Design of a New Messaging System for use in the Mobile Space with the IOT protocol CoAP

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Khaled Haggag

©Khaled Haggag, July/2016. All rights reserved.

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

This thesis details the design and development of a new messaging system for use in the mobile space. This messaging system consists of one back-end server, an API, a proxy server, client components, and a website. Developing such a system for the mobile space is a challenge, however, due to the unique characteristics found in the mobile space. Solutions to overcome these challenges are derived from RESTful web services and cloud computing technologies. Furthermore, the HTTP and CoAP protocols are explored for use in this messaging system. Experiments are then conducted to derive the most optimal protocol. For the purposes of testing this messaging system, two adaptive application management features are developed and provided through the system to the mobile development community. The first feature is GUI Menu Ordering, which allows developers to adapt their application menu's automatically depending on user usage. The second feature is Proficiency User Modeling, which allows developers to automatically configure their applications depending on user use behavior style. Results show the successful development of an event messaging system. In terms of the protocols CoAP surpasses HTTP in terms of performance and data consumption. However, the addition of a required CoAP proxy server affects performance and client/server data consumption. Thus CoAP becomes slower than HTTP, but consumes less data if only the client data is measured. Furthermore, results show that if encryption is used there is an affect. CoAP with encryption remains faster than HTTPS, even with a CoAP proxy server.

Acknowledgements

Contents

List of Tables

List of Figures

List of Abbreviations

AAM	Adaptive Application Management
AES	Advanced Encryption Standard
API	Application Programming Interface
CRUD	Create Read Update Delete
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
DTLS	Datagram Transport Layer Security
GB	Gigabyte
GPS	Global Positioning System
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
LBS	Location Based Service
M2M	Machine to Machine
MB	Megabyte
MVC	Model View Controller
MQTT	MQ Telemetry Transport
REST	Representational State Transfer
SQL	Structured Query Language
TB	Terabyte
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Chapter 1

Introduction

Most existing messaging systems are currently primarily designed for wired environments making them inoperable or at best inefficient for use in the mobile space. With the continued growth of mobile computing and the applications therein the need arises for messaging systems better suited for the mobile space. This thesis outlines the design and development of such a new messaging system specifically for use on mobile devices in the mobile space. However, this requires some work due to unique challenges found in the mobile space. Research is thus conducted to find solutions. Furthermore, a study and analysis is conducted on the Hyper Text Transfer Protocol (HTTP) and the Constrained Application Protocol (CoAP) for use within the system. This all accumulates in the development of a new messaging system.

Figure 1.1: The messaging system is compatible with Android, iOS, and Windows (the figure used is from Daily Tech [19]).

Messages in the messaging system detailed in this thesis follow an event model, thus from here on the messaging system is referenced as an event messaging system. A message or event object is the result from an action taking place on the client side. It can be the click of a button, the navigation to another screen, the measurement of new data, etc. Each message thus describes and represents an event. These events are further detailed in chapters three and four.

For the purposes of testing the event messaging system, adaptive application management (AAM) features are developed and provided through this system to mobile application developers around the world. AAM is the process of managing software in relation to continually changing conditions. These conditions can be related to frequency of use, geographic location, user behavioral style, and other environmental factors. The following are some examples of AMM;

- Software running in cars can adapt to night time versus day time driving depending on the time of day.

- The start menu in Windows provides an adaptive menu by displaying the most current frequently used programs top down [29].
- The front page of some browsers adapts to display the most frequently used websites as shown in Figure 1.2.

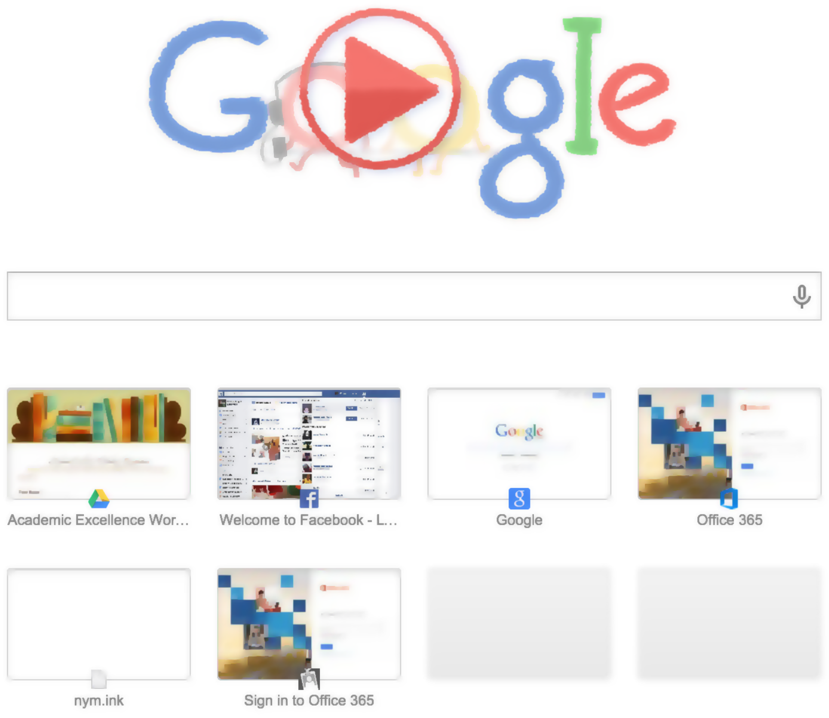


Figure 1.2: Chrome's new tab page showcases thumbnails for most frequently used websites. [18]

The following thesis is structured into six different chapters. Chapters one and two, provide an introduction to the challenges when building an event messaging system for the mobile space, and the communication protocols to be researched.

Chapter three, is a literature review providing a description of the research undertaken in finding solutions to the challenges previously described. This includes the following;

- An introduction and overview to mobile computing technologies.
- An analysis of communication paradigms in event based messaging systems.
- An introduction and overview of cloud computing technologies.
- A study of SOAP and RESTful web services.
- An introduction to Adaptive Application Management.
- An introduction to HTTP.

- A study of UDP vs TCP communication.
- An introduction and outline of CoAP.
- A survey of existing messaging systems.

Chapter four, details the design plan used in developing the event messaging system. This includes the technical implementations of the solutions derived from the literature review. Furthermore, the specific AAM features provided are detailed, and a couple prototype mobile applications developed for testing the new messaging system are also described.

Chapter five, describes a series of experimentation's performed on the event messaging system. Each experiment is designed to prove that the challenges described in the Problem Definition chapter have indeed been overcome.

Chapter six, provides a summary of the results from the experiments. The final research conclusions to the thesis are outlined as well in addition to future work.

Chapter 2
Problem Definition

2.1 Definition

The primary goal in this thesis is to design and develop an event messaging system for use primarily in the mobile space. Building an event messaging system for the mobile space is a challenge, however, due to the unique characteristics found therein. Furthermore, as with all systems, architectural challenges are also present. These challenges are described in the proceeding sections and overcoming them constitute sub goals. Beyond overcoming the challenges, the HTTP and CoAP protocols are also researched. They are compared against each other for use in the system.

2.2 Challenges In The Mobile Space

To develop an event messaging system for the mobile space specific challenges in the mobile space are overcome. These challenges are;

- Intermittent connectivity.
- Low storage and processor capabilities.
- The existence of many platforms.
- Low bandwidth availability.

2.2.1 Intermittent Connectivity

In the mobile space devices rely on wireless connections due to not being in fixed locations. Thus data connections in the mobile space are prone to intermittent connectivity. For instance, the connection of a smart phone can easily be lost if the user goes in a deep basement, outside the city, or in a metal warehouse. According to a June 2014 survey done in London, in fact one in three of all mobile internet tasks failed and one in seven voice calls attempted also failed [3]. The low power environment of the mobile space also compounds this issue with devices turning off as soon their batteries die. This is drastically different than traditional computing where the connection is more reliable, for instance with a home desktop computer.

2.2.2 Low Storage and Processor Capabilities

Storage space availability is also taken into account when designing software and systems in the mobile space. Smart phones, tablets, and other mobile devices have less storage than traditional fixed computers. For instance, the maximum storage space for a brand new iPad Air 2 (tablet) and iPhone 5s (smart phone) is just 128 gigabytes (GB), and 32 GB respectively. Meanwhile a brand new Mac Pro desktop computer has a maximum storage space of 1 terabyte (TB) [21]. This difference is due to fundamental design goals; it is important for devices in the mobile space to be as light and small as possible to increase portability, while for fixed computers it is not. For this reason, processor capabilities are also drastically different.

2.2.3 The Existence of Many Platforms

With traditional non-mobile computing, different platforms from Microsoft and Apple exist but they are rarely required to collaborate with one another. Thus, on a desktop computer most software only supports one platform. For instance, many computer games can only run on Windows. Regarding the mobile space and devices therein this is, however, required to be significantly different. Users in the mobile space own laptops and desktops, as well as other devices such as smart phones, tablets and so forth. These devices can have completely different platforms, even though they need to communicate with each other. For instance, a user may have an iOS laptop, a Windows home desktop, an iOS tablet, and an Android smart phone with the same calendar application on each.

For developers to release mobile applications that can run on all devices, they need to be cross platform compatible. Microsoft's One Note mobile application for example functions on computers, tablets, and phones belonging to; Windows, iOS, and Android.

2.2.4 Low Bandwidth Availability

The data connection of non-mobile computers have significantly higher bandwidth than mobile devices. Beyond device or environmental restrictions, mobile service providers place data caps depending on the price plan of a user. In 2014 global IP traffic for fixed Internet was 46,290 petabytes per month, while for mobile Internet it was 3,806 petabytes per month [4]. However, per device there is a greater difference. The average smart phone in 2014 generated 819 megabytes (MB) of traffic per month [7]. Developers of software and messaging systems consequently are restricted by this low bandwidth. To overcome these challenges a literature review is conducted to research existing solutions as well as to derive new ones.

2.3 Architecture Challenges

Beyond the challenges in the mobile space, there are also architectural challenges. A proposed event messaging system for the mobile space is shown in Figure 2.3. Following an event messaging model, data could be

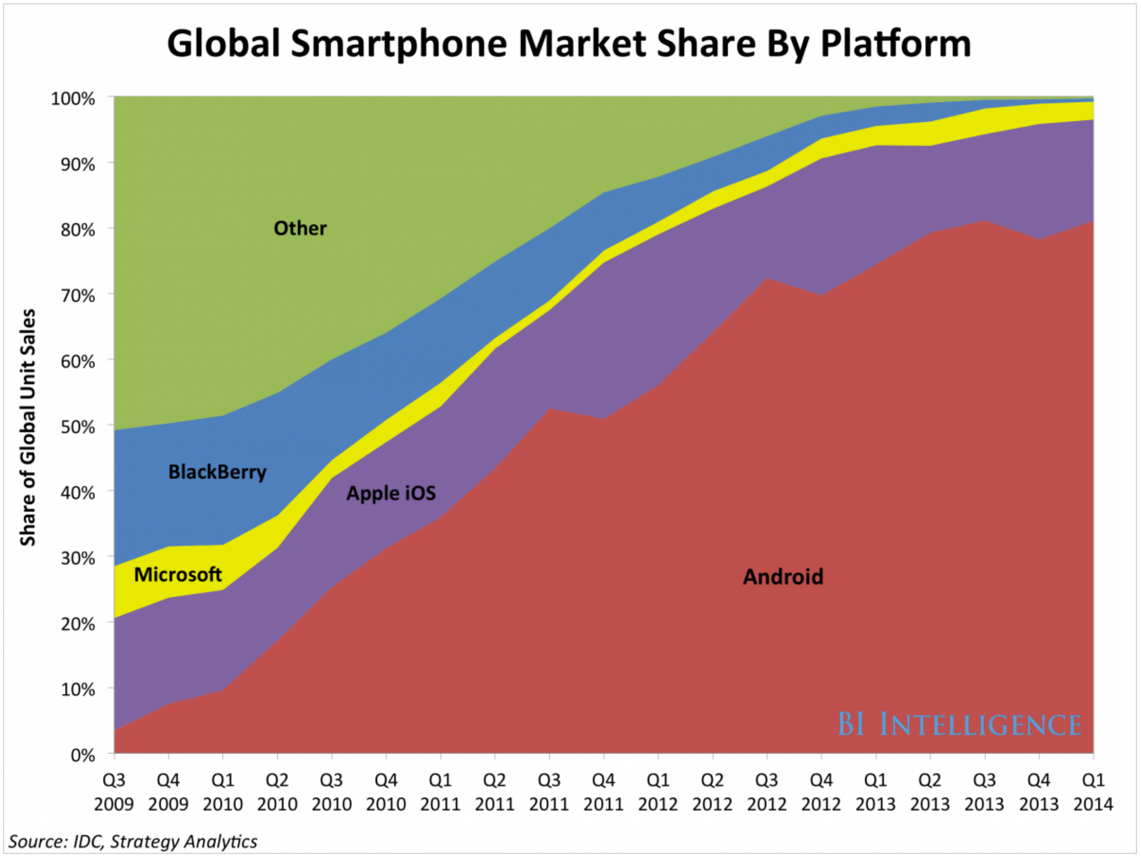


Figure 2.1: Android is the most used operating system, followed by iOS and Windows. [8]

organized into events and then sent to a cloud server. A local database on the mobile device could account for periods of no connectivity, and a cloud database could serve as the final storage location. An example use case would have events generated every time users perform an action on the graphical user interface (GUI). In doing so registering all user behaviour and allowing behavioural adaptation to occur as a result.

Furthermore by following a Model View Controller (MVC) architecture, each aspect of an application could interact with the system. For instance the event generation could occur in the view, event transition in the controller, and changes reflecting adaptation in the model.

The scenario in Figure 2.3, however, is just one untested possibility with many unknowns. For instance, how are the events organized from each other? How should the local database be configured? What type of communication protocol is optimal? What type of data format is most efficient? How is scalability handled? Answering these questions constitute further sub goals in this thesis.

2.4 Communication Protocols

After an event messaging system is developed, research is conducted on the system using HTTP and CoAP to compare the protocols against each other. Both CoAP and HTTP have different advantages and disad-

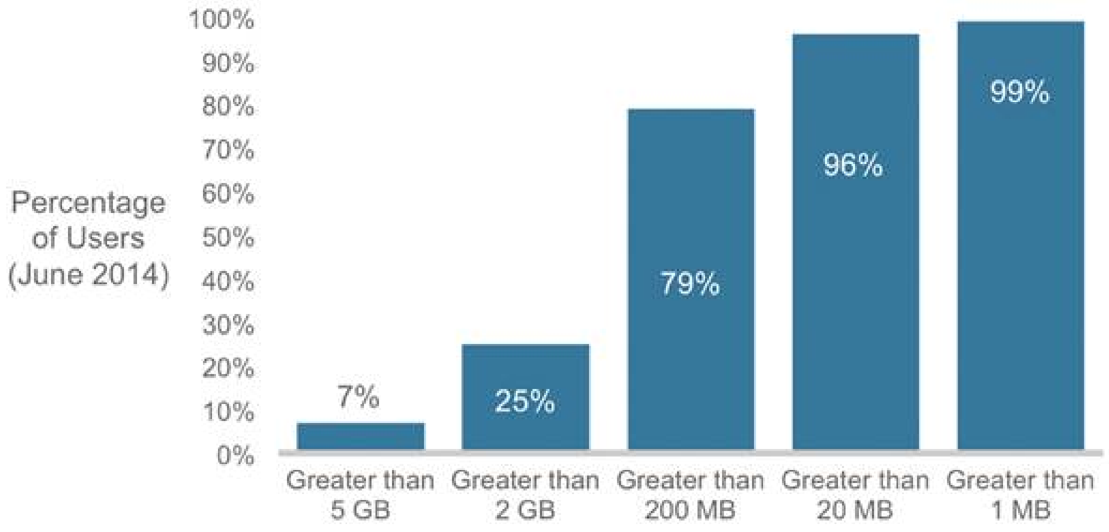


Figure 2.2: In 2014 only 7 percent of smart phone users consumed more than 5 GB per month. [7]

vantages, each offering potentially unique properties.

2.4.1 HTTP

The HTTP protocol first appeared in 1997, and became the most common protocol used as the foundation for all communication in the world wide web. It follows a request-response model in a client server environment as shown in Figure 2.4.

When viewing HTTP for the mobile space a few disadvantages arise. For instance, due to the overhead required, HTTP can relatively consume large amounts of data. How much exactly, and what is the difference with alternative protocols? Data consumption is critical in the mobile space due to the low memory and bandwidth environment. Another disadvantage is that HTTP does not provide secure communication. This issue is compounded in the mobile space where communication is predominantly wireless and eavesdroppers can easily intercept data. HTTPS provides a solution, however, how does this affect data consumption and performance? These and similar issues will be researched.

2.4.2 CoAP

CoAP is a protocol developed by the Internet Engineering Task Force Constrained Restful Environments Working Group. It is intended for low resource environments such as with IoT or machine to machine (M2M) devices, where power and processing capabilities are extremely restricted. Thus, CoAP focuses on efficiency. It also has been made compatible with HTTP by also following a request response model of communication. This allows devices using CoAP the ability to integrate with HTTP systems easily, meaning for instance that a CoAP client can potentially connect to a HTTP server. This in addition to a required proxy server allows CoAP to operate in the mobile space.

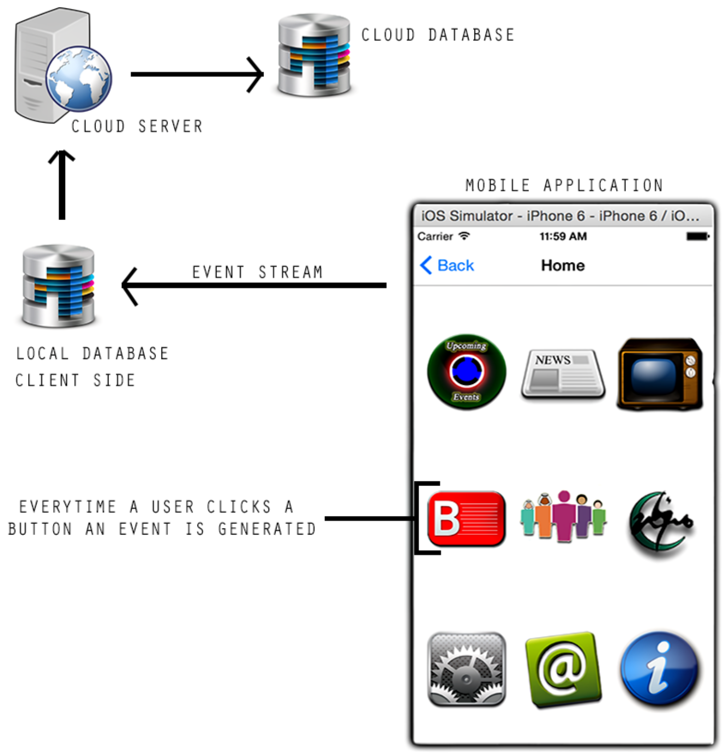

CLOUD DATABASE
CLOUD SERVER
MOBILE APPLICATION
iOS Simulator - iPhone 6 - iPhone 6 / iO...
Carrier
11:59 AM
Back
Home
Upcoming Events
NEWS
EVENT STREAM
LOCAL DATABASE CLIENT SIDE
EVERYTIME A USER CLICKS A BUTTON AN EVENT IS GENERATED

Figure 2.3: A sample messaging system design for the mobile space, with an example use case.

Introducing CoAP as a communication protocol in the mobile space is a new possibility. Is it possible? How does it compares against HTTP in terms of data consumption and performance? How does the addition of a proxy server affect the protocol characteristics? This will all be researched on the event messaging system.

2.5 Formalization of Problem and Research Goals

The primary aim of this thesis is to research and develop a new event messaging system for the mobile space. In order to do so mobile space and architectural challenges need to be overcome. Furthermore different protocols are explored to derive the most optimal choice for use in the system. Lastly, in order to demonstrate and test the event messaging system, AAM features are provided through the system to mobile developers around the world. The following is a breakdown of the research goals;

- Design and Develop a New Event Messaging System for the Mobile Space
  - Overcome Challenges Found in the Mobile Space
    - ∗ Intermittent Connectivity
    - ∗ Low Storage and Processor Capabilities

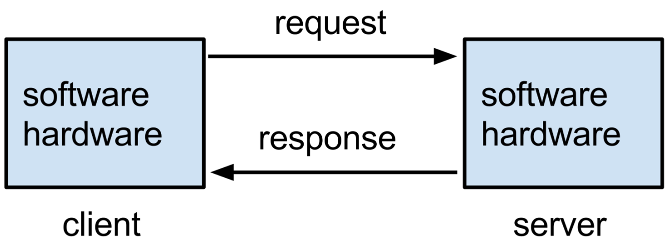


Figure 2.4: The request response model.

    * The Existence of Many Platforms
    * Low Bandwidth Availability
  - Overcome Architectural Challenges
    * Event Organization
    * Local Database Configuration
    * Contrasting Different Data Formats
    * Addressing Scalability
  - Compare and Contrast HTTP and CoAP for Use Within the System
- Evaluate and Test the System by Providing AAM Features

The literature review also includes a survey on existing similar messaging systems already in use. Google for instance, has an event messaging system servicing their Google Analytic's product. The purpose behind researching these similar systems is to derive additional solutions from them to some of the challenges described and further examine key similarities and differences.

Chapter 3

Literature Review

The following sections of the literature review detail the research undertaken to develop the event messaging system. This includes research in finding solutions to the challenges described in the Problem Definition chapter.

3.1 Mobile Computing

Traditionally computing has been within the domain of non-nomadic environments. For instance, the first computers were housed in climate-controlled rooms the size of a few bedrooms. Users accessed the computer through terminals spread through out the building. At this point not only was it assumed that the computer was stationary, but physically it was not possible for it to be mobile. Furthermore after significant advancements lead to desktop computers and laptops, it was still safe for software developers to assume that their code ran with the following conditions;

- A constant and stable internet connection, with bandwidth not being a major cost concern.
- A powerful power source with an expected little chance of interruption.
- An expectation that the user was not mobile (i.e. he or she is sitting at a desk).

It is to be noted that, with laptops it was a little different. However, this was not enough to influence software design. For instance, Microsoft Word is the same regardless of it running on a laptop or a desktop; it has the same features, GUI, resources required, and so forth. Laptop manufactures, however, did adjust their software a little and provided small suites of utilities. Such utilities are;

- Adjusting the display brightness.
- Adjusting the central processing unit (CPU) clock speed.
- Monitoring Wi-Fi with greater control.

These changes were made for the purpose of conserving battery life and/or increasing usability. However, these were not significant software design changes.

Significant changes only became required with the advent of the smart phone in the 1990's. Existing software could not run on these power and memory constrained devices, and an entirely new class of mobile applications began to be developed. These mobile applications were required to overcome memory and power limitations as well as other issues found in the mobile space. In 2014, 497 million mobile devices and connections were added [7]. Popular smart phones include the Galaxy S6 from Samsung and the iPhone 6 from Apple.

3.1.1 Smart Phones

Smart phones are characterized by three essential features [6]. These are;

- Having a specific development platform.
- Being multi-modal.
- Being spatially aware.

Development platforms are similar to operating systems of computers and allow smart phones to provide some of the same features a computer would provide. Android, Windows, and iOS are all different smart phone development platforms for instance. While, multi-modality serves the key purpose of keeping the device continuously connected by switching through a combination of different connections if one becomes unavailable. These connections can be; 4G, Wi-Fi, and Bluetooth for instance. Finally, spatial awareness is a concept where software have a constant awareness of the device location. This can be achieved through a variety of different sensors such as Global Positioning System (GPS), Wi-Fi, Bluetooth, compasses, and accelerometers. Spatial awareness easily enables Location Based Service (LBS) applications, which are the most popular. This can already be found on many existing mobile applications with features providing directions, such as the Google Maps mobile application. In 2014, 74 percent of adult smart phone owners ages 18 and older said they used their phone to get directions or other information based on their current location [5].

Figure 3.1: Lexus auto mobiles are able to automatically switch clock time zones as they travel across borders due to spacial awareness. [7]

Following the success of smart phones many other devices have also been introduced into the market such as tablets, e-readers, and so forth. According to Gartner, 195 million tablets were sold in 2013 [28]. These devices are also characterized by multi-modality and spatial awareness, and in many cases are similar to smart phones. For instance, an iPad (with a mobile network connection) is similar to an iPhone, except with a larger screen. The explosion of the mobile computing market has led many analysts to predict the end

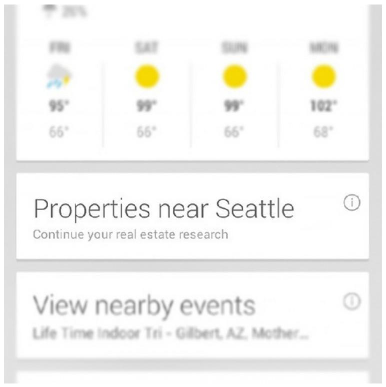

FRI
SAT
SUN
MON
95°
99°
99°
102°
65°
66°
66°
68°
Properties near Seattle
Continue your real estate research
View nearby events
Life Time Indoor Tri - Gilbert, AZ, Mother...

Figure 3.2: The Google Now application is able to provide many features due to spacial awareness. Some of these features include; a reminder of where you parked you car, nearby attractions, and timings for the closest bus stop. [17]

of desktop computers within a decade [28]. Parallel to the growth of mobile devices, the mobile application industry has boomed. According to Gartner, 102 billion applications were downloaded in 2013.

3.1.2 Mobile Space

The field of mobile computing is defined by computing that takes place on nomadic or mobile devices, and the issues that pertain to the implications of software running therein. This nomadic environment, also referred to as the mobile space, is characterized by unique challenges being;

- Intermittent connectivity.
- The existence of many platforms.
- Low bandwidth availability.
- Low processing and storage capabilities.

”Intermittent connectivity” as described in the problem definition section involves not having a stable or constant data connection. Thus, communication can be interrupted and messages can be lost during transmition. For the event messaging system this means events could be lost when clients loose connection unexpectedly. Furthermore the challenge is magnified if events are dependant upon each other. The loss of one could corrupt many other events. Intermittent connectivity requires a two-fold solution. One to prevent the physical loss of an event during transmition, and another to make sure events are not dependant upon each other if event loss should occur. The following two solutions are thus explored;

- RESTful web services.
- Off-line event buffer.

Figure 3.3: Devices in the mobile space.

RESTful web services is a communication architecture that provides a potential solution due to its stateless property. This is detailed in a proceeding section. Meanwhile an off-line event buffer would store events during periods of no connectivity for later transmission. For the event messaging system a client side local database could have all communication pass through it from the client to the server and act as a buffer. Furthermore by making use of multi-modality, the database before sending data to the server could check if a connection is present. If no connection at the time is present, then it would not send the data until the connection is re-established. Thus preventing the device from attempting to send data during a period of no connectivity and potentially loosing the data. To also address the "Existence of many platforms" challenge this client side local database could use a generic language and back-end library. Thus allowing the same database to be deployed on Android, Windows, and iOS. This database and an implementation configuration is further detailed in the System Design chapter.

"Low bandwidth availability", and "Low storage and processor capabilities" have also been described in the Problem Definition section. These two challenges are centred around data size, and thus, are mitigated by selectively choosing the right data format. This is also connected to overcoming the architectural challenge of "Contrasting different data formats", and finding the right one. The proceeding section describes research performed on comparing two common data formats already used in the mobile space.

3.1.3 JSON versus XML

Douglas Crockford was the first to specify and popularize the JavaScript Object Notation (JSON) format, and by 2001 it was available on JSON.org. It is supported inside Javascript and is a lightweight human readable data format, thus making it easy to parse and use. However, it does not have name space support, or input validation. JSON is popularly used for communication on the web due to the popularity of Javascript client side. This is only expected to increase as Javascript also begins to be used server side with the introduction of jNode.

Extensible Markup Language (XML) is a universal user defined data format that branched out from SGML (Standard Generalized Markup Language). It first became a W3C recommendation in 1998. The data format is both human and computer readable, with tags either being defined by users or machines. Tags in XML are also hierarchical, by having tags within tags to organize the data within. XML is another data format used for communication on the web.

XML and JSON are both human readable and appear similar in composition as shown in the proceeding examples from JSON.org. However, when processed by computers and in terms of memory usage the two formats differ. Many studies have compared XML and JSON in terms of performance and data size.

```
Example JSON:
{"menu": {
"id": "file",
"value": "File",
"popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

```
Example XML:
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

In the article ”JSON vs. XML: Some hard numbers about verbosity” [32], the author concludes in a case study performed that between XML and JSON, JSON is shown to achieve a reduction in data sizes. The study compared JSON and XML both in the case of Zipping the data and without zipping. One could argue that servers and clients today use gzipping, which factorizes redundant information like overheads and thus makes the size difference between JSON and XML not significant. However, the results of the study specifically showed that even with Gzip XML consumed 10 percent more data, and also consumed 20 percent more CPU time. Without Gzip, XML consumed 84 percent more data due to overhead.

Another case study on XML and JSON was conducted in a paper titled ”Comparison of JSON and XML Data Interchange Formats: A Case Study” [31], in which an evaluation was done on different applications using each format. Results therein showed that JSON is faster and uses fewer resources than XML. This is further strengthened by a third paper titled ”Mobile Application Webservice Performance Analysis: Restful Services with JSON and XML” [33], an analysis was conducted on restful web services using JSON and XML. Results again showed that JSON web service requests consumed less bandwidth and less processing power than XML web service requests.

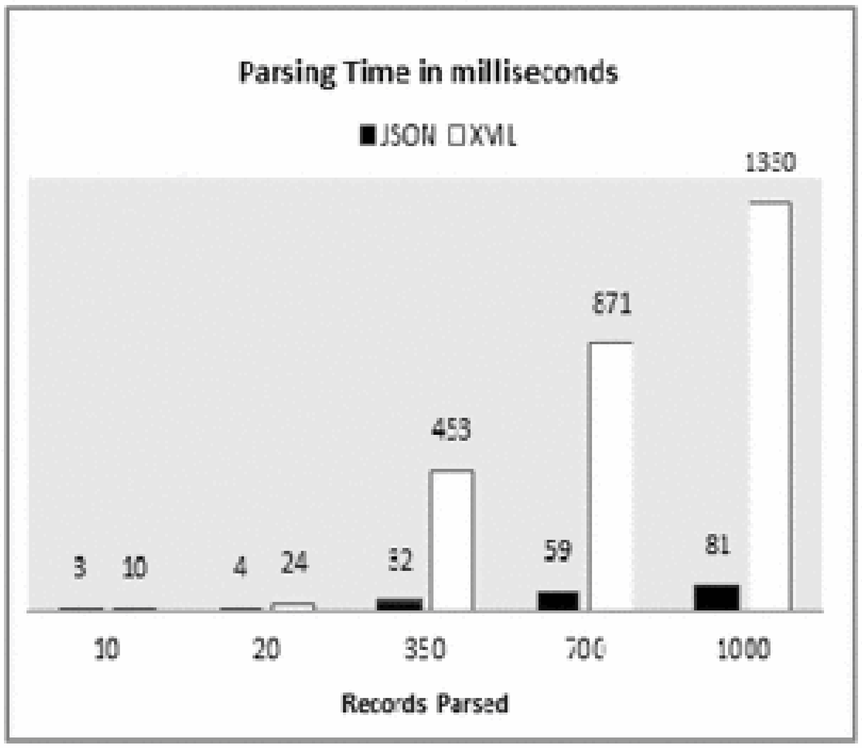


Figure 3.4: JSON is shown here to take considerably less time to process than XML requests. [33]

XML, however, in some use cases is superior due to limitations imposed on JSON such as not having namespace support. XML also provides extensibility, which allows it to easily be modified to different circumstances [26]. Furthermore XML can provide meta-data and thus describe the data itself. This makes XML superior for instance when using it to send full documents.

For the specific use cases of the messaging system described in this thesis, the research shows JSON to be superior than XML. JSON consumes less data, and uses less processing power. This tackles the "Low Bandwidth availability" and "Low storage and processor capabilities" challenges found in the mobile space. Additionally, an architectural challenge is also overcome of "Contrasting different data formats". The follow two sections detail the approach taken in addressing the remaining two architectural challenges being;

- Event Organization.
- Addressing Scalability.

3.2 Communication in Event Based Messaging Systems

Messaging systems come in a variety of different types and configurations. Event messaging systems are when the messages in the system are modeled as events. These messages are triggered by real events taking place on the client side. The click of a button, the navigation to another screen, the report of a crash are all examples of possible events.

To organize communication in messaging systems a publish/subscribe paradigm can be used. A type of publish/subscribe model is being channel based [20]. In a channel based publish/subscribe model, events are published into channels to categorize them. Each channel is unique. Clients then subscribe to the different channels, and publishers publish to the channels. The data flow of channels can be one-way or two-way. Each channel has two endpoints, and at each endpoint is a subscriber or a publisher. Furthermore, channels also have different access controls [2]. Some channels can only be subscribed to or published by a select few.

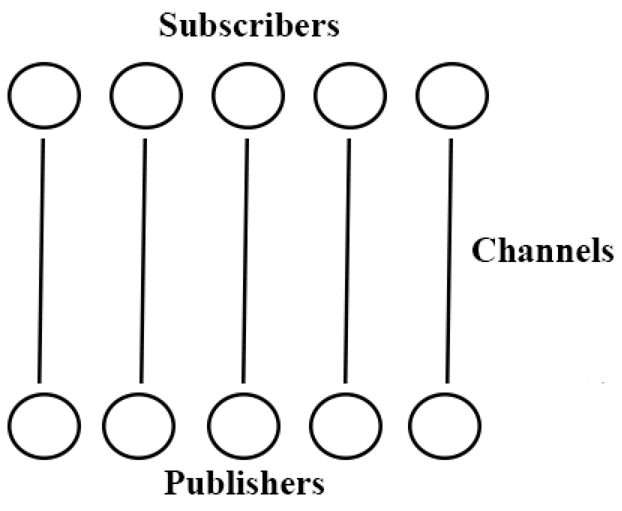


Figure 3.5: Example publisher subscriber system with channels.

Using a channel based publish/subscribe model can solve the "Event organization" architectural challenge as described in the Problem Definition chapter. Channels prevent different events from getting mixed up with one another, and facilitate organized storage and future retrieval. This is further explored and detailed in the System Design chapter.

Furthermore, the use of message queues can provide a level of fault tolerance for an event messaging system [23]. These queues can be inserted in between publishers and their channels. Having these queues allows publishers to publish events to a channel regardless if the component or subscriber at the other end of the channel endpoint is offline or online.

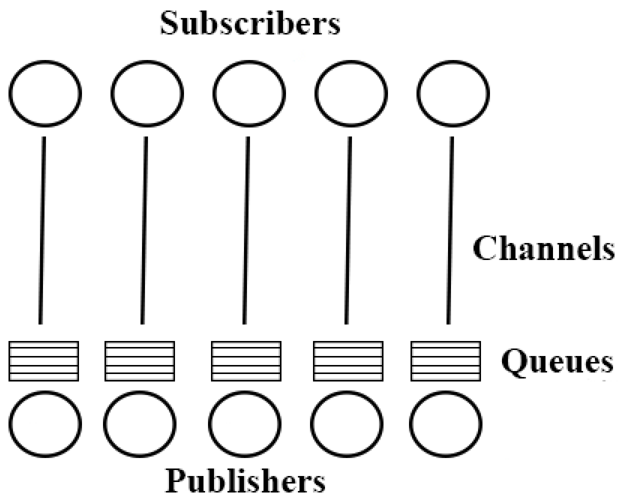


Figure 3.6: Example publisher subscriber system with channels and message queues.

Using message queues can help overcome the "Intermittent connectivity" challenge described in the Problem Definition chapter. If connection is lost at any time, events can continue to be created and stored on these queues until connection is re-established. The implementation details of these queues for the event messaging system described in this thesis is further detailed in the System Design chapter.

Within event messaging systems the number of publishers and subscribers does not have to be the same as well. Depending on the type of system or on it's intended use the number of subscribers and publishers can change. This can also change during use, for instance new publishers can be added as the number of users using a system increases. Considering the mobile space, if we modelled publishers as mobile devices and a central back-end server as a single subscriber the following publish/subscribe model could be used as shown in figure 3.7.

3.3 Cloud Computing

In 1961 John McCarthy first predicted that computation might one day be used as a utility [11]. In the 1990's this began to become a reality with the emergence of grid computing. These clusters of machines connected together provided large amounts of computational power to remote users. TeraGrid, Open Science Grid, and Earth System Grid are all examples of grid computing. However, these are all federated systems,

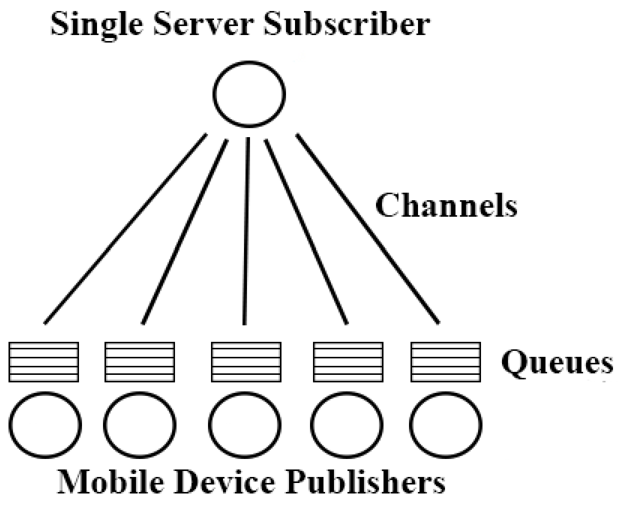


Figure 3.7: Example publish/subscribe model for use in the mobile space.

in the commercial space grid computing never developed. It in fact took another decade for commercial applications to come forth selling computation power as a utility, but this time using a different approach called cloud computing.

Cloud computing is similar to grid computing but different on a few unique points;

- Cloud computing is massively scalable with services easily provided around the world.
- Cloud computing can be delivered as an abstract entity to clients, while at the same time offering many different services.
- Cloud computing is driven by economics of scale.

The largest developers of cloud computing technology are Microsoft, Amazon, and Google. For example, popular products like Azure is from Microsoft, Google App Engine is from Google, and Amazon Elastic Compute Cloud is from Amazon. These services are provided on demand (scaling to increased or decreased loads), are delivered to clients with a layer of abstraction, and revolve around a for profit business model. Within cloud computing there are three different types of services provided being;

- IaaS (Infrastructure as a Service).
- PaaS (Platform as a Service).
- SaaS (Software as a Service).

IaaS is when infrastructure is provided as a service for developers to use. This lets developers have complete control, allowing them to install or configure anything on their virtual system. However, the largest disadvantage is that developers are also in charge of maintenance. They are in charge of installing updates, fixing bugs, managing memory, and so forth. Amazon EC2 (Elastic Compute Cloud) is one example of this type of service.

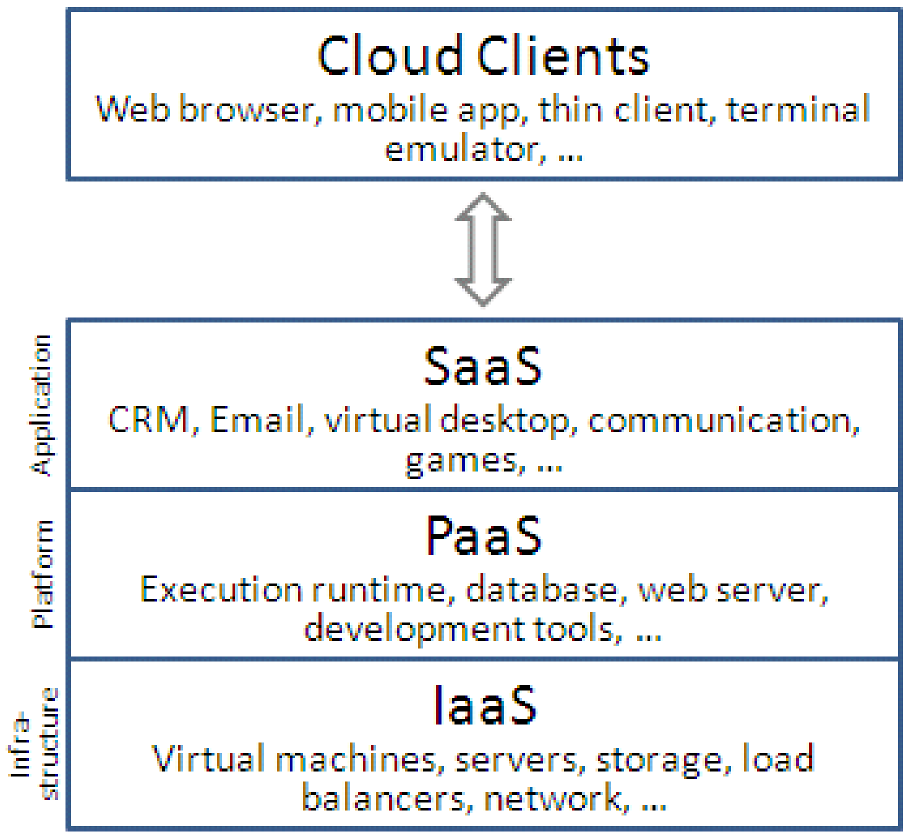

Cloud Clients
Web browser, mobile app, thin client, terminal emulator, ...
SaaS
CRM, Email, virtual desktop, communication, games, ...
PaaS
Execution runtime, database, web server, development tools, ...
IaaS
Virtual machines, servers, storage, load balancers, network, ...
Application
Platform
Infra-structure

Figure 3.8: Cloud computing services.

PaaS is when a development platform is provided to developers, with the underlying infrastructure supporting the platform hidden. For instance, developers can write their code and have the platform compile and run it, but they do not have control over the compiler or the server running their code. The greatest advantage with this model is that developers do not have to worry about back-end administration and maintenance, installing updates, scaling to meet client requests, or memory management. This can lead to quick start-ups and rapid development. However, a disadvantage is the reduced power given to developers. Without access to the infrastructure they are limited completely by the specific platform. For instance, if they want to switch operating systems for a difference in performance they would not be able to do so.

SaaS is when software is provided to developers or regular user clients. Some examples of which include Prezi, Google Docs, and Drop box. In that situation developers or other clients are provided software without any exposure to the underlying platform or infrastructure. The advantages of SaaS are its simplicity and clear business model. However, it means the service provider is in complete control, with little to no room for developer customization.

Cloud computing services (IaaS, PaaS, or SaaS) can be provided from three different cloud models. These cloud models are;

- Public Clouds.
- Private Clouds.
- Controlled Clouds.

Public clouds have their services provided to the general public, usually at a cost. While private clouds only provide services within an organization or business, and represent strictly internal processes. Differences between clouds are not usually technical; but instead relate to the security or company policies in place. Controlled Clouds go beyond private clouds by offering further security; they are not publicly available, and all communication is encrypted. Other sub type clouds also exist, with each one used depending on the situation and context.

The "Low storage and processor capabilities" challenge described in the Problem Definition chapter can be further overcome with the use of cloud computing. If servers servicing mobile devices are hosted on clouds, data storage and processing power can be offloaded away from limited mobile devices. Additionally, the architectural challenge of "Addressing scalability" is completely solved by offloading it to the cloud service provider as well. Implementation details on how the event messaging system detailed in this thesis takes advantage of cloud computing is provided in the System Design chapter.

To layout communication between clients and servers different web-service architectures are currently used. The following section describes two common ones, and details research comparing them for use in the mobile space. These two are; Representational State Transfer (REST) and SOAP (Simple Object Access Protocol) web services.

3.4 REST vs SOAP

Roy Thomas in the year 2000 with his doctoral dissertation first introduced REST [6]. SOAP on the other hand was first designed in 1998, and became a World Wide Web Consortium recommendation in 2003. Both have different advantages and disadvantages depending on the environment and use cases.

Specifically when mobile devices need to access web services, REST has emerged over SOAP as an excellent architecture. With SOAP, devices have to formulate and parse complex XML tree structures which are expensive in terms of memory and processing power. However, with REST simpler HTTP verbs can be used being: POST, PUT, GET, and DELETE. This aspect of REST allows it to provide a communication layer for accessing cloud services with lower data usage, which is well suited for the mobile space. Thus leading to the increased popularity of REST on the web as shown in Figure 3.9.

The response in REST is also more memory friendly by using common and minimal HTTP status codes. For example, to send a geocoding request to Google it simply is a GET request to a URL, followed by a "200

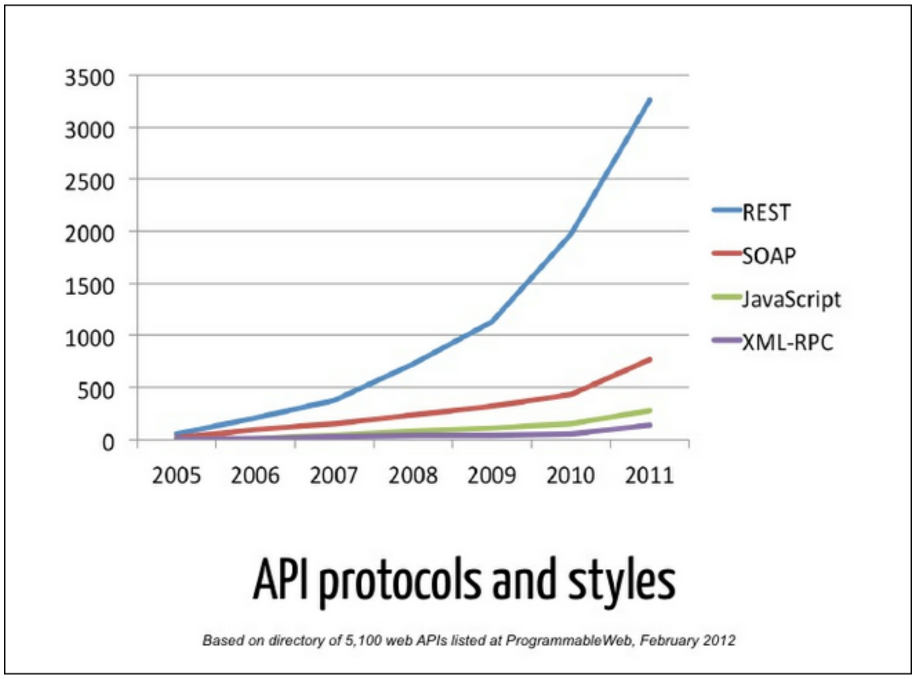


Figure 3.9: REST vs SOAP Usage on the Web.

OK" HTTP response. Any arguments or identification numbers are simply included at the end of the URL, such as "1919" in the following example request;

```
GET:
http://maps.google.com/maps/geo?1919
```

The superiority of REST in the mobile space is further shown due to the following characteristics as described in the "Using RESTful Web-Services and Cloud Computing to Create Next Generation Mobile Applications" paper [6];

- REST is stateless, minimizing the impact of network volatility.
- REST is URL based, therefore easy to invoke.
- REST responses are usually HTTP based, therefore discrete and minimizes the impact of network volatility.

The stateless aspect of REST specifically suites the mobile space, as it means the server does not hold information representing the state of the client. Instead every request from the client holds within it all the information needed for the server to service a request. This is paramount to help solve the challenge of "Intermittent connectivity" as mentioned in the Problem Definition chapter and earlier in this literature review. An error in one request does not affect the service of future requests.

As mentioned earlier SOAP does have advantages over REST for specific use cases and certain environments. For instance, SOAP allows more complex transactions beyond the simple Create Read Update Delete

(CRUD) operations of REST. SOAP also provides more information for designers, making the intent of each SOAP operation easier to understand. The protocol is also more mature and structured thus providing a greater degree of security. This for instance better suits a banking system.

3.5 Adaptive Application Management

For the purposes of testing the event messaging system, AAM features are researched and provided. Adaptive Application Management first appeared in the 1990's, with personalized user interfaces [40]. AAM is the concept of adapting an application (while running) based upon changing factors to enhance software in a variety of ways, as well as avoid common usage problems. For instance, over time software can become bloated with many features. Users end up only comprehending and using the features they need, while other features become a hindrance to the efficient use of the program. One application of AAM is to customize the GUI so that the features used most, such as in a menu, are presented first. Users are then saved from having to navigate through features rarely used and thus increasing efficiency and usability of the software.

Use case modelling is another example of AAM. It can be used to adapt the application to better suit a behavioural model detected. For instance, instead of having the same software configuration for both advanced and novice users, there can be a different configuration for each group. Meaning that if an advanced behavioural model is detected, AAM can adapt the application so that more advanced features are provided to that user automatically.

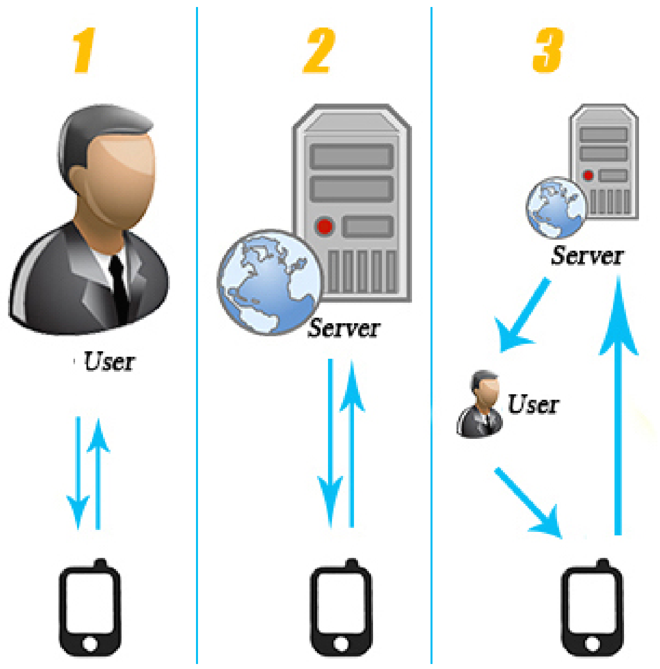


Figure 3.10: Different AAM techniques.

AAM can be implemented in three different techniques [10], as shown in Figure 3.10. The first technique puts the user in charge of adapting the application. Such as when users are given the ability to configure the arrangements of features in a menu themselves. For instance, allowing users to move the features they use the most to the top of a menu, and features they do not use much to the bottom.

The second technique puts a back-end server in charge. For instance, instead of the users manually moving features up and down, this would be done automatically by communicating feature frequency of use to a server. An example of this technique from industry, is when Google displays the most frequently searched items on it's search engine.

Finally, the third technique combines the first two approaches by letting a server be in charge but allows the user to still have some control. An example of this would be where the server, instead of providing instructions directly to the application, only provides suggestions to the users. An example from industry is when Facebook provides suggested friend contacts.

AAM can be used both in the mobile and non-mobile spaces. In the mobile space AAM offers the following advantages;

- Simplifying things for users whom in the mobile space have their level of engagement limited.
- Offloading computing power away from mobile devices and to back-end servers.
- Allowing developers to clone features and code across multiple applications.

The proceeding sections of the literature review focus on two main protocols. Research is conducted to compare and contrast them for use in the event messaging system.

3.6 HTTP

HTTP currently serves as the foundation for all communication in the World Wide Web. First introduced in 1997 in coordination by the Internet Engineering Task Force and World Wide Web Consortium. The protocol functions in a request-response paradigm, and follows a client to server model. For instance, when a web browser visits www.google.ca it functions as a client and sends an HTTP GET request to the Google server. After a period of time, the Google server returns a HTTP response with the information required to display the www.google.ca web page.

Each request in HTTP carries a request line, a header field, and a body. The request line contains the Uniform Resource Identifier (URI) and method type. The header field contains information such as the language the response should be in, the type of content in the body, and more. Finally the body contains any data to be submitted with the request such as a user name.

There are 4 main types of HTTP request methods;

- GET: These requests are only used to retrieve data, such as a web page.
- POST: These requests are used for posting data to a server, such as a user filled form.

- PUT: These requests put resources onto the server with a specified URI.
- DELETE: Theses requests are used to delete specified resources.

HTTP is a protocol that usually uses Transmition Control Protocol (TCP) as the underlying transport layer. Other transport layers exist, however, such as User Datagram Protocol (UDP).

3.7 UDP vs TCP

UDP and TCP are both underlying protocols that serve as communication backbones for large scale networks around the world including the world wide web. Other higher level protocols such as HTTP, and CoAP, then use either UDP or TCP each with varying disadvantages and advantages.

"TCP is the most commonly used protocol on the Internet" [34]. This is primarily due to the reliability it provides. During transmission a lot of data on networks is usually lost due to errors and collisions. If a message is lost on TCP, however, it is resent until all the data is correctly transferred.

Meanwhile UDP is used where speed is favoured over reliability. In this case some data can be lost during transmission, however, the data is sent faster. Examples of UDP include online media streaming since speed is critical, however, this is why media is not always the best quality.

3.8 CoAP

CoAP is another new high level protocol similar to HTTP, however, it uses asynchronous UDP instead of TCP. This approach is well suited for M2M communication and devices in the IoT. CoAP has many advantages to name just a few: it reduces the overhead on message requests, is simple, and reduces the amount of data used. It is primarily used in constrained environments such as smart energy power grids and building automation networks. Through the use of a proxy server, however, CoAP is still able to be easily integrated with HTTP. This opens the door to a great multitude of possibilities, allowing devices using CoAP to communicate with existing HTTP devices and servers which are connected to the world wide web.

CoAP has the following main features as described in "The Constrained Application Protocol (CoAP)" [36]:

- Web protocol fulfilling M2M requirements in constrained environments.
- UDP binding with optional reliability supporting uni cast and multi cast requests.
- Asynchronous message exchanges.
- Low header overhead and parsing complexity.
- URI and content-type support.

- Simple proxy and cashing capabilities.
- A stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP.
- Security binding to Data-gram Transport Layer Security (DTLS).

The client server interaction in CoAP is similar to that of HTTP. Clients, can send requests to resources with a method code and the server(s) respond with a response code. These request/response codes (which are similar to HTTP) are included in different types of messages. Because CoAP uses UDP, however, this communication is not synchronous. CoAP itself has four types of messages. These are;

- Confirmable
- Non-confirmable
- Acknowledgement
- Reset

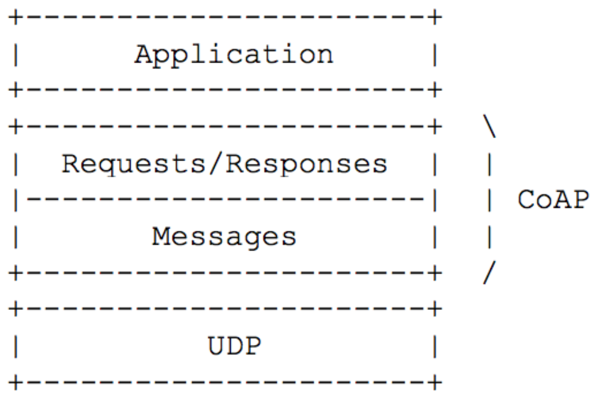


Figure 3.11: The request/response codes and the different types of messages from CoAP form two layers in the protocol (the figure used is from "The Constrained Application Protocol (CoAP)" paper [36]).

3.8.1 Message Types

Each type of message in CoAP is used for a specific purpose. For instance, in the case of common client server interaction, a Confirmable message is used when reliability is required. This type of message would continually be sent to the server until the server replies with an Acknowledgement message. To avoid flooding a system, a Confirmable message is sent with an exponential back-off iteration rate. This means that a client would iteratively send a message, for example, during the first second, again after two seconds, again after four seconds, again after eight seconds, and so on and so forth. If the server is unable to reply to the message, a message of type Reset is returned.

A Non-confirmable message is similar to a Confirmable message, however, it is used when reliability is not required. That means that an Acknowledgement message is not needed in response. This, for instance, is well suited for sensors continuously sending out data.

A Reset message is sent back when a recipient does not understand the received message or encounters an error. This can be in response to both Non-confirmable and Confirmable messages. It is commonly used to also test if an endpoint or server is still alive. This can be done by sending a flawed message on purpose, and seeing if a Reset message is returned.

3.8.2 Request/Response Codes

Request/Response codes can be included within these messages, including but not limited to the common: GET, POST, PUT, and DELETE. Thus providing similar functionality as HTTP following RESTful constraints. This also allows a CoAP system to be able to communicate with HTTP systems, such as the world wide web through a proxy server translating CoAP to HTTP and vice-versa.

Requests are made using Confirmable or Non-confirmable messages, and can contain different method codes such as GET or POST. CoAP, similar to HTTP, supports 200, 400, and 500 level response codes. However, the response mechanism is not always the same and depends on the situation. For instance, if you have a client sending a GET request on a Confirmable message and a server is able to respond, it will simply send an Acknowledgment message with the response contained within. However, if the server is not able to respond right away to a client, an empty Acknowledgment message will be sent back instead. This is to let the client know that their message has been received, and to stop iteratively re-sending the original request. When the server is ready to process the original request, it then sends a new Confirmable message with the response inside. Consequently, the client upon receiving the response finally sends back an empty Acknowledgment message as depicted in figure 3.12.

3.8.3 Message Format

CoAP messages are designed to be transported on UDP or DTLS, as mentioned in ”The Constrained Application Protocol (CoAP)”. Each CoAP message occupies the data section of one UDP datagram [36]. The structure of a CoAP message begins with just a 4 byte sized header, followed by an up to 8 byte token, followed by specified CoAP options, and finally the payload.

The header itself contains five fields. The first field is a 2 bit unsigned integer that details the version of CoAP being used. Then comes another 2 bit unsigned integer that specifies the message type (Confirmable, Non-Confirmable, Acknowledgement, and Reset). This is followed by a 4 bit unsigned integer which represents the length of the token located further in the message after the header. The fourth field, is a 8 bit unsigned

```
Client              Server
   |                  |
   |   CON [0x7a10]   |
   | GET /temperature |
   |   (Token 0x73)   |
   +----------------->|
   |                  |
   |   ACK [0x7a10]   |
   |<-----------------+
   |                  |
   ... Time Passes  ...
   |                  |
   |   CON [0x23bb]   |
   |   2.05 Content   |
   |   (Token 0x73)   |
   |     "22.5 C"     |
   |<-----------------+
   |                  |
   |   ACK [0x23bb]   |
   +----------------->|
   |                  |
```

Figure 3.12: A Get request with a delayed server response, using Confirmable and Acknowledgement CoAP messages (the figure used is from ”The Constrained Application Protocol (CoAP)” paper [36]).

code. This is where a response code (4.04 for example) or request method would be located. The last field in the header is a 16 bit unsigned integer representing the message ID.

3.8.4 CoAP for Use Within the System

The event messaging system described in this thesis targets mobile devices and not typical devices in the IoT viewpoint, which CoAP is designed primarily for. However, CoAP provides reliable and simple communication that consumes small amounts of data. This provides certain advantages when considering the challenges found in the mobile space. Furthermore CoAP can integrate with HTTP servers, and thus becomes a potential protocol for use within the event messaging system.

As mentioned in the Problem Definition chapter one of the goals in this thesis is to ”Compare and contrast HTTP and CoAP for use within the system”. This literature review has provided a technical introduction to the two protocols. The System Design chapter explains the implementation of HTTP and CoAP for use within the event messaging system.

The proceeding and last section of the literature review details a survey conducted on existing similar messaging systems. The goal of this survey is to look for further solutions to overcoming some of the challenges described in the Problem Definition chapter. Note that certain properties of these systems exist making them different than the event messaging system described in this thesis. These are;

- The data flow is only one-way (from clients to servers).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.13: The format of a CoAP message (the figure used is from "The Constrained Application Protocol (CoAP)" paper [36]).

```
       WITH COAP                    WITHOUT COAP
     IMPLEMENTATION                IMPLEMENTATION


+----------------------+        +----------------------+
|   BACK-END SERVER    |        |   BACK-END SERVER    |
+----------------------+        +----------------------+
+----------------------+
|    HTTP OVER TCP     | \      +----------------------+ \
|----------------------|  |     |                      |  |
|     PROXY SERVER     |  |     |    HTTP OVER TCP     |  |
|----------------------|  |     |                      |  |
|     COAP OVER UDP    | /      +----------------------+ /
+----------------------+
+----------------------+        +----------------------+
|MOBILE CLIENT DEVICES|         |MOBILE CLIENT DEVICES|
+----------------------+        +----------------------+
```

Figure 3.14: System communication layers with and without CoAP.

- They have not been developed primarily for use in the mobile space but have adapted to it.
- They are not open-source.
- They only function using one communication protocol (HTTP).

3.9 Survey of Similar Messaging Systems

3.9.1 Google Analytics

Google Analytics first appeared in November 2005 and has since been growing rapidly. It's goal is to provide developers information about how their applications are being used [15]. For example, developers can find out how many users are using their software at a specific period in time, or where most of their users are located. A critical drawback with Google Analytics, however, is that data cannot contain specific user details. The platform providing Google Analytics for Websites, Android, Windows, and iOS, consists of four main components as shown in figure 3.15 being;

- Collection
- Configuration
- Processing
- Reporting

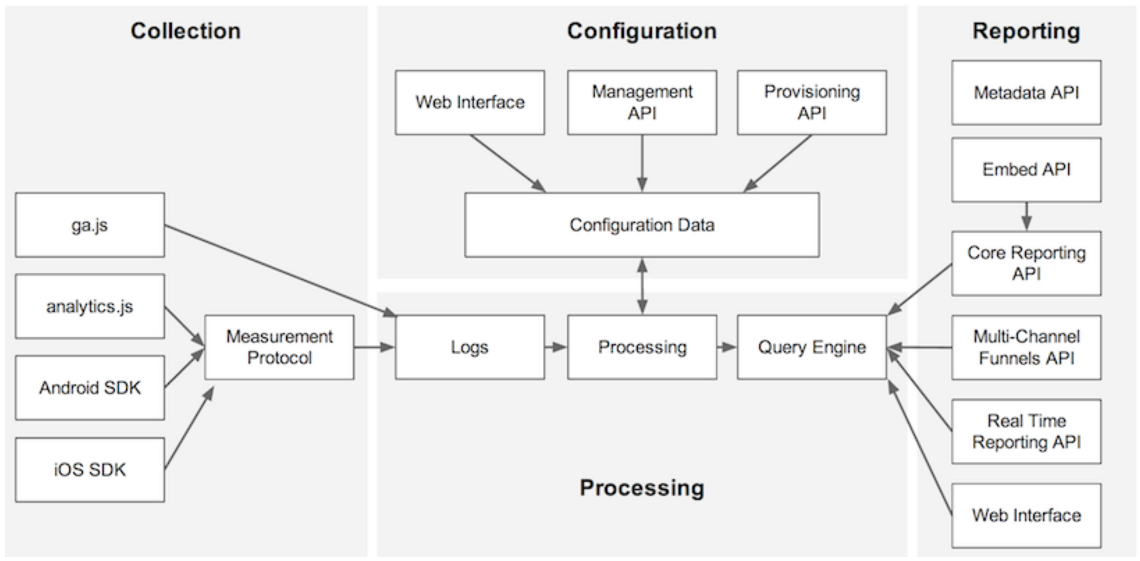


Figure 3.15: Google analytics platform components.

There are three main API's (Application Programming Interface) for collecting data in the collection component, with each API used for different data collection sources. JavaScript plug-in libraries facilitate data collection on websites. Android and iOS are also supported.

In the configuration component, developers are able to configure how they want their data to be processed through a web interface. This can also be done through management and provisioning API's.

The processing component takes the configuration settings and applies it to the collection data. The processed data is then saved for later retrieval.

A web interface finally provides access to all the processed saved data in the reporting component. Five APIs are also provided allowing many different methods to access the data. One of Google Analytics greatest advantages is showcasing data with a wealth of different analysis models and report generation options.

3.9.2 Kissmetrics

Kissmetrics advertises itself as being superior than Google Analytics by providing the ability to collect data with personal user information included [25]. Thus, later analysis on the data reveals more personalized information. The largest drawback with Kissmetrics, however, is the high price as well as the propriety nature of it. The Kissmetrics system consists of three components; .

- Back-End Servers, which store and process all the events sent from different users.
- A JavaScript Plug-In, which gets installed on the client side and is responsible for logging and sending events to the back-end servers.
- A Website, which is where developers can access and analyse their data, developer account registration, payments, and configurations.

3.9.3 Conclusions from Survey

In the problem definition section, one of the challenges found in the mobile space was "The existence of many platforms". Both the Kissmetrics and Google Analytics systems make use of generic frameworks and languages to allow cross platform compatibility. This presents a solution for the event messaging system detailed in this thesis as well. The implementation of which is detailed in the proceeding System Design chapter. Additionally, other key points noted are;

- A website component allows developers the ability to interact with the data collected, register with the system, and perform other administrative tasks.
- A client end Plug-In facilitates efficient data collection and transmission to servers.
- Storing personal client information provides additional features, however, it is controversial.

3.10 Summary

The aim of the literature review is to detail research performed in order to develop an event messaging system for the mobile space. This includes research in finding solutions to challenges found in the mobile space as well as on AAM who's features are used for testing purposes. Furthermore, HTTP and CoAP are introduced for use in the event messaging system.

The first section of the literature review focuses on the emerging field of Mobile Computing. This includes a definition, introduction, and description of Smart Phones, the Mobile Space, and data formats used therein. This section is crucial to understanding some of the challenges in the mobile space and avenues to finding their solutions.

The second section of the literature review covers the field of event messaging systems. An introduction is given to the publish/subscribe messaging paradigm with an emphasis on the channel based approach. Messaging queues and how they function is also detailed. This section concludes with the outline of a messaging model for use in the mobile space.

The third section of the literature review covers the field of Cloud Computing, and the significant amounts of computing that take place on remote servers (the cloud). Cloud computing provides new avenues for connections between computing devices, allowing them to share storage and work together in a variety of ways. This is well suited for the mobile space, and detail is given on how cloud computing can be used to help solve some of the challenges mentioned in the Problem Definition section.

The fourth section of the literature review covers REST and SOAP which are both common web service architectures, that facilitate communication between clients and servers. A comparison between REST and SOAP is provided with detail specifically on which is better suited for the mobile space. One important aspect of REST, is that all requests to a server include all data required to represent the current state of a user. Therefore, when the server receives a request, that request is not dependent upon previous requests.

The fifth section of the literature review covers the field of AAM, which is the concept of adapting an application while it is being used to increase efficiency and usability. This adaptation can be based on a variety of changing factors, such as user behaviour. Furthermore, there are also different types of adaptation as Leah Findlater and Joanna McGrenere in their paper A Comparison of Static, Adaptive, and Adaptable Menus presented and analysed. These different types are outlined with details covering both the front visible features and the back-end system dynamics required.

The sixth section of the literature review covers the HTTP and CoAP protocols. The standard and most used protocol is HTTP, and is used for the world wide web. However, another alternative protocol for communication is called CoAP and uses UDP instead of TCP. CoAP like HTTP follows the REST architecture, however, it is less complex and uses much less data as it is designed for an IoT environment. These properties make CoAP potentially well suited for overcoming some of the challenges found in the mobile space.

Finally, in the seventh section, a survey of existing messaging systems similar to the system described in this thesis is presented. Similarities and differences are outlined with a focus on deriving lessons to help overcome current and future challenges.

The following table provides an executive summary of key papers included in the literature review. Detail is also given on each paper's greatest research contribution, as well as potential remaining issues.

Table 3.1: Literature Survey Papers

Paper	Solution Extracted	Current Open Issues
Cloud Computing and Grid Computing 360-Degree Compared [11]	This paper provides the understanding required to be able to use cloud computing to implement the back-end server required.	In the long term an issue may be encountered with incurring costs, as cloud computing follows a business model.
Design of MOBILE MOM: Message Oriented Middleware Service for Mobile Computing [23]	This paper provides an outline to the design of message queues providing fault tolerance in an event delivery messaging system.	It is unknown how messaging queues can be implemented in the event messaging system described in this thesis.
A Comparison of Static, Adaptive, and Adaptable Menus [10]	This paper provides specific descriptions of how AAM can be implemented, and the different techniques possible.	NA
Using RESTful Web-services and Cloud Computing to Create Next Generation Mobile Applications [6]	This paper specifically describes how RESTful web-services is the best choice for connecting devices in the mobile space to the cloud using HTTP.	There are a few other possible protocols that are not discussed in this paper however.
The Constrained Application Protocol (CoAP) [36]	This paper describes a new communication protocol called CoAP, which is similar to HTTP, also follows REST constraints, but uses UDP instead of TCP.	The addition of a required CoAP to HTTP proxy server introduces a layer of complexity, thus the net affect of using CoAP is unknown.

Chapter 4

System Design

This chapter details the design and development of an event messaging system for use in the mobile space. Solutions to overcome the challenges detailed in the Problem Definition chapter are taken from the Literature Review and implemented. Furthermore, the implementation details of HTTP and CoAP is also provided. Finally, the AAM features used to test the system are outlined both in implementation and use.

4.1 Design Theory

The primary goal of this thesis is to develop an event messaging system. This system consists of one cloud hosted server, client side local databases, an API, and a website. Figure 4.1 shows these different components and the flow of data from users.

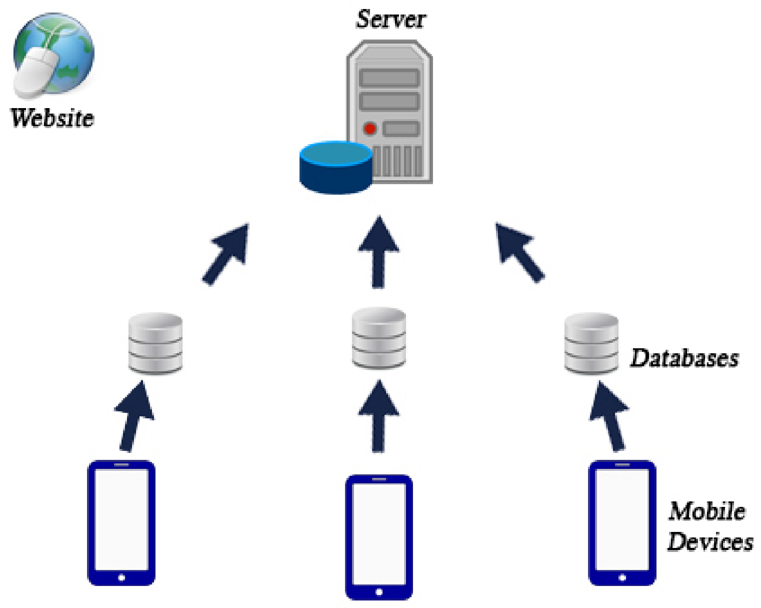


Figure 4.1: The data collection layout.

The data flow of the system is modelled on the publish/subscribe paradigm presented in the Literature Review as shown in Figure 4.2. Here the mobile events are the publishers, the local buffer databases are the message queues, and the central server is a single subscriber. This design is chosen to address the architectural challenge of event organization. All communication within the system uses RESTful web-services. This design choice was made as it was derived in the Literature Review to be a better option.

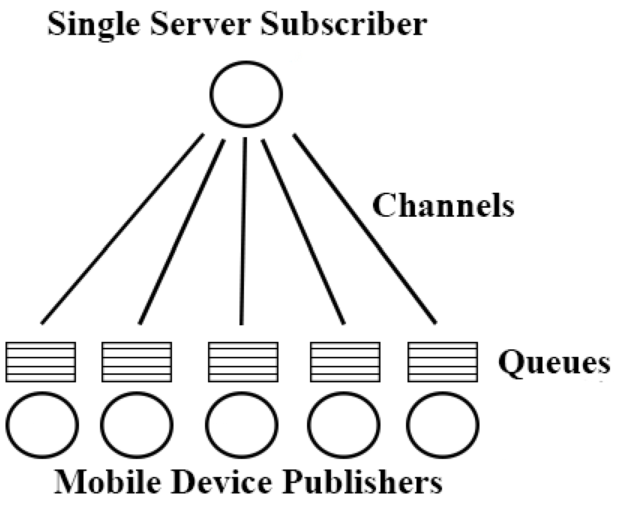
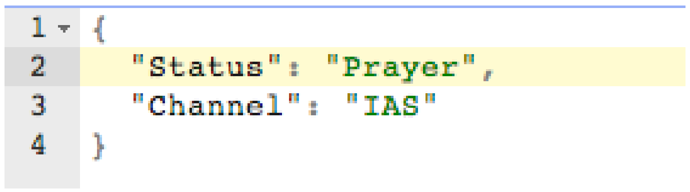


Figure 4.2: Publish/subscribe model with single subscriber.

4.1.1 Events

For the use cases of this system as shown in the Literature Review, the JSON data format design choice addresses the "Low Bandwidth availability", "Low storage and processor capabilities", and the "Contrasting different data formats" challenges described in the Problem Definition section. All the messages in the messaging system are modelled as events, with each event as a JSON object. Events are created on mobile clients (publishers), and then sent to the server (subscriber). Figure 4.3 shows a sample JSON event with two fields.



```
{
  "Status": "Prayer",
  "Channel": "IAS"
}
```

Figure 4.3: JSON structure of an event object.

Because this event is being used for GUI Menu Ordering the status field represents information about a specific menu option. Specifically it describes a feature being selected in a menu. The data contained within each event is different depending on the specific AAM feature being used, thus for the Proficiency User Modelling feature the data within events is different.

The channel field represents a specific event channel name. The channel name might differentiate between users, mobile applications, companies, platforms, countries, and so forth. Similar to TV channels every channel is unique but not confidential. Figure 4.4 shows collections of events in different channels.

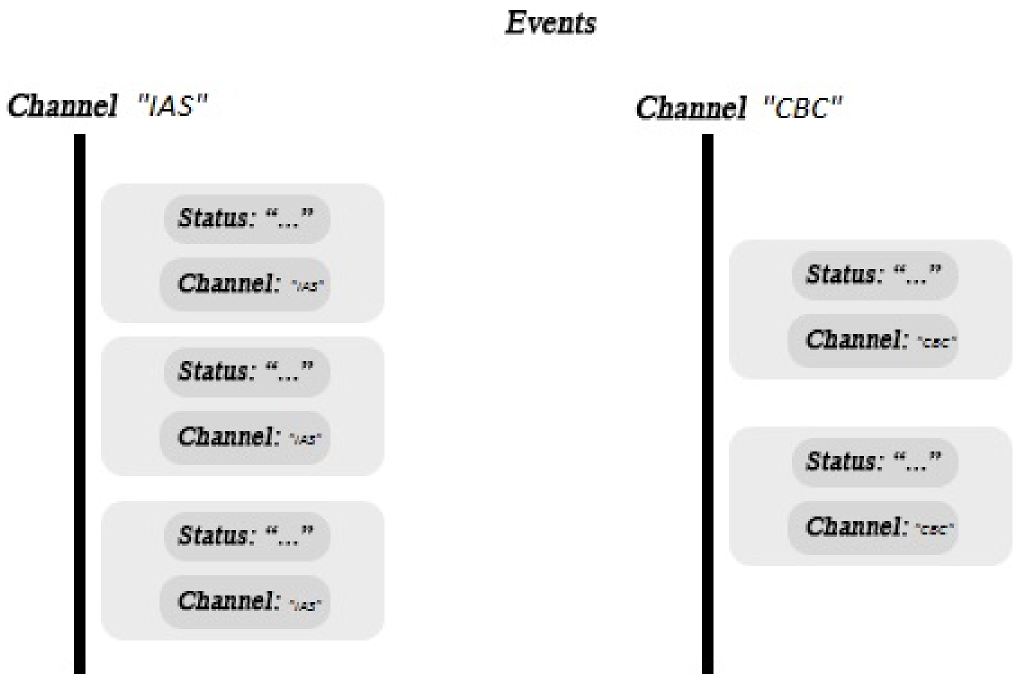


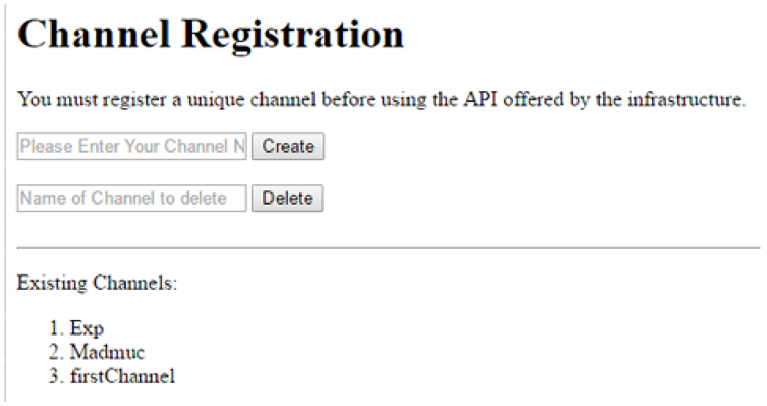
Figure 4.4: Event objects are sent to the server organized into channels.

To create a channel, developers are required to go to the system website and register a new channel with a unique name. At the same place they may also delete an existing channel. A screen shot of this website page is shown in Figure 4.5.

4.1.2 Website

The website functions primarily to let developers register their channels. This provides a mechanism to make sure that channels remain unique, and therefore prevents events from getting mixed up. The website is hosted on the same cloud server that collects all the events.

The website also functions as a platform to provide documentation on how to use the API, the AAM features offered, and how to configure the client side database. This documentation may grow over time as more features are added, and the API is correspondingly updated. Additionally, experimentation simulation controls are also hosted on the website.



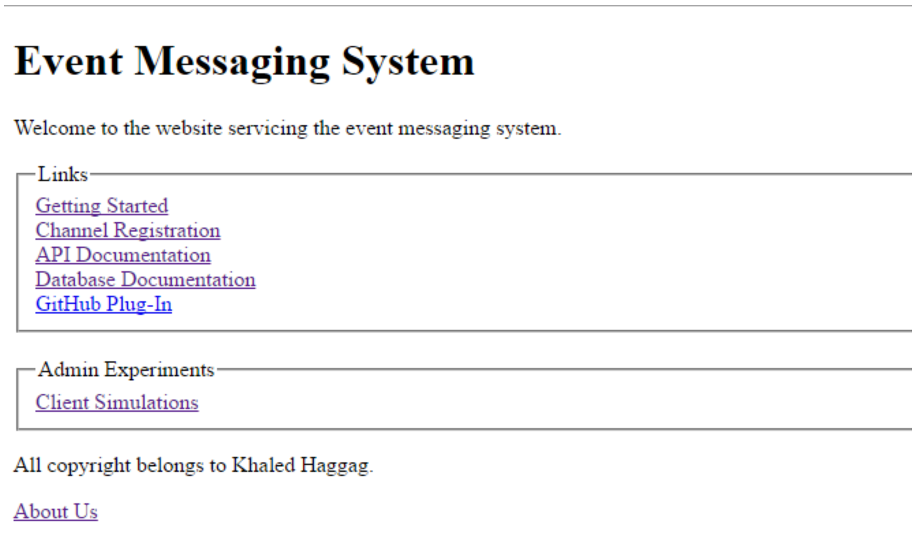
Channel Registration

You must register a unique channel before using the API offered by the infrastructure.

Existing Channels:

1. Exp
2. Madmuc
3. firstChannel

Figure 4.5: Screen shot of channel creation and deletion page.



Event Messaging System

Welcome to the website servicing the event messaging system.

All copyright belongs to Khaled Haggag.

Figure 4.6: Screen shot of the main page on website.

4.1.3 The Back-End Server

Servicing the mobile devices and website is a cloud-hosted server residing on Google App Engine. Google App Engine is a PaaS offered on a public cloud in a pay as you go manner. This design choice provides the following advantages as mentioned in the Literature Review;

- Rapid server and back-end database development, with minimal maintenance required.
- Accessibility from anywhere around the world with an internet connection.
- On demand computation, automatically scaling with increased and decreased client demand.
- Available computation and data storage offloading from mobile devices.

The last two advantages specifically address the ”Low storage and processor capabilities” as well as ”Addressing Scalability” challenge. For the server back-end database, the accompanying Google Data Store is used. This data store is similar to a Structured Query Language (SQL) database, it provides storage with good partition tolerance, availability, and eventual consistency.

The server itself is split into many different modules, with each module servicing a specific AAM feature or system component. Following RESTful web services, the modules handling specific features each have a POST and GET handler as well as a unique URI. The POST handler receives events that are sent from clients to the server, and stores them. Meanwhile the GET handler receives requests from clients and responds with the data required for specific features. For instance, this is the python module handling the GUI Menu Ordering feature;

```
class MenuOrdering(webapp2.RequestHandler):
    def get(self):
        channel = self.request.get('Channel')
        channels = Channel().all().filter('Name',channel).count()
        events = Ordering_Event().all().filter('Channel',channel).order('Status')
        returnJson = []
        tmp = []
        i = 0

        if channels == 1:
            if events.count(1):
                tmp.append([])
                tmp[i].append(0)
                tmp[i].append(events.get().Status)

            for event in events:
                if tmp[i][1] == event.Status:
                    tmp[i][0] += 1
                else:
                    i += 1
                    tmp.append([])
                    tmp[i].append(1)
                    tmp[i].append(event.Status)

            tmp.sort(key = lambda ev: ev[0])
            tmp.reverse()
            results = tmp
            for result in results:
                returnJson.append({'Status':result[1]})

            self.response.write("%s%s"%(urllib2.unquote(self.request.get('callback')),json.dumps(returnJ
        else:
            self.response.write("Your Channel: " + channel + ". Is Not Registered")



    def post(self):
        event = Ordering_Event()
        event.Channel = self.request.get('Channel')
        channels = Channel().all().filter('Name',event.Channel).count()
        event.Status = self.request.get('Status')

        if channels == 1:
            event.put()
            self.response.write("Success. "+event.Channel + " " + event.Status)
        else:
            self.response.write("Your Channel: " + event.Channel + ". Is Not Registered")
```

Justification for Using Google App Engine

Many different options exist for developing a back-end server. Google App Engine was specifically chosen to benefit from the advantages as previously described. Having a PaaS cloud service is most optimal for the use cases of the event messaging system described, which does not require custom customization's to the server or complex services outside what Google App Engine can provide.

4.1.4 Client Databases

As described in the Literature Review messaging queues in publish/subscribe models provide a level of fault tolerance to messaging systems, and thus addresses the "Intermittent Connectivity" challenge. The system design includes local client databases which act as the messaging queues. Events are stored and saved in a queue on each database, before being sent to the server as shown in Figure 4.7. If for a duration a device has no connection, or the connection is interrupted, the events simply remain on the database. Once connection is re-established, the events are sent to the server, and removed from the database in the same order they are stored.

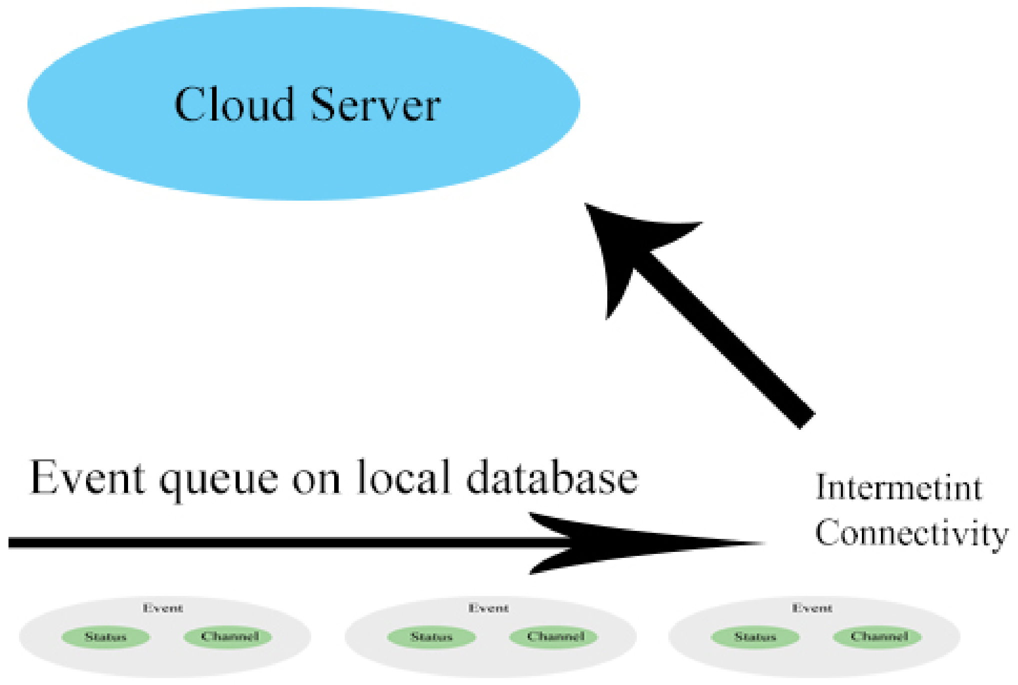


Figure 4.7: Events queued on local database.

The database uses SQLite [37], a lightweight and free database commonly used by mobile developers. SQLite is well suited for mobile devices because it is cross platform compatible, reliable, and performs well in low memory environments. Details for developers to configure the local database are on the website.

4.1.5 Protocol Alternatives

By default the event messaging system uses HTTP. However, another version is designed that uses CoAP. This consequently allows research to take place comparing and contrasting between HTTP and CoAP within the system.

CoAP Version of Event Messaging System

As described in the Literature Review, CoAP can potentially minimize data usage and increase simplicity and speed within the system. To include CoAP, however, a proxy server is required and placed in-between the cloud hosted Google App Engine and client mobile devices. The URL for this proxy server is as follows;

$$http://khaledyhaggag.centralus.cloudapp.azure.com/$$

The proxy server is written using Node JS and hosted on the Microsoft Azure cloud. The following is a snippet of code from the proxy server receiving a CoAP GET request and then sending a HTTP GET request;

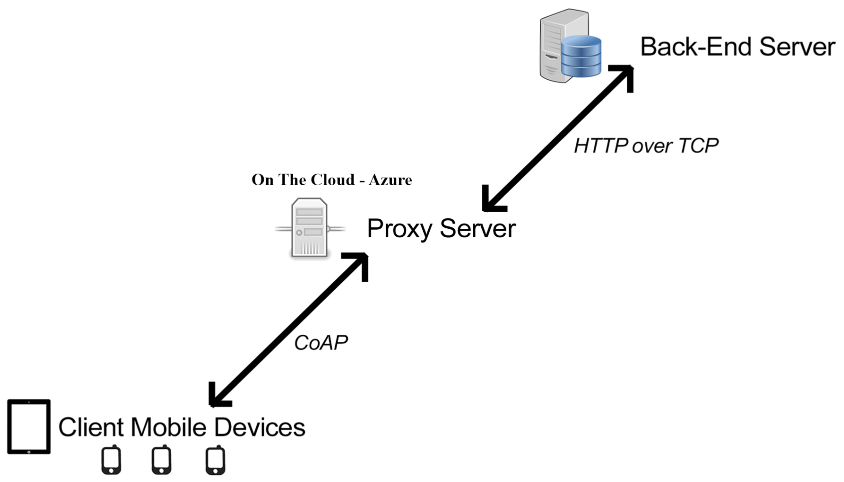


Figure 4.8: The proxy server converts CoAP communication to HTTP and vice versa.

```
if(req.method == 'GET'){
//CoAP request received is a GET request
        //generate http get request and send it

        //Parse data out of CoAP GET request and prepare it for HTTP
        var json = JSON.parse(req.payload)
        var data = querystring.stringify({
            Channel: json.Channel
        });

//Prepare HTTP GET request options
        var options = {
            host: 'khaledyhaggag.appspot.com',
            port: 80,
            path: '/menuordering',
            method: 'GET',
            headers: {
                'Content-Type': 'application/x-www-form-urlencoded',
                'Content-Length': Buffer.byteLength(data)
            }
        };

//Prepare HTTP GET request, and return CoAP response to original CoAP request
```

```
        var httpReq = http.request(options, function(httpRes) {
            httpRes.setEncoding('utf8');
            httpRes.on('data', function (chunk) {
                //console.log("body: " + chunk);
                res.end(chunk + " /n")
            });
        });

//send HTTP GET request with data from original CoAP GET request
        httpReq.write(data, function(err) {
             httpReq.end();
        });
}
```

With the CoAP version complete, experiments are conducted to compare and contrast between HTTP and CoAP. Their results can be found in the Results and Conclusion chapter. Beyond the addition of a CoAP proxy server translating the CoAP communication to HTTP, the client devices are given access to new API connections just for CoAP. A CoAP library is also included on the client devices allowing them to send and receive Confirmable Asynchronous CoAP messages as described in the Literature Review chapter. Beyond these changes the system as designed remains the same.

4.2 Provided AAM Features

The proceeding AAM features are provided by the event messaging system for testing purposes. Currently the design includes two features being GUI Menu Ordering, and Proficiency User Modelling.

4.2.1 GUI Menu Ordering

The GUI Menu Ordering feature enables developers to order a set of menu options according to client use frequency. For instance, a menu using this feature automatically has the most used options at the top, and the least used at the bottom. The purpose of this feature is to increase user satisfaction, usability, and efficiency.

Developers are provided a URI specific to this feature, which they use to send and receive data. To keep track of a menu option use frequency, every time a menu option is selected an event is fired and sent to this URI in a POST request. GET requests to this URI then return the ordered menu options.

At first a lot of variability on a menu using GUI Menu Ordering is expected, with options switching back and fourth relatively quickly. However, as the menu is used certain patterns will emerge making the menu more stable.

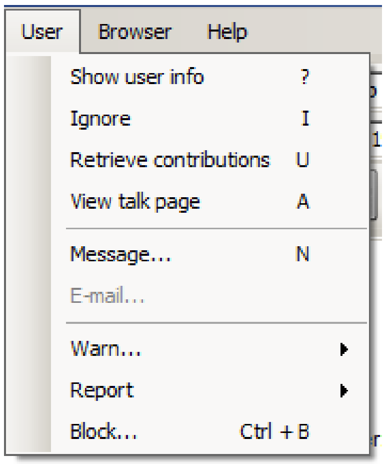


Figure 4.9: The features in this menu for instance, are ordered top down according to how often they are used.

4.2.2 Proficiency User Modelling

Proficiency User Modelling allows the adaptation of a mobile application depending on the proficiency level of a user. This is analysed based on how often a user uses a mobile application; such as monthly, weekly, or daily. Based on these frequencies there are three levels of proficiency a user can have being: novice, intermediate, and advanced. These are described as follows;

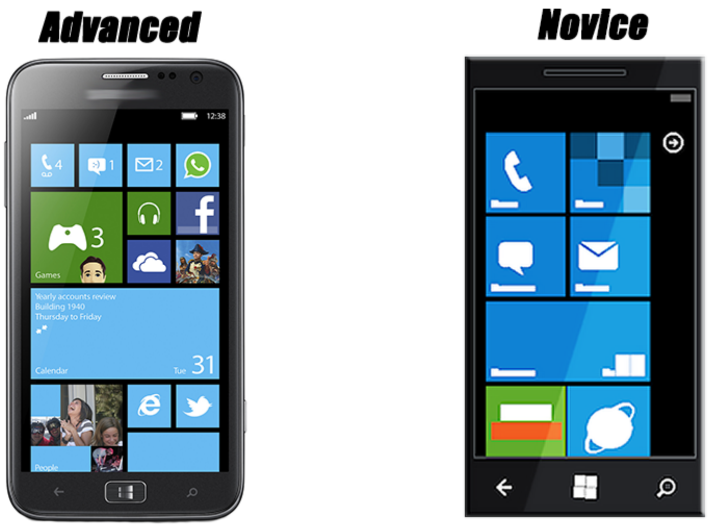


Figure 4.10: Novice vs. advanced main page adaptation example.

- All users begin with the most beginner level being 'novice', and remain at this level unless their average use of the application exceeds once a month.
- The next proficiency level is 'intermediate', and reflects application usage of at least once a week.

- Finally the 'advanced' level reflects application usage of at least once a day.

Figure 4.10 shows an example mobile application adaptation dependent on the proficiency level of a user. Events measuring application use and responses from the server reflecting the proficiency level are sent and received on a specific URI supplied to developers. Similar to the Menu Ordering feature this is done through GET and POST requests. This facilitates developers to configure application adaptation depending on the proficiency level of users. For instance, applications can reveal new features to advanced users that novice users do not have access too.

4.2.3 The Client Side API

There are 4 API service endpoints included in the system design. Each AAM feature is given two endpoints each, with one for uploading data (events) and another for attaining the result. Thus each feature has a service endpoint for POST requests and another for GET requests. The HTTP endpoints are;

```
GET:
http://khaledyhaggag.appspot.com/menuordering
Fields: Channel

POST:
http://khaledyhaggag.appspot.com/menuordering
Fields: Status, Channel

GET:
http://khaledyhaggag.appspot.com/usermodelling
Fields: Channel, Username

POST:
http://khaledyhaggag.appspot.com/usermodelling
Fields: Channel, Username, Date
```

GET requests sent to the GUI Menu Ordering service endpoint would return a JSON containing an array of status's in order according to status frequency. Similarly GET requests sent to the Proficiency User Modelling endpoint return a JSON with the user's proficiency level. Note, that in both GET requests the channel field is required in order to grab the appropriate data.

The POST requests to each respective endpoint web service is for posting the events containing the data required for the AAM features. JSON is again used, and apart from feature data, the channel field is again included to organize the events. Note that only one event can be sent at a time per request. Encryption and security certificates are also supported and are accessible by switching to HTTPS for each endpoint.

The CoAP API

Alternatively for the CoAP side of the system there are 4 corresponding API endpoints also using JSON with the same data parameters. However, unlike the previous HTTP endpoints which connect directly with the systems back-end server, the following are connected to the CoAP proxy server.

```
GET:
coap://khaledyhaggag.centralus.cloudapp.azure.com:5683
Fields: Channel
```

```
POST:
coap://khaledyhaggag.centralus.cloudapp.azure.com:5683
Fields: Channel, Status
```

```
GET:
coap://khaledyhaggag.centralus.cloudapp.azure.com:5681
Fields: Channel, Username
```

```
POST:
coap://khaledyhaggag.centralus.cloudapp.azure.com:5681
Fields: Channel, Username, Date
```

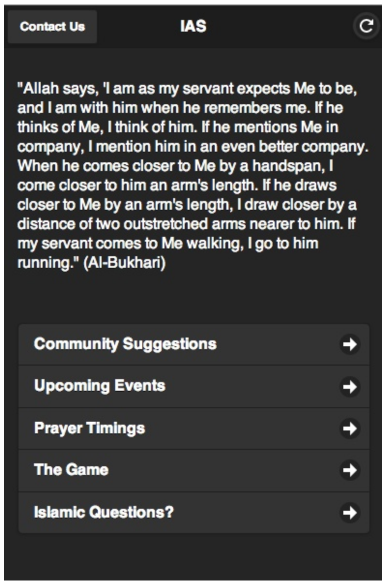
Port 5683 is used for the GUI Menu Ordering feature and port 5681 is used for the Proficiency User Modelling feature. Note that going down one port enables CoAP with advanced encryption standard (AES) to the proxy server and HTTPS from there to the back-end server. For example, the following is the endpoint for sending a GET request on the GUI Menu Ordering feature using AES encryption (port 5682);

```
GET:
coap://khaledyhaggag.centralus.cloudapp.azure.com:5682
Fields: Channel
```

4.3 Prototype Stage

4.3.1 IAS Mobile Application

Early during the development of this research, IAS (Islamic Association of Saskatchewan) presented a unique opportunity. In their goals to engage the local Muslim community, they wanted to develop a mobile application. A prototype mobile application was developed for them, and it was also used to perform a small proof of concept test for the system developed as part of this thesis. Specifically, the aim was to insert the GUI Menu Ordering feature into the IAS prototype.



"Allah says, 'I am as my servant expects Me to be, and I am with him when he remembers me. If he thinks of Me, I think of him. If he mentions Me in company, I mention him in an even better company. When he comes closer to Me by a handspan, I come closer to him an arm's length. If he draws closer to Me by an arm's length, I draw closer by a distance of two outstretched arms nearer to him. If my servant comes to Me walking, I go to him running." (Al-Bukhari)

Figure 4.11: Screen shot of main page of the IAS prototype.

The IAS prototype used HTML 5, JavaScript, jQuery, Local Storage, and Phone Gap for development. The application then connected to the system's back-end server to access the GUI Menu Ordering feature.

The main page of the IAS prototype consisted of a menu with a set of options. The GUI Menu Ordering feature was then used to automatically adapt the menu. Every time a user chose a menu option, an event was created with the status field representing the menu option. These events then were all sent to the back-end server. When the application loaded the main page it would then retrieve the GUI Menu Ordering result. Consequently, the menu options would be ordered top down, depending on the frequency of use by users.

Furthermore the IAS prototype was run on iOS, Android, and Window's devices. This demonstrated that the "The existence of many platforms" challenge described in the Problem Definition chapter was indeed overcome following solutions derived from the Literature Review.

4.3.2 DreamIT Mobile Application

A few months after the IAS application was completed, another opportunity to test the system arose. DreamIT a small Saskatchewan start up company, also wanted to develop a prototype mobile application. An agreement was made similar to the agreement with IAS.

This application was developed using Xamarin Forms, allowing the application to be available on tablets and phones; as well as Android, iOS, and Windows. The mobile application consisted of a client side SQLite database, a Google App Engine server, and an existing PHP server.

The Proficiency User Modelling feature was used within the DreamIT application. This feature configured the main page according to the proficiency level of each user. Novice users, for instance, simply had nine icons presented as shown in figure 4.12. While intermediate and advanced users could then have different configurations for their main page.

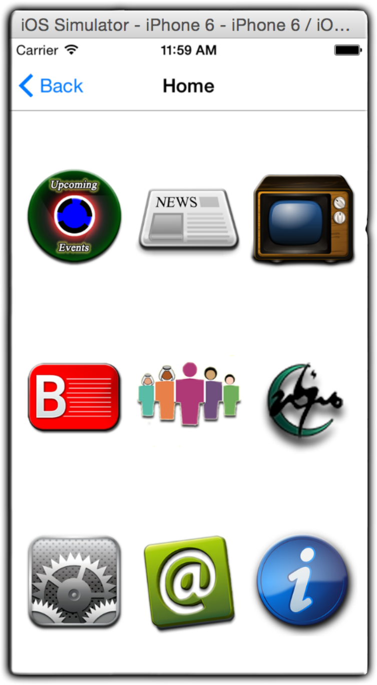


Figure 4.12: Main page of DreamIT mobile application.

4.3.3 Results from Prototype Applications

The IAS prototype mobile application successfully demonstrated GUI Menu Ordering. Feedback specifically regarding the AAM feature was well, and users noted how it increased the speed of which they used the application. Regarding the DreamIT application, results showed a successful application of the Proficiency User Modelling feature. Feedback from testers again was positive.

This concluded the pre-experimentation stage. Core components and concepts in the event messaging system were tested and shown to work successfully. These were as follows;

- Server client communication, using a Google App Engine cloud hosted server following RESTful protocols.
- Storing events on a local client database as a buffer.
- Showcasing the benefits of GUI Menu Ordering and Proficiency User Modelling.

This stage was significant in showing the successful solutions to some of the challenges outlined previously in the Problem Definition Section. For instance, Intermittent Connectivity was mitigated by using local storage as a buffer. Most importantly this stage showed that the fundamentals were working, such as being able to connect to the cloud server.

Chapter 5
Experiments

The following chapter details the experimentation's performed on the event messaging system. The aim of the experiments is to see if the research goals stated in the Problem Definition chapter have indeed been met.

5.1 Internal Testing Stage

The internal testing stage consists of testing within the MADMUC lab at the University of Saskatchewan. The entire system was tested with the Menu Ordering AAM feature, complete website, and using both CoAP and HTTP on a sample mobile application developed just for MADMUC. This served as an alpha release, demonstrating the successful development of the event messaging system and not just parts of it as with the prototype applications. The proceeding section of code, is from the mobile application sending a GUI Menu Ordering event from it's main menu;

```
Button btnHttp = (Button) findViewById(R.id.btnHttp);
btnHttp.setOnClickListener(new View.OnClickListener() {

@Override
    public void onClick(View view) {
new DownloadTaskHttp().execute(
"http://khaledyhaggag.appspot.com/menuordering? Channel=Madmuc&Status=0");
    }
});
```

5.1.1 Intermittent Connectivity Test

During the internal testing stage, an experiment is conducted to evaluate that the goal of overcoming the "Intermittent connectivity" challenge defined in the Problem Definition chapter has indeed been met. The experiment consists of sending 15 simulated requests three times. Each iteration is conducted a little differently as defined as follows;

- 1st Iteration: First 5 requests are sent with an interrupted connection, then 10 requests are sent with a normal connection.
- 2nd Iteration: First 5 requests are sent with a normal connection, then 5 requests with an interrupted connection, then 5 requests with a normal connection.
- 3rd Iteration: First 10 requests are sent with a normal connection, then 5 requests with an interrupted connection (connection is restored at the end, but no additional requests are sent).

An interrupted connection is defined as a sudden loss in connection. On the iPod devices this is represented with turning the WIFI connection off during the test. At the end of each iteration, a measurement will be taken verifying that all 15 requests have indeed been sent through to the server.

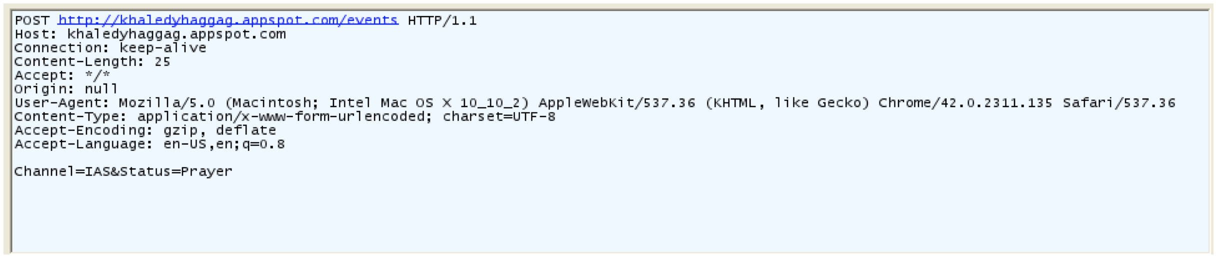
This experiment also serves as a test to see if the "Local database configuration" architectural challenge as defined in the Problem Definition chapter has indeed been met. If the database configuration detailed in the System Design chapter is flawed this experiment will fail.

5.2 Automated Experimentation Stage

This stage expands upon the previous experimentation stages and further tests the system while proving that the research goals defined in the Problem Definition chapter have been achieved. To ensure a high degree of accuracy for the results a large scope was required and an experimental server was configured to automate the testing. The tests consist of a simulated client sending requests to the back-end server. These tests are available online at the following URL;

$$https://khaledyhaggag.appspot.com/experiment$$

To ensure the authenticity of the simulated client, each simulated request is exactly the same as a real request. This is made possible by duplicating requests collected by Fiddler from the internal testing stage.



```
POST http://khaledyhaggag.appspot.com/events HTTP/1.1
Host: khaledyhaggag.appspot.com
Connection: keep-alive
Content-Length: 25
Accept: */*
Origin: null
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.135 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8

Channel=IAS&Status=Prayer
```

Figure 5.1: The raw data in a simulated request.

The proceeding sections describe how the different experiments are constructed. These three experiments consist of sending requests to the system's cloud server and model a client using the GUI Menu Ordering AAM feature. Thus each request is an event, and contains a status and a channel field within the payload. Each experiment focuses on testing different objectives.

5.2.1 HTTP and HTTPS Test

The first experiment consists of sending a burst of 1111 simulated requests 8 times using HTTP. This is then repeated using HTTPS instead of HTTP. With HTTPS, each request is encrypted with AES on Transport Layer Security (TLS) and uses a security certificate to verify the identity of the server. This is then all repeated again with sending just 1 simulated request 8 times. The objective of this experiment is to evaluate the performance and data usage of HTTPS and HTTP. A visualization of this experiment is shown in Figure 5.2.

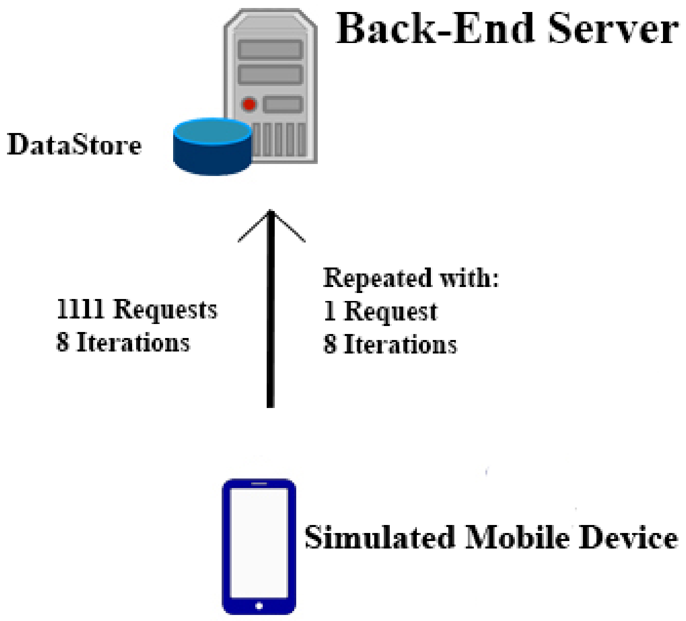


Figure 5.2: Visualization of HTTP and HTTPS Test

5.2.2 CoAP and CoAP with AES Encryption Test

The second experiment is similar to the first experiment except that CoAP is used. To do so the CoAP proxy server is used. This is also repeated using AES encryption on CoAP from the client to the proxy and HTTPS from the proxy to the back-end server. The objective of this experiment is to evaluate the performance and data usage of CoAP with and without AES encryption. Additionally when compared with the results of experiment 1, it will also provide an evaluation of HTTP and CoAP. This thus provides the data required to achieve the goal to "Compare and contrast HTTP and CoAP" as stated earlier in the Problem Definition chapter.

5.2.3 HTTP Load Bearing Test

The third experiment is a load bearing experiment to test the system itself. The experiment has 4 phases with each phase increasing the number of requests (10 to 710 to 1410 to 2110 requests). Each phase is repeated 11 times. This experiment provides an evaluation to see if the architectural challenge of scalability mentioned in the Problem Definition chapter was indeed overcome.

Chapter 6
Conclusion and Future Work

6.1 Results

Table 6.1 contains the results of the HTTP and HTTPS test, as well as the CoAP and CoAP with AES Encryption Test. Each iteration consists of sending a burst of 1111 requests. There are 8 iterations in total. The round trip time of all the requests in each iteration is given in seconds. This is measured by a timer which begins once the first request is sent to the server and ends when the response of the last request is received back. In total HTTP had an average time of 235.375 seconds and HTTPS had 416.375 seconds. Each HTTP burst consumed 564388 bytes while each HTTPS burst consumed 565499 bytes in total, from the client side.

Table 6.1: HTTP vs HTTPS vs CoAP vs CoAP AES. From Mac Local Machine (Results in seconds).

Iteration	HTTP	HTTPS	CoAP	CoAP AES
1	239	392	6.8	13.3
2	232	388	6.1	12.7
3	233	408	6.9	7.1
4	232	408	7.5	12.8
5	242	382	5.9	12.7
6	259	422	6	10.4
7	249	436	7	10.6
8	253	487	7.2	11.9
AVG	242.375	415.375	6.675	11.4375

The CoAP iterations had an average round trip time of 7.9625 seconds, and 12.4625 seconds if AES encryption was included. Data consumption was 68882 bytes and 69993 bytes respectively. Note that the HTTP requests used TCP and the CoAP requests used UDP. The data in table 6.1 was collected from a simulated client running on a local Mac machine.

The results from table 6.1 were expected following the research in the literature review. However, the difference in round trip time between CoAP and HTTP (242 vs 7 seconds) was larger than anticipated. This difference is further investigated with further experimentation the results of which are in table 6.3.

Table 6.2: HTTP vs HTTPS vs CoAP vs CoAP AES. From Windows Local Machine (Results in seconds).

Iteration	HTTP	HTTPS	CoAP	CoAP AES
1	227	411	12.7	17.2
2	235	429	7.1	16.7
3	246	411	9.7	15.7
4	230	424	6.2	12.2
5	233	421	5.7	8.9
6	235	423	9.7	8.1
7	241	406	7.2	8.5
8	236	406	5.4	12.4
AVG	235.375	416.375	7.9625	12.4625

To see if the local machine used had an effect, the experiments were repeated on a Windows desktop machine. The results of which are in in table 6.2, and show that the overall difference is negligible. For instance, HTTPS on the Mac machine had 415.375 seconds versus 416.375 on the Windows machine.

Finally the results in table 6.3 are from the experiments repeated again with sending a single request instead of 1111 requests in each burst iteration. The numbers for this table are in milliseconds. The average round trip time for a single HTTP and HTTPS request was 235.75 and 369.375 milliseconds respectively. For a single CoAP request without and with AES encryption it was 278.25 and 310.375 milliseconds respectively. Data consumption per request was 508, 509, 62, and 63 bytes for HTTP, HTTPS, CoAP, and CoAP with encryption respectively.

The results in table 6.3 highlight a difference between earlier results. While they concur that encryption increases round trip time and data consumption, there are two significant differences. Firstly the round trip time difference of CoAP versus HTTP is not as large as before, and secondly CoAP without encryption is shown to be slower than HTTP.

A deeper investigation reveals that the differences are due to differences between UDP and TCP, as well as the presence of a proxy server for CoAP. TCP is synchronous, however, UDP is asynchronous. Thus when sending multiple requests (e.g. 1111) UDP is faster. Since CoAP used UDP, it was faster than HTTP (e.g. 8 vs 235 seconds) when sending multiple requests even with the presence of a proxy server.

Furthermore, the CoAPToProxy column in table 6.3 shows the round trip time of a CoAP request/response sent from the local machine just to the proxy and back. The average time was 130.125 ms. This shows the affect of the proxy server and thus explains why CoAP took longer than HTTP within the event messaging system (235.75 vs 278.25 ms). However, CoAP with AES encryption took less time than HTTPS (310.375 vs 369.375 ms), due to the different encryption methods used by HTTP and CoAP.

Table 6.3: (Single Request) HTTP vs HTTPS vs CoAP vs CoAP AES. From Windows Local Machine (Results in mili-seconds).

Iteration	HTTP	HTTPS	CoAP	CoAP AES	CoAPToProxy
1	323	359	369	282	141
2	228	398	219	248	156
3	206	343	250	344	110
4	207	398	297	266	141
5	227	351	266	328	110
6	199	352	250	344	157
7	267	346	328	296	110
8	229	408	247	375	157
9	237	385	359	334	109
10	264	380	282	407	156
11	206	415	313	365	110
12	385	409	312	250	141
13	241	389	344	391	109
14	251	390	234	250	156
15	245	404	272	463	110
16	246	372	267	282	109
AVG	235.75	369.375	278.25	310.375	130.125

The Windows machine and the Mac machine used to simulate the client in the previous experiments have the following specifications respectively;

$$Windows10$$

$$Desktop$$

$$Processor : 3.6GhzAMDFX(tm) - 4100$$

$$Memory : 8GB1333MHzDDR3$$

$$StartupDisk : PatriotHD$$

$$OSXElCapitan$$

$$MacbookPro(Retina, 15-inch, Early2013)$$

$$Processor : 2.7GhzIntelCorei7$$

$$Memory : 16GB1600MHzDDR3$$

$$StartupDisk : MacintoshHD$$

Table 6.4 shows the results from the HTTP Load Bearing Test on the system's back-end server. Any crashes detected with each burst iteration are marked with "Failure", otherwise the burst iteration is marked with "Success". Each burst was repeated 11 times, with 4 different burst types increasing from 10 to 710 to 1410 to 2110 requests.

Table 6.4: Load Bearing Test with HTTP from Windows Local Machine.

Iteration	10 Burst	710 Burst	1410 Burst	2110 Burst
1	Success	Success	Success	Success
2	Success	Success	Success	Success
3	Success	Success	Success	Success
4	Success	Success	Success	Success
5	Success	Success	Success	Success
6	Success	Success	Success	Success
7	Success	Success	Success	Success
8	Success	Success	Success	Success
9	Success	Success	Success	Success
10	Success	Success	Success	Success
11	Success	Success	Success	Success
AVG	100	100	100	100

Table 6.5 shows the results obtained from the Intermittent Connectivity Test. It successfully shows that the "Intermittent connectivity" challenge has been overcome. Each iteration had a different period of connection loss, and yet in each iteration all 15 requests were successfully sent and received.

6.2 Summary Conclusions

Cloud Computing and RESTful web services have facilitated the design of a simple and reliable event messaging system, while overcoming challenges unique to the mobile space. This has been achieved due to a

Table 6.5: Intermittent Connectivity Test.

Iteration	Requests Sent	Requests Received	Result
1	15	15	Success
2	15	15	Success
3	15	15	Success

better understanding of the following;

- Mobile Computing.
- Event-messaging systems.
- Cloud Computing technologies.
- The REST architecture.

Experiments have shown the successful operation of an event messaging system for the mobile space. The following research sub goals have also been met;

- Overcome Challenges Found in the Mobile Space.
- Overcome Architectural Challenges.
- Compare and Contrast HTTP and CoAP for Use Within the System.

In regards to the research sub goal of comparing HTTP and CoAP. The following conclusions are derived from the experiments;

- The addition of a CoAP proxy server makes CoAP slower than HTTP.
- Encrypted CoAP is faster than HTTPS, even with a proxy server.
- The use of CoAP always reduces data consumption on the client side. However, total data consumption increases if server side data is included due to the addition of a proxy server.
- When sending multiple requests, CoAP is faster than HTTP regardless if encryption is used or if a proxy server is placed due to the properties of UDP.

This research has shown that CoAP may be used outside of an IoT environment, and may be used in the mobile space on mobile devices. However, for CoAP to function in the mobile space a proxy server is required to translate CoAP coming from devices to the existing HTTP back-end cloud server. This addition of a proxy server may make CoAP slower than HTTP depending if encryption is used or not, and the amount of requests being transmitted. CoAP does, however, always save data consumption on the client side. Thus in conclusion the benefits native to CoAP as described in the Literature Review only partially translate to the mobile space.

6.3 Research Contribution

The main research contribution of this thesis is the design and development of an event messaging system for the mobile development community to use. This event messaging system overcomes challenges unique to the mobile space. Servicing this system is a full architecture from the back-end server, to a website, to the API for mobile clients.

Furthermore an additional research contribution is the work done and demonstrated on the CoAP protocol and comparing it with HTTP for use on mobile devices. CoAP is usually found only on IoT devices for which it was originally designed for.

6.4 Future Work

The following chapter describes future planned and potential work. Each section includes an expansion of research, or further work enhancing the event messaging system.

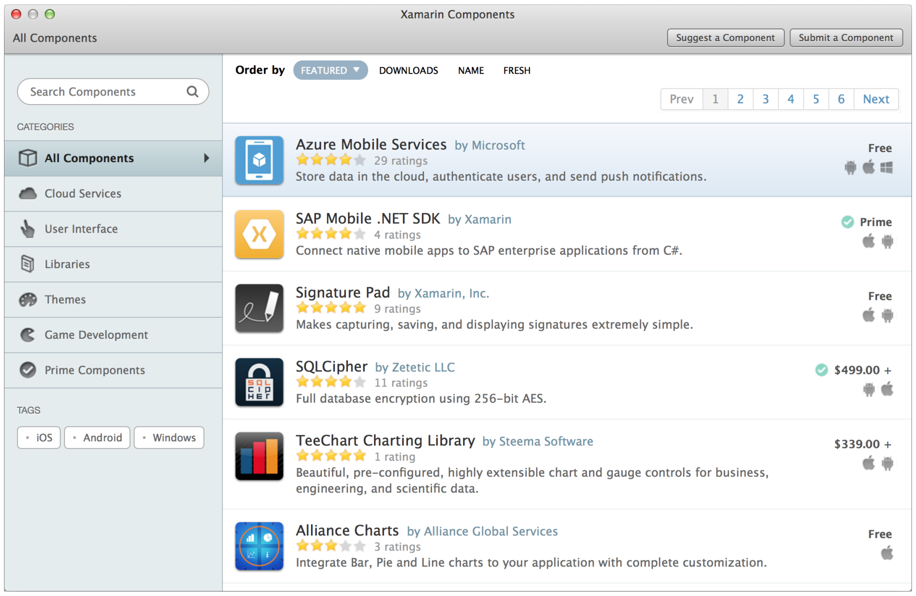
6.4.1 Expanding to the IoT

IoT devices could potentially use the event messaging system as well. This would greatly increase the scope of the system, however, would not involve significant additional work if a proxy server is already in place for IoT devices to connect with over CoAP. The potential benefit to IoT devices would be significant considering their limited computational availability. For instance, instead of a small device keeping track of the time of day a fridge is most often used, it could simply offload those calculations to the AAM system. Thus saving the device from consuming the little computation power it has.

Furthermore the system could facilitate communication between IoT and mobile devices. This would pave the way for a host of features such as being able to adjust the oven temperature from a smart-phone. Or adapt bedroom temperatures based on usage.

6.4.2 Open-Source Git-hub Plug In

Packaging a down loadable plug-in would be advantageous for the event messaging system. This would empower developers with what they need, to use the system immediately upon installation and more reliably. The package or plug-in design would include a local client database and a client side API already configured with a link to the back-end server.


Xamarin Components

All Components

Azure Mobile Services by Microsoft 29 ratings Store data in the cloud, authenticate users, and send push notifications.	Free
SAP Mobile .NET SDK by Xamarin 4 ratings Connect native mobile apps to SAP enterprise applications from C#.	Prime
Signature Pad by Xamarin, Inc. 9 ratings Makes capturing, saving, and displaying signatures extremely simple.	Free
SQLCipher by Zetetic LLC 11 ratings Full database encryption using 256-bit AES.	$499.00 +
TeeChart Charting Library by Steema Software 1 rating Beautiful, pre-configured, highly extensible chart and gauge controls for business, engineering, and scientific data.	$339.00 +
Alliance Charts by Alliance Global Services 3 ratings Integrate Bar, Pie and Line charts to your application with complete customization.	Free

Figure 6.1: The plug-in could be placed on the Xamarin component store for instance (the figure used is from Xamarin [22]).

Finally, a potential package or plug-in would also provide safeguards by facilitating certain checks and policies on the client side. These protective policies would review the data before being sent to the server; such as double-checking the data format and fields for corruption. However, having a package installed on the client also causes new problems that could potentially arise. Thus new considerations would need to be taken into account, such as the the mobile application's development environment.

When developing a mobile application, the software or tools used to do so constitutes the development environment. When creating a downloadable plug-in an issue of how to deal with the different development environments arises. For instance, a mobile application developed using PhoneGap, is completely different than one developed using Xamarin. The programming languages are different, as well as the technologies used. For instance, Phone Gap uses web technologies; thus a package for Phone Gap would simply be a JavaScript library with a HTML 5 Local Storage Database. However, Xamarin uses C Sharp; a plug-in for Xamarin would thus need to be written in C Sharp and include a SQLite Database.

6.4.3 Application Testing Feature

Another feature that could be provided is the ability to use the system to test a mobile application during development. This feature would allow developers to track how users navigate through their prototype, as well as provide them with the detail they need when a crash occurs.

For instance, if a prototype crashed an event detailing the crash would be sent out using the AAM system. The exact time the crash occurred, the screen the user was on, and the operating conditions is all information that would be included in the event. Furthermore beyond crash reporting, by seeing the user traces decisions could be better made on re-arranging screens and overall application navigation design.

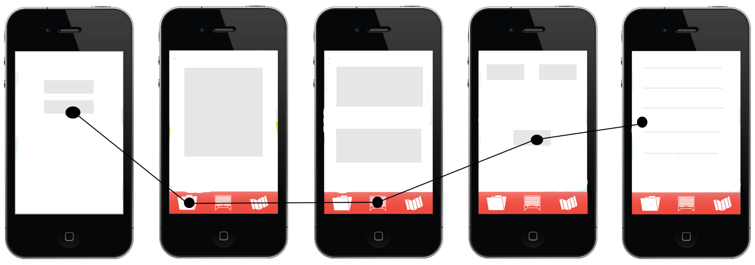


Figure 6.2: An example user trace through an application.

6.4.4 Further Protocols

Different protocols were researched and compared against each other, for use in the event messaging system described. This was done to compare the properties of each protocol and connect them with solving some of the mobile space and architectural challenges. This included the HTTP and CoAP protocols. Additional work includes expanding the research to other protocols such as;

- XMPP
- MQTT

References

[1] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[2] Guruduth Banavar, Tushar Chandra, Robert Strom, and Daniel Sturman. A case for message oriented middleware. In *Distributed Computing*, pages 1–17. Springer, 1999.

[3] Colin Barkerf. Forget 4g, networks struggle with 2g to keep london connected, says survey, 2015.

[4] Jon Brodkin. Bandwidth explosion: As internet use soars, can bottlenecks be averted?, 2015.

[5] Pew Research Center. Mobile technology fact sheet, 2015.

[6] Jason H. Christensen. Using restful web-services and cloud computing to create next generation mobile applications. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pages 627–634, New York, NY, USA, 2009. ACM.

[7] Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 2014-2019, 2015.

[8] Jim Edwards. The iphone 6 had better be amazing and cheap, because apple is losing the war to android, 2015.

[9] Manuel Fähndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen Hunt, James R Larus, and Steven Levi. Language support for fast and reliable message-based communication in singularity os. *ACM SIGOPS Operating Systems Review*, 40(4):177–190, 2006.

[10] Leah Findlater and Joanna McGrenere. A comparison of static, adaptive, and adaptable menus. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 89–96. ACM, 2004.

[11] I. Foster, Yong Zhao, I. Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, Nov 2008.

[12] Apache Software Foundation. Apache jmeter, 1999.

[13] Martin Fowler. Richardson maturity model: steps toward the glory of rest. *Online at http://martinfowler.com/articles/richardsonMaturityModel.html*, 2010.

[14] Seth Gilbert and Nancy Lynch. Brewers conjecture and the feasibility of consistent. *Available, Partition-Tolerant Web Services*, 2002.

[15] Google. Google analytics, 2014.

[16] Google. Google app engine: Platform as a service, 2014.

[17] Google. Google now, 2015.

[18] Google. Using the new tab page, 2015.

[19] Michael Hatamoto. Idc: Windows will gobble android and ios market share; smartphone prices will drop, 2015.

[20] Annika Hinze, Kai Sachs, and Alejandro Buchmann. Event-based applications and enabling technologies. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 1:1–1:15, New York, NY, USA, 2009. ACM.

[21] Apple Inc. Start something new., 2015.

[22] Xamarin Inc. Xamarin studio the best ide for cross-platform mobile development, 2013.

[23] Do-Guen Jung, Kwang-Jin Paek, and Tai-Yun Kim. Design of mobile mom: Message oriented middleware service for mobile computing. In *Parallel Processing, 1999. Proceedings. 1999 International Workshops on*, pages 434–439. IEEE, 1999.

[24] Theo Kanter. Event-driven, personalizable, mobile interactive spaces. In *Handheld and Ubiquitous Computing*, pages 1–11. Springer, 2000.

[25] Kissmetrics. Kissmetrics, 2013.

[26] Sean Lindo. Xml vs. json - a primer, 2013.

[27] One Click Technologies Pvt. Ltd. Computing devices, 2015.

[28] Ingrid Lunden. Gartner: 195m tablets sold in 2013, andriod grabs top spot from ipad with 62

[29] Microsoft. The start menu (overview), 2015.

[30] Nicolas Niclausse. Tsung is an open-source multi-protocol distributed load testing tool, 2013.

[31] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of json and xml data interchange formats: A case study. *Caine*, 9:157–162, 2009.

[32] Pragmateek. Json vs xml: Some hard number about verbosity, 2013.

[33] Carlos Rodrigues, José Afonso, and Paulo Tomé. Mobile application webservice performance analysis: Restful services with json and xml. In *ENTERprise information systems*, pages 162–169. Springer, 2011.

[34] Eric Rodriguez. Tcp vs udp, 2015.

[35] Peter Saint-Andre. Extensible messaging and presence protocol (xmpp): Core. 2011.

[36] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (coap). 2014.

[37] SQLite. About sqlite, 2014.

[38] Andy Stanford-Clark and Hong Linh Truong. Mqtt for sensor networks (mqtt-sn) protocol specification, 2013.

[39] Adobe Systems. Easily create apps using the web technologies you know and love: Html, css, and javascript, 2014.

[40] Daniel S Weld, Corin Anderson, Pedro Domingos, Oren Etzioni, Krzysztof Gajos, Tessa Lau, and Steve Wolfman. Automatically personalizing user interfaces. In *IJCAI*, volume 3, pages 1613–1619, 2003.

[41] XMPP. Xmpp standards foundation., 2015.