi=10.1007/11908562_2

Esposito, D. (2011) 'Cutting Edge - Static Code Analysis and Code Contracts', MSDN Magazine, (), pp. [Online]. Available at: http://msdn.microsoft.com/en-us/magazine/hh335064.aspx (Accessed: 4th September 2013).

Evans, D. (2011) The Internet of Things. How the Next Evolution of the Internet Is Changing Everything, : Cisco Internet Business Solutions Group.

Faily, S., Flechais, I. (2011) Persona Cases: A Technique for Grounding Personas, Canada: .

Fielding R.T. (2000) Architectural Styles and the Design of Network-based Software Architectures, Irvine: University of California.

Garlan,D. and Shaw, M. (1994) An Introduction to Software Architecture, 1st edn., Pittsburgh: Carnegie Mellon University.

Google (2013) Google Maps JavaScript API v3, Available at:https://developers.google.com/maps/documentation/javascript/ (Accessed: 28 August 2013).

Gregg Keizer (9th May 2013) Estimates peg 59M Windows 8 devices in use, Available at:http://www.computerworld.com/s/article/9239047/Estimates_peg_59M_Windows_8_devices_in_use(Accessed: 21st September 2013).

Gu, G., Li, Q., Wen, X., Gao, Y., Zhang, X (2012) 'An Overview of Newly Open-Source Cloud Storage Platforms', International Conference on Granular Computing, (), pp. .

Gubbi, J., Buyya, R., Marusic, S. and Palaniswami, M. (2013) ' Internet of Things (IoT): A vision, architectural elements, and future directions', Future Generation Computer Systems, 29(7), pp. 1645-1660.

Guinard, D. (2011) A Web of Things Application Architecture – Integrating the Real-World into the Web, Zurich, Switzerland: ETH Zurich.

Haller, S., Karnouskos, S., & Schroth, C. (2009). The internet of things in an enterprise context. In Future Internet–FIS 2008 (pp. 14-28). Springer Berlin Heidelberg.

Hartman, B. (2009) An Introduction to Planning Poker, Available at:http://agile.dzone.com/articles/introduction-planning-poker (Accessed: 15 June 2013).

Highcharts (2013) , Available at: http://www.highcharts.com/ (Accessed: 28 August 2013).

Hodges, S., Taylor, S., Villar, N., Scott, J., & Helmes, J. (2013, February). Exploring physical prototyping techniques for functional devices using. NET gadgeteer. In Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction (pp. 271-274). ACM.

Hodges, S., Taylor, S., Villar, N., Scott, J., Bial, D., & Fischer, P. (2013). Prototyping Connected Devices for the Internet of Things

HP Solutions (2012) Managing the Internet of things, United States: HP Solutions.

IFTTT Blog () IFTTT Blog, Available at: http://blog.ifttt.com/page/2 (Accessed: 19th August 2013).

Intel Corporation (2011) Rise of the Embedded Internet, Available at: http://newsroom.intel.com/docs/DOC-2297 (Accessed: 18th August 2013).

Jones, C. (1994). Assessment and Control of Software Risks. NJ: Yourdon Press.

Kafura, D., & Reddy, G. R. (1987). The use of software complexity metrics in software maintenance. Software Engineering, IEEE Transactions on, (3), 335-343.

Khan, R., Khan, S.U., Zaheer, R. and Khan, S. (2012) 'Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges', , (), pp. 4.

Khandelwal, S., Misra, C., Kim, H.S. and Demirtsoglou, G. (2013) MSR LoT Analytic Engine, UK: UCL.

Kuladinithi, K., Bergmann, O., Pötsch, T., Becker, M., & Görg, C. (2011). Implementation of coap and its application in transport logistics. Proc. IP+ SN, Chicago, IL, USA.

Lawson, L. (2013) 'Big Data Gets Bigger as Internet of Things Awakens', , (), pp. [Online]. Available at: http://www.itbusinessedge.com/blogs/integration/big-data-gets-bigger-as-internet-of-things-awakens.html (Accessed: 23 August 2013).

Leffingwell, D. (2010) Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise, : Addison-Wesley Professional - Publisher.

Li, J., Moe, N. B., & Dybå, T. (2010, September). Transition from a plan-driven process to Scrum: a longitudinal case study on software quality. InProceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement (p. 13). ACM.

Liu, Y., Wang, Y., Jin, Y. (2012) 'Research on The Improvement of MongoDB AutoSharding in Cloud Environment ', , (), pp. .

Mathew G. (2007) 'Implementing Service Oriented Architecture', SETLabs Briefings, 5(2), pp. [Online]. Available at: http://www.infosys.com/infosys-labs/publications/Documents/SETLabs-briefings-implementing-SOA.pdf (Accessed: 2nd September 2013).

Maurits (2011) A simulation to show the importance of backlog prioritization, Available at: http://maurits.wordpress.com/2011/06/08/a-simulation-to-show-the-importance-of-backlog-prioritization/ (Accessed: 16 June 2013).

McConnell, S. (1996). Rapid Development: Taming Wild Software Schedules. Microsoft.

Meier, J., Homer, A., Hill, D., Taylor, J., Bansode, P., Wall, L., Boucher Jr, R. and Bogawat, A. (2013) 'Architectural Patterns and Styles', in Meier, J., Homer, A., Hill, D., Taylor, J., Bansode, P., Wall, L., Boucher Jr, R. and Bogawat, A. (ed.) Microsoft Application Architecture Guide. United States: Microsoft Corporation, pp. 19-35.

Meier, J.D., Farre, C., Bansode, P., Barber, S. and Rea, D. (2007) 'Performance Test Reporting Fundamentals', in Microsoft Patterns & Practices. United States: Microsoft Corporation, pp.

Microsoft Corporation (2011) Gadgeteer, Available at: http://www.netmf.com/gadgeteer/(Accessed: 3rd September 2013).

Microsoft Developer Network (2005) Unit Testing Framework, Available at:http://msdn.microsoft.com/en-us/library/ms243147(v=vs.80).aspx (Accessed: 4th September 2013).

Microsoft Developer Network (2012) Code Metrics Values, Available at: http://msdn.microsoft.com/en-us/library/bb385914.aspx (Accessed: 27th August 2013).

Molyneaux, I. (2009) The Art of Application Performance Testing, 1st edn., : O'Reilly Media.

MongoDB Documentation Project (2013) 'Replication and MongoDB', , (), pp. [Online]. Available at: http://docs.mongodb.org/master/MongoDB-replication-guide.pdf(Accessed: 26 August 2013).

Mulloy, B. (2012) API Facade Pattern. Apigee [Online]. Available at:http://pages.apigee.com/rs/apigee/images/api-facade-pattern-ebook-2012-06.pdf(Accessed: 26th August 2013).

Project Management Institute. (2004). A Guide to the Project Management Body of Knowledge (PMBOK Guide) [3rd Edition]. Author.

Raza, S., Shafagh, H., Hewage, K., Hummen, R., & Voigt, T. (2013). Lithe: Lightweight Secure CoAP for the Internet of Things. Sensors Journal, IEEE, 3711 - 3720 .

Richard MacManus (2011) Pachube Acquired: Why Did It Sell So Early?,Available at:http://readwrite.com/2011/07/20/pachube_acquired#awesm=~oeYdusIhZ9arYQ(Accessed: 19th August 2013).

Rising, L., & Janoff, N. S. (2000). The Scrum software development process for small teams. Software, IEEE, 17(4), 26-32.

Salvatori, P. (2013) Managing and Testing Topics, Queues and Relay Services with the Service Bus Explorer Tool, Available at: http://msdn.microsoft.com/en-us/library/windowsazure/hh532261.aspx (Accessed: 5th September 2013).

Schligloffm H., Roggenbach, M. (2007) 'Path Testing', , (), pp. [Online]. Available at:http://www.cs.swan.ac.uk/~csmarkus/CS339/dissertations/GregoryL.pdf (Accessed: 26 August 2013).

Siricharoen, W.V. (2012) 'User Persona Roles in the End-user Web Developing Approach', , (), pp. .

Sommerville, I. (2011). Software engineering. Pearson Higher Ed.

Spiess, P., Karnouskos, S., Guinard, D., Savio, D., Baecker, O., Souza, L. and Trifa, V. (2009) 'SOA-Based Integration of the Internet of Things in Enterprise Services', Web Services, 2009. ICWS 2009. IEEE International Conference, (), pp. 968-975.

Stoneburner, G., Goguen, Alice., Feringa, A. (2002) 'Risk Management Guide for Information Technology Systems', , (), pp. .

Sundmaeker, H., Guillemin, P., Friess, P., & Woelfflé, S. (2010). Vision and challenges for realising the Internet of Things. Cluster of European Research Projects on the Internet of Things, European Commision.

Takeuchi, H., & Nonaka, I. (1986). The new new product development game.Harvard business review, 64(1), 137-146.

Trifa, V., Guinard, D., Karnouskos, S., Spiess, P. and Savio, D. (2010) 'Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services', Services Computing, 3(3), pp. 223-235.

Vasters, C. (2012) 'Using Windows Azure Service Bus for ... Things!', The Microsoft Journal For Developers, 27(6), pp. [Online]. Available at: http://msdn.microsoft.com/en-us/magazine/jj133819.aspx (Accessed: 21st August 2013).

Villar, N., Scott, J., Hodges, S., Hammil, K., & Miller, C. (2012). . NET gadgeteer: a platform for custom devices. In Pervasive Computing (pp. 216-233). Springer Berlin Heidelberg.
Weber, R.H (2010) 'Internet of Things – New security and privacy challenges', Computer Law & Security Review, 26(1), pp. 23-30.

Windows Azure (2013) Service Bus Topics, Available at: http://msdn.microsoft.com/en-us/library/windowsazure/hh532029.aspx (Accessed: 21th August 2013).
Xively (2013) Public Cloud for the Internet of Things, Available at: https://xively.com/(Accessed: 19th August 2013).

Yang, Y., Wang, Z., Liu, Q. and Wang, L. (2012) 'Building a Pervasive SOA Based IOT Communication Middleware Using Runtime Compilation and Reflection', Services ComputingJournal of Computational Information Systems, 8(2), pp. 643-654.

Zhao, M., Wohlin, C., Ohlsson, N., & Xie, M. (1998). A comparison between software design and code metrics for the prediction of software fault content. Information and Software Technology, 40(14), 801-809.

Zhou, J., Hu, L., Wang, F., Lu, H., Zhao, K. (2013) 'An Efficient Multidimensional Fusion Algorithm for IoT Data Based on Partitioning', , 18(1007-0214), pp. 369-378.

Bibliography

Castellani, A. P., Gheda, M., Bui, N., Rossi, M., & Zorzi, M. (2011, June). Web Services for the Internet of Things through CoAP and EXI. In *Communications Workshops (ICC), 2011 IEEE International Conference on* (pp. 1-6). IEEE.

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*, : Addison-Wesley.

Lamsweerde, A van (Axel) (2009) *Requirements Engineering : From Goals to UML Models to Software Specifications*, England: John Wiley & Sons Ltd.

Michahelles, F. (2010) *The Internet of Things How it has started and what to expect* , Switzerland:

Mike Cohen (2010) *Succeeding with Agile: Software Development Using Scrum*, : Addison-Welsey.

Neil A. Gershenfeld (1999) When Things Start to Think, : Henry Holt & Company.

Schwaber, K. (2004). *Agile project management with Scrum*. O'Reilly Media, Inc..

Shelby, Z., Hartke, K., & Bormann, C. (2013). Constrained application protocol (coap).

Sundmaeker, H., Guillemin, P., Friess, P., Woelfflé, S. (2010) *Vision and Challenges for Realising the Internet of Things*, : European Commission Information Society and Media.

Watson, K., Nagel, C., Pedersen, J. H., Reid, J. D., & Skinner, M. (2010). Beginning Visual C# 2010. Wiley. com.