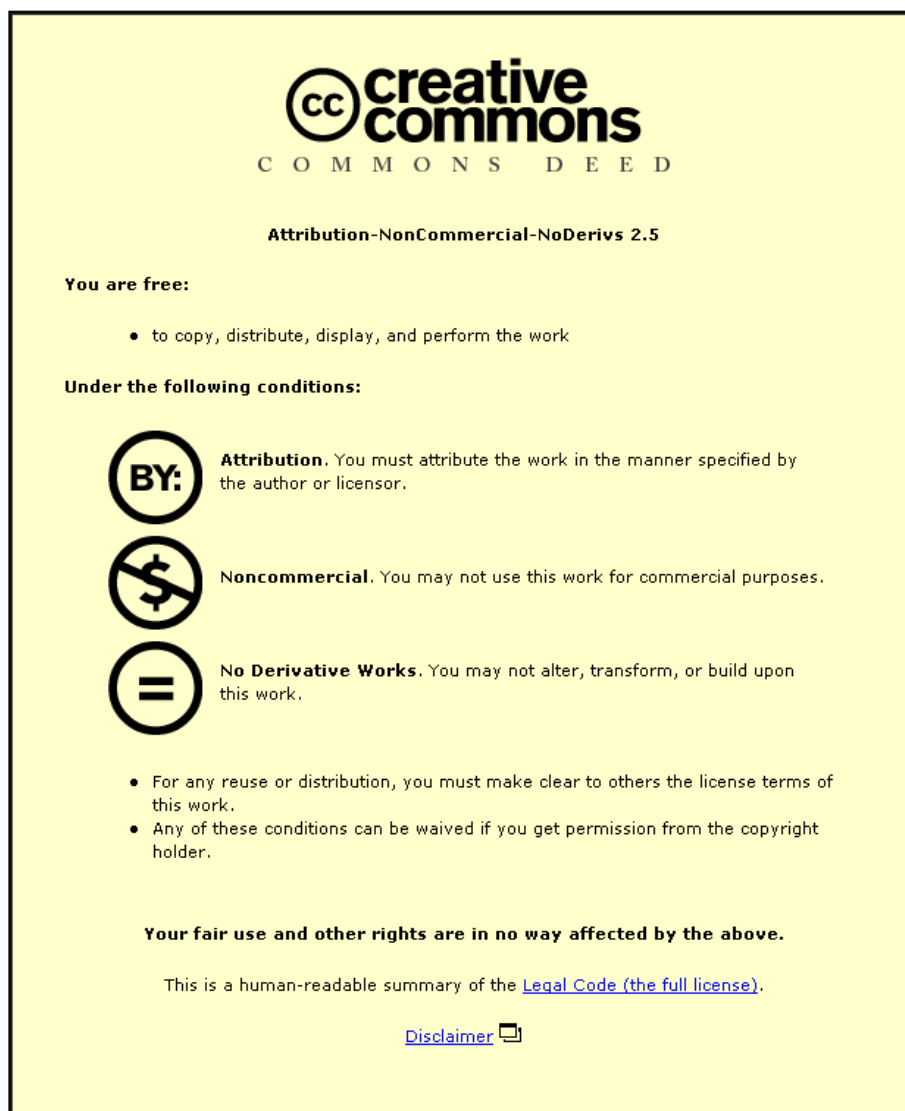


This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (<https://dspace.lboro.ac.uk/>) under the following Creative Commons Licence conditions.



For the full text of this licence, please go to:  
<http://creativecommons.org/licenses/by-nc-nd/2.5/>

A Novel Platform Incorporating  
Multiple Forms of Communication  
to Support Applications in a Mobile  
Environment

by  
James Elton

A Doctoral Thesis  
Submitted in partial fulfillment of the requirements  
for the award of  
Doctor of Philosophy of Loughborough University  
February 2014

© by James Elton, 2014

## Abstract

This thesis discusses the creation of a novel platform that incorporates multiple communication methods, including SMS, email and web-based technologies, for interacting with users of mobile communication devices. The platform utilises people in a mobile environment to solve a range of different application problems, where each problem is a separate and distinct scenario type with unique objectives.

There are existing applications available that interact with users of mobile communication devices to provide a service, such as regular weather updates to the users. Other applications have been designed to manage and coordinate the users to perform tasks within a mobile environment, such as performing field studies for scientific purposes. However, the existing applications are designed for only one specific scenario, with the design and implementation solely focused on solving the objectives of that scenario. Each component of these applications needs to be developed from scratch in order to cater for the application's requirements.

There is currently no integrated communications platform that offers a framework for supporting a range of different scenario types. The new platform, entitled the *Connected-Mobile Platform*, aims to support the rapid development and implementation of new scenarios. This platform is composed of a framework of generic components that enable the active running of multiple scenarios concurrently, with the ability to tailor to the requirements of new scenarios as they arise via a structured process. The platform facilitates a means to coordinate its users in order to tackle the objectives of a scenario.

The thesis investigates several system architectures to determine an appropriate architectural design for constructing the proposed platform. The platform has a generic framework, based on a client-server architecture, to facilitate the inclusion of a multitude of scenarios. A scenario represents a problem or an event, whereby the platform can utilise and interact with users of mobile communication devices to attempt to solve the objectives of the scenario. Three mobile communication methods are supported; the Short Message Service, electronic mail and web-forms via the mobile internet. Users are able to select and switch between the different methods. The thesis describes the platform's tailored communication structure for scenarios and autonomous analysis of messages.

The thesis discusses case studies of two different scenarios to evaluate the platform's facilities for rapid scenario development. The *Diet Diary* scenario, which is for individual users, aims to manage a user's daily calorie intake to help them reach their desired weight goal. The focus is on the platform's functionality for analysing and responding to messages autonomously. The *Missing Persons* scenario, which utilises multiple users, involves tracking and locating people who have been reported missing. The focus is on the platform's functionality for coordinating the multiple users, through the creation of assignments, in order to distribute the scenario objectives. The thesis concludes by highlighting the novel features of the platform and identifying opportunities for future work.

**Keywords:** generic framework; mobile communication; mobile society; client-server architecture; data analysis; email; short message service.



## **Acknowledgements**

Firstly, I would like to thank my supervisor, Professor Paul Chung, for the supervision and guidance he has provided throughout my PhD research. He has motivated me to stay focused at each stage of my thesis, been very supportive of my work and provided me with valuable research skills.

I would like to thank my second supervisor, Dr Andrew May, for reading through my thesis and the feedback he has provided on my work.

I would also like to thank Dr Lin Guan and Professor David Parish for their help and advice in the early years of my research.

I would like to express my gratitude to the staff of the Computer Science department for their help and assistance over my time researching and working at Loughborough University, especially Christine Bagley, Pamela Taylor, Jo McOuat, Judith Poulton and Gurbinder Singh Samra. I would also like to thank the research students of the Computer Science department for their support and providing a friendly environment in the office.

I am grateful to Loughborough University for funding this work.

I would like to give a special thanks to my family. My parents have been very encouraging, especially through the tough times of my work. I would not have arrived at this stage without their tremendous love and support.

## **Abbreviations**

ADO	:	ActiveX Data Objects
ASP	:	Active Server Pages
BMI	:	Body Mass Index
CPU	:	Central Processing Unit
DER	:	Distributed Energy Resource
Email	:	Electronic Mail
GB	:	Gigabyte
GHz	:	Gigahertz
GPRS	:	General Packet Radio Service
GPS	:	Global Positioning Satellite
GSM	:	Global System for Mobile Communication
GUI	:	Graphical User Interface
HTML	:	HyperText Markup Language
HTTP	:	Hypertext Transfer Protocol
IDAPS	:	Intelligent Distributed Autonomous Power System
IDE	:	Integrated Development Environment
JPEG	:	Joint Photographic Experts Group
LAN	:	Local Area Network
MET	:	Metabolic Equivalent of Task
MMS	:	Multimedia Messaging Service
MPSS	:	Mobile Pharmacy Service System
P2P	:	Peer-to-Peer

RDA	:	Recommended Daily Allowance
SETI	:	Search for Extra-Terrestrial Intelligence
SMS	:	Short Message Service
SMSC	:	Short Message Service Centre
SMTP	:	Simple Mail Transfer Protocol
SNS	:	Social Networking Site
SOA	:	Service-Oriented Architecture
SOAP	:	Simple Object Access Protocol
SQL	:	Structured Query Language
RAM	:	Random Access Memory
RDBMS	:	Relational Database Management System
TAU	:	Template Analysis Unit
UDDI	:	Universal Description Discovery and Integration
URI	:	Uniform Resource Identifier
WMRS	:	Web-based Manufacturing Resource Service
WSDL	:	Web Services Description Language

## Table of Contents

1	Introduction .....	1
1.1	Background .....	1
1.2	Aim and objectives .....	3
1.3	Project overview .....	5
1.4	Research methodology .....	7
1.5	Structure of the thesis .....	9
2	Communication Methods and Devices Operating in a Mobile Society .....	10
2.1	Introduction.....	10
2.2	Communication methods .....	10
2.2.1	Short Message Service .....	10
2.2.2	Multimedia Messaging Service .....	14
2.2.3	Electronic mail.....	15
2.2.4	SMS web services.....	17
2.2.5	Summary and conclusions .....	19
2.3	Devices and technology .....	20
2.3.1	Mobile phones .....	20
2.3.2	Smartphones .....	22
2.3.3	Tablet computers .....	25
2.3.4	Integration of a digital camera.....	26
2.3.5	Location-based services.....	27
2.3.6	Summary and conclusions .....	30
2.4	Conclusions.....	30
3	Review of Communication-Based Applications that Interact with Multiple.....	
	Users.....	32
3.1	Introduction.....	32
3.2	Applications utilising a mobile society .....	33
3.2.1	<i>Pharmaceutical Care</i> application.....	33
3.2.2	<i>Attendance Improvement</i> applications.....	34
3.2.3	<i>Interactive Learning</i> application .....	35
3.2.4	<i>Smoking Cessation</i> application.....	38
3.2.5	<i>Telemedicine Monitor</i> application.....	39
3.2.6	<i>Road Safety Alert</i> application .....	42

3.2.7	<i>UbiquitousSurvey</i> system .....	43
3.3	Non-mobile applications .....	45
3.3.1	<i>HELP</i> system .....	45
3.3.2	<i>Remote Experimentation</i> system .....	46
3.3.3	<i>Perfect Diet Tracker</i> application .....	48
3.4	Social networking sites .....	51
3.5	Conclusions.....	56
3.5.1	Communication methods and devices .....	56
3.5.2	Application features.....	58
3.5.3	Necessity to combine features into a generic platform .....	61
4	System Architectures .....	63
4.1	Introduction.....	63
4.2	Client-server architecture.....	63
4.3	Peer-to-peer architecture .....	68
4.4	Multi-agent architecture.....	72
4.5	Service-oriented architecture .....	75
4.6	Conclusions.....	82
5	Design and Implementation .....	85
5.1	Introduction.....	85
5.2	Design of the <i>Connected-Mobile Platform</i> .....	86
5.2.1	The platform's client-server architecture .....	86
5.2.2	Integrated architecture to support multiple communication methods ...	88
5.2.3	Generic framework to manage multiple scenarios .....	91
5.2.4	Implementation of templates for autonomous operations .....	93
5.2.5	User preferences .....	95
5.2.6	Assignments for collective intelligence.....	97
5.2.7	Roles of the operator and scenario builder .....	99
5.2.8	Message entity .....	101
5.2.9	News entity .....	102
5.2.10	Data layer of the platform.....	103
5.3	Data analysis process .....	105
5.3.1	<i>MailHandler</i> class .....	106
5.3.2	<i>RequestHandler</i> class .....	110

5.3.3	<i>TemplateHandler</i> class .....	112
5.3.4	<i>AnswerHandler</i> and <i>TriggerHandler</i> classes .....	113
5.3.5	<i>DatabaseHandler</i> class.....	115
5.3.6	<i>ActionHandler</i> class.....	117
5.3.7	<i>DataOutHandler</i> class .....	121
5.4	Developing a scenario application.....	122
5.4.1	Stage 1: Creating the scenario's database .....	122
5.4.2	Stage 2: Template and analysis criteria .....	123
5.4.3	Stage 3: Application module development.....	124
5.5	Development environment and software tools .....	125
5.6	Experimental setup for operating scenarios in the case studies.....	127
5.7	Conclusions.....	131
6	Case Study 1: <i>Diet Diary</i> Scenario .....	134
6.1	Introduction.....	134
6.2	Details of the scenario.....	135
6.3	Implementation of the scenario.....	137
6.3.1	Database entities .....	137
6.3.2	Template and analysis criteria .....	139
6.3.3	Application module development.....	147
6.4	Development time comparison for the <i>Diet Diary</i> scenario .....	149
6.5	Running a <i>Diet Diary</i> instance.....	152
6.6	Conclusions.....	171
7	Case Study 2: <i>Missing Persons</i> Scenario .....	174
7.1	Introduction.....	174
7.2	Details of the scenario.....	175
7.3	Implementation of the scenario.....	177
7.3.1	Database entities .....	177
7.3.2	Template and analysis criteria .....	179
7.3.3	Development of new modules .....	185
7.4	Development time comparison for the <i>Missing Persons</i> scenario.....	186
7.5	Running a <i>Missing Person</i> instance.....	192
7.6	Scalability test to assess the performance of the platform.....	207
7.7	Conclusions.....	218

8	Conclusions and Future Work.....	221
8.1	Conclusions.....	221
8.1.1	Support of multiple communication methods .....	222
8.1.2	Features of the novel framework .....	224
8.1.3	Autonomous functionality for real-time responses .....	225
8.1.4	Assignment functionality for coordination of users .....	227
8.1.5	Rapid scenario development process.....	228
8.1.6	Summary of contributions .....	230
8.2	Future work.....	233
8.2.1	Improvements to the platform's functionality.....	233
8.2.2	Ideas for future areas of research.....	238
	References .....	245
	Appendices .....	264
A1	Algorithms for the <i>Diet Diary</i> Scenario .....	264
A2	Published Work.....	267

# **1 Introduction**

## **1.1 Background**

People in today's society commonly use mobile communication devices to perform a variety of tasks, which includes communicating with other people, interacting with an array of services and accessing the internet. The users of mobile communication devices are free to work on the tasks while they are on the move [1]. The rise in popularity of mobile phones has enabled people in society to communicate with each other from almost any location and at any time [2]. The user may either be at a fixed location such as their home or on the move, whilst in the process of the interaction. Therefore, the ability for a user to interact with an application via a mobile communication device from any location is referred to, in this thesis, as interacting in a mobile environment. There has been a rapid growth in one particular type of device, the mobile phone, with subscriptions worldwide growing by approximately 45% per year since 2006 [3]. Mobile phones have become a must-have technology that a large number of people would always have close to them [4].

Mobile phones support multiple methods of communication in order for the user to interact with other device users. These methods of communication range from the standard voice communication to the Short Message Service (SMS) protocol, which is used for the communication of short text messages between mobile phones [5]. SMS has become a popular communication method for mobile phone users [6] with the majority of mobile phones adopting SMS as the default method for sending and receiving messages between devices [7, 8]. The introduction and advancement of mobile internet technology has enabled a large number of mobile phones to now include integrated electronic mail (email) functionality [9]. Therefore, these devices can take advantage of the advanced features available via email communication. For example, the images that are captured on camera-integrated devices can be sent immediately, as an attachment, in an email [10].

Originally, the purpose of SMS was for person-person messaging, whereby two subscribers could exchange messages between each other [6]. However, it has become an increasingly common practice for organisations to communicate with numerous mobile phone users in order to supply information and gather data from the



users in a process known as machine-person messaging. This process has been used for information services, whereby a mobile phone user may subscribe to the service in order to receive regular message updates on topics of interest. The messages are automatically created by these information services, which are then sent to the subscriber's device. An example information service is an SMS weather service that sends users daily updates on the weather in the user's current location [11].

People are now able to communicate with others and utilise a vast range of information services from almost any geographical location, leading to the society we are living in becoming increasingly more mobile. This new type of mobile society that has developed in recent years has enabled increasingly complex services and applications to be developed, which provide a more interactive scope for mobile phone users. The users of these applications are able to interact via mobile communication methods. For example, a *Pharmaceutical Care* application [12] has been developed and implemented in a hospital in China to provide individual care to outpatients by the use of SMS communication. The application focuses on tailored SMS communication, generating and sending individualised text messages to outpatients that provide alerts specifically related to the attributes of these users. The application would also receive messages from a user in order to be kept up-to-date with their condition. The *UbiquitousSurvey* system [13] facilitates field studies to observe population characteristics of animals. The application focuses on managing users as data gatherers, whereby their mobile communication devices are used to collect the data from the field. This data is then sent to a central server for analysis using a data communication method over the mobile internet.

These bespoke systems can take a considerable amount of effort and time to create, whereby each component has to be developed and tested from scratch. This may include designing a component for sending and receiving messages and another component for the analysis of received messages. The design elements, within these components, could overlap considerably between different systems. However, the framework for each system is only created for a specific focal problem. To cater for a new problem would require large changes in the system design. To create a communication system that is generic to support multiple scenarios would reduce the

development effort for the individual scenarios, making the system far more robust in an ever changing and dynamic world.

## **1.2 Aim and objectives**

The aim of this project is to design and create a new platform, with multiple sources of input and output, to support the development and implementation of scenario applications that would run in a mobile environment. Existing applications that interact with users in a mobile environment are only designed for a specific focal problem, whereby each component of an application must be developed from scratch to resolve that problem. The proposed platform would consist of a novel framework of generic components to enable new types of scenario applications to be rapidly integrated for active use via a structured process. This would include support for scenario applications that provide a direct service to individual users of mobile communication devices via tailored interactions. The platform would also support scenario applications that focus on managing and coordinating multiple users to solve tasks in a dynamic and changing environment.

The novel platform would primarily act as the communications handler for each scenario application. The key functions of the platform would be to receive communicated messages, extract and analyse the message data, store the data in meaningful locations and perform appropriate actions in response to the data. These functions would be dependent on the scenario application that the communicated data is regarding. The entire interaction process, from the platform initially receiving new information to performing appropriate actions in response, would need to be completed within a short timeframe with the appropriate threshold determined by the scenario application. This is to ensure that the platform can react to time-relevant data swiftly, allowing users to receive updates or further instructions without a long delay. This requirement is referred to as the platform's ability to process message data in real-time.

The platform would interact with people in our mobile society, who are the users of mobile communication devices. The platform would utilise these people to negotiate through and solve the objectives of each scenario application. The participating users could be requested to undertake application-specific tasks or gather field data to assist in the running of the application. Through the research of current communication

techniques this project should consider the means to effectively stay in contact with the users of the platform at all times, regardless of their geographical location. This research explores the methods that would ensure as many potential users of mobile communication devices as possible can interact with the platform. The platform should then be able to coordinate multiple users in order to solve objectives that arise in the running of a scenario application. Multiple methods of communication should be available within the platform to cover the wide range of devices, as well as benefit from the different features and properties that each method offers. This would provide a means to establish an effective communications structure to maintain a dialogue with each user in order that they can participate in an application.

The platform requires a novel framework design to support the development and integration of a range of different applications. This should be based on a suitable system architecture to ensure an efficient method is formulated for handling system data and communicating with users of mobile communication devices. Each application would represent a problem scenario with specific requirements, which would require fulfilling by the platform. The platform's framework would have generic properties to provide the necessary functionality to run each application.

A proof of concept prototype of the platform is to be developed, based on the selected system architecture. A survey of current applications that utilise people in a mobile society is needed to be able to identify functionality for implementation in the platform. The platform needs to include a straightforward development process for integrating each scenario application in order to facilitate an application's rapid implementation. Each of these factors should help contribute to a large reduction in the time and effort required to build each application, as the underlying components of the platform would be re-usable for each application. Therefore, a new application would not need to be built from scratch each time.

Case studies are to be performed for evaluating the feasibility and effectiveness of the platform by running example scenario applications that test the different aspects and functionality of the platform. The platform's development process would need to be compared to the implementation of each application from scratch in order to demonstrate the amount of time and effort saved by using the platform. Once the scenario application is ready for use within the platform, an experiment is required to

assess the real-time running of the application from the users' points of view. Furthermore, a scalability test is required to assess the performance of the platform when a large quantity of users are registered and interacting in the relevant scenario application. This would highlight the limitations of the platform by discovering the maximum amount of users that can interact simultaneously with the platform, without sacrificing the platform's capability to respond in real-time. This evaluation would help discover areas of improvement that can be made to enhance the platform.

### **1.3 Project overview**

An integrated communications platform, entitled the *Connected-Mobile Platform*, has been developed to support three different types of mobile communication methods; SMS, email and data input via a web-based user interface. Each communication method has unique properties that the platform is able to utilise. SMS is ubiquitous for the sending and receiving of short text messages between mobile phones. Other types of internet-enabled communication devices can use email and the web-based user interface to interact with the platform. These internet-based methods offer additional functionality, including the transmission of multimedia messages and a more user-friendly interface respectively. By incorporating support of the three communication methods, the platform is able to interact with a broad range of mobile communication devices. The users choose their preferred method of communication for the sending and receiving of messages, which the server would then utilise to interact with them. This preference is automatically updated whenever the user switches to an alternative communication method, enabling the user to select the most effective and convenient communication method each time they initiate the interaction process.

The platform has a framework of generic components to facilitate the inclusion of a range of different scenario applications, whereby each scenario represents a specific type of problem. Each occurrence of a scenario is defined as an instance of the scenario. The framework is designed based on a client-server architecture where operations are split between two types of components; a server and client devices [14]. The server is a computing device that houses the core application of the platform and the databases for data storage, containing the necessary hardware and software to run the platform. Each user in the field is a client with their mobile

communication device being a client device. The users perform tasks and gather field data, which is then sent to and analysed by the server. The platform design is based on three separate layers; a presentation, logic and data layer. These layers have been designed using the foundation of a tiered communication model, which has been implemented within the *Remote Experimentation* system [15]. This provides a structure for the communication between the server and its clients.

The core functionality of the platform covers a wide range of features that can be utilised by each scenario, which includes the underlying communication process for interacting with the users. A template mechanism has been implemented, whereby each message exchanged between the server and its users is based on a pre-defined template structure. The template mechanism supports all three communication methods, enabling a simplified process of data extraction by the server and an effective means for users to send field data.

The platform's middleware, entitled the *Template Analysis Unit* (TAU), handles the analysis of field data extracted from a user's message. The TAU utilises both template criteria and analysis entities that have been pre-defined and tailored to the requirements of the scenario. The purpose of the TAU is to autonomously process user messages. Therefore, actions and responses can be achieved in real-time, based on the results of analysis on the field data sent from a user. Additionally, each scenario can have its own application module integrated within the platform for scenario specific functionality and analysis.

There is a *Principal* database to store data on the core entities of the platform. These include scenarios, users, messages, templates and the criteria for data analysis procedures. Additionally, each scenario has its own specific database to store collected field data relating to the scenario.

The platform incorporates assignment functionality and location-based technology to manage and coordinate users in multi-user scenarios. New assignments are dynamically generated during the active running of an instance, whereby each assignment is created for a user to resolve a specific objective of the instance. An assignment that is location-dependent would be efficiently allocated to a user based on the closest user's geographical proximity to the assignment's location.

The platform enables a smooth and rapid integration of scenario applications, which is based on a three-stage scenario development process. It is the role of the scenario builder to develop each scenario. This process involves the design of the scenario's database, creation of the template and analysis criteria and the programming of the application module.

There are two different scenarios that are designed to test and evaluate the platform's functionality. The first scenario, entitled the *Diet Diary* scenario, tracks the daily food consumption and activities performed by a subscribed user. This is a basic scenario as only one user is assigned to each instance. Its purpose is to evaluate the communication process between the server and a user, as well as the TAU's ability to process field data. The second scenario, entitled the *Missing Persons* scenario, investigates the platform's ability to handle and coordinate multiple users. The aim of this scenario is to track and locate people who have been reported missing. Assignments can be generated for this scenario to provide specific tasks to each user, making it possible for the platform to coordinate the tasks being performed by the multiple users. During the investigation of these scenarios the development time and effort is analysed to determine the platform's ability to support a rapid implementation of future scenarios.

## **1.4 Research methodology**

This project has been carried out based on a five-stage research methodology. The initial stage of the project involves undertaking a requirements analysis to build a functional specification of the features to be incorporated in the proposed platform. The requirements analysis starts with a review of current mobile communication technology. This review is performed to identify both the communication methods and hardware (mobile communication devices and their integrated components) that the platform would need to support.

The requirements analysis also consists of a review of existing applications that interact with their users via the researched mobile communication methods. This would include an investigation into applications that provide a tailored one-to-one service to each of its users and applications that interact with multiple users to resolve a specific problem. This review is performed to identify the procedures used to interact with and manage the application users within a mobile environment as well

as to explore the practical features that have been implemented to resolve the objectives of the applications. Furthermore, it is important to determine missing functionality that would provide improvements to the operations of these applications. Conclusions from this review would determine the necessary functionality to include in the platform.

The next stage is to select a suitable systems architecture for the design of the platform. It is important to investigate multiple system architectures, examining applications that use each architecture to assess their benefits and drawbacks for use in a mobile environment.

The gathered requirements within the functional specification need to be translated into a design specification for the platform. The platform would be split into several components, each tasked with supporting one or more of the required features initially outlined in the functional specification. Using the design specification a prototype of the platform can be developed and implemented within a programming environment. This involves selecting a suitable programming language and data storage solution to ensure that the platform is able to operate and support the identified feature-set.

Once the platform has been implemented it is important to observe its functionality in supporting active scenarios. This is achieved by performing experimental case studies to test the different types of scenarios that the platform is required to support. This stage involves running the structured development process for each scenario type, where the speed and ease of integrating new scenarios into the platform is assessed. The experiments also include testing the platform's functionality in running instances of each scenario and supporting multiple users simultaneously.

The final stage of the research methodology is to evaluate the platform's feasibility in achieving the aim and objectives that were initially identified and defined, using the resultant data from the two case studies. The conclusions are used to determine the contributions of the thesis to knowledge, as well as to identify areas of improvement to advance the scope of the platform and its wider applicability to resolve practical problems.

## 1.5 Structure of the thesis

Chapter 2 reviews the different technologies that are currently available for facilitating communication in a mobile society. This includes the investigation of communication methods, such as SMS and email, which transmit information between two different devices. The devices that provide a means for people to communicate with each other, such as mobile phones, are also investigated.

Chapter 3 investigates applications that have utilised current mobile communication technologies to achieve a particular goal and explores the successes and shortcomings of these applications. Features of each application are identified to establish the functionality that should be implemented into the proposed platform, which aims to support a range of different applications. Other applications are also investigated that include functionality which may be advantageous to the platform.

Chapter 4 investigates three different system architectures to determine the most appropriate type for the proposed platform. The investigated types are the client-server, peer-to-peer and multi-agent architectures. Current systems that utilise each architecture are examined to identify their benefits and drawbacks.

Chapter 5 discusses the novel design and implementation elements of the new platform, detailing how the platform's generic framework is created together with the design elements for integrating scenario applications. The stages of the data analysis process performed on user messages are described in this chapter. This is followed by the details of the scenario development process required for each scenario to be integrated into the platform.

Chapters 6 and 7 examine case studies of different scenarios used to evaluate the platform's functionality. These scenarios are the *Diet Diary* scenario (Chapter 6) and the *Missing Persons* scenario (Chapter 7). The design elements of each scenario and the way in which these scenarios utilise functionality from the platform's generic framework are discussed. The scenario development process is examined for these scenarios, focusing on the implementation of functionality specific to each scenario.

Chapter 8 presents a summary of the project and highlights the ways that the aim and objectives of the project have been met. This chapter also discusses the contributions of the project and identifies future work that is required.



## **2 Communication Methods and Devices Operating in a Mobile Society**

### **2.1 Introduction**

This chapter investigates the current techniques that are employed for mobile communication. The review surveys the communication methods and devices that are currently available, exploring their use in a mobile environment. These methods and devices are explored to view how they can assist services that interact with people in a mobile society, where these people need to be contacted from various locations. In recent years the popularity and continuous development of mobile phones has provided flexible means of communication in a mobile society. This development has been both on the hardware side of mobile phones and the available methods of communication.

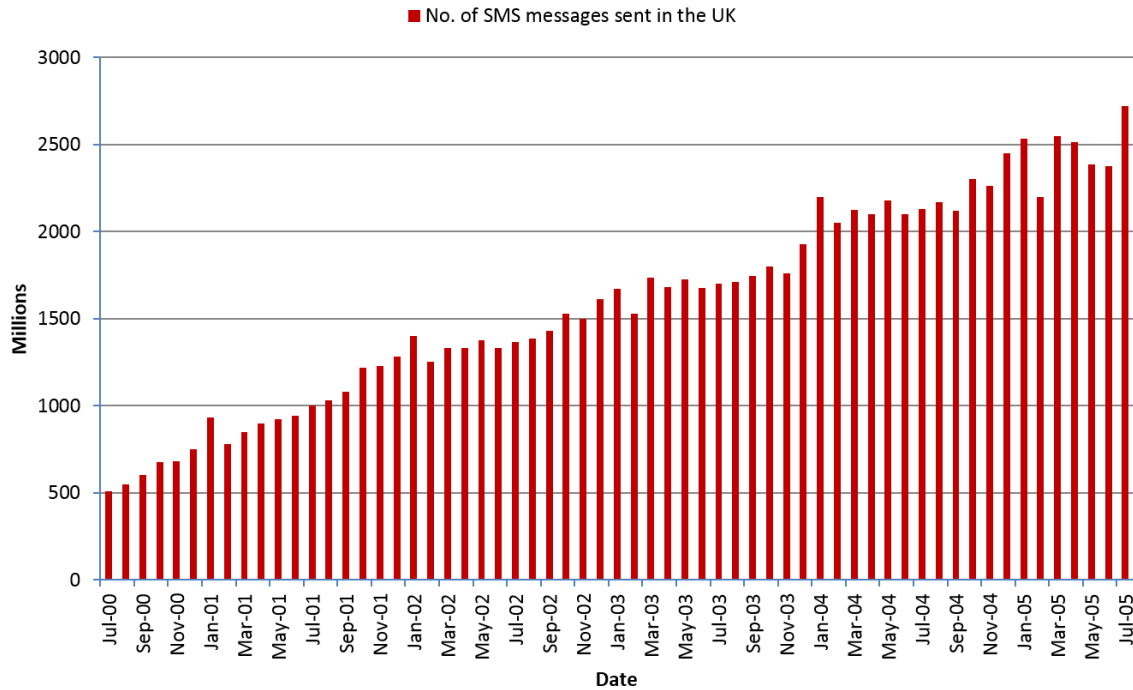
### **2.2 Communication methods**

This section focuses on communication methods that can be utilised for communicating in a mobile environment. When mobile communication devices first came to the market, voice communication was the only available method. Over the last few decades a range of new communication methods have been developed [2]. These new methods have allowed for a variety of different ways to communicate in a mobile society, which started with simple text-based communication. This has progressed to more advanced methods, which allow for communicating with multimedia and more flexible methods to transmit data, such as the internet.

#### **2.2.1 Short Message Service**

Short Message Service (SMS) is a mobile communication method, whereby mobile phone users can communicate through the sending and receiving of short text messages [5]. There has been a rapid growth in this communication method since the first text message was sent in the early 1990s [6, 16]. Figure 2.1 illustrates the early growth of SMS within the UK, showing the rise in the number of text messages sent each month between the years of 2000 and 2005. Figure 2.2 illustrates the continued growth worldwide to the year 2010. These charts show that there has been a steady and consistent increase in messages sent, which were still increasing by over 40% in 2010. There have been two key reasons for this growth. The majority of mobile

phones are compatible with SMS and provide straightforward ways of using this method to send and receive messages. Secondly, sending a text message is a less expensive option, provided by the mobile networks, to that of a voice call [7, 5]. These attributes have helped to make sending SMS messages a ubiquitous method of communication [8].

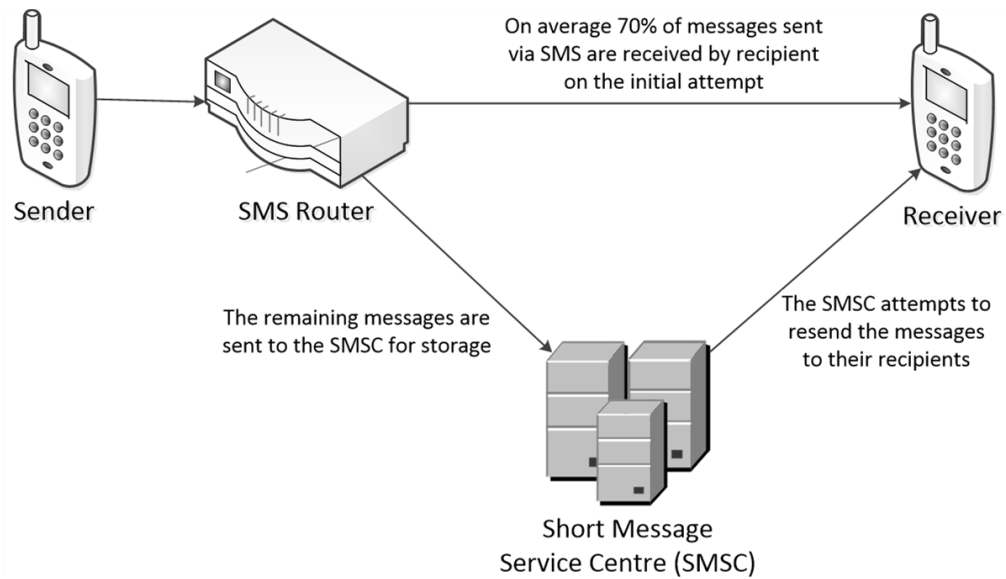


**Figure 2.1:** Initial growth of SMS in the UK from 2000-2005 (adapted from [17])



**Figure 2.2:** Continued growth of SMS messages sent worldwide from 2007-2010  
(adapted from [18])

Originally, the purpose of SMS was for person-person messaging, whereby two human subscribers could exchange messages between each other [6]. Messages are sent over the mobile network infrastructure and received by the receiver when their device is ready to accept incoming data. If the receiver's device is switched on and can obtain a network signal then a message is usually received within seconds. However, the store-and-forward specification of SMS means it is not necessary for the receiver to have their device on when the message is sent [19]. If the receiver's device cannot obtain a network signal then the message would be stored in an SMS centre, an intermediary location held by the network, until the message is successfully delivered, as illustrated in Figure 2.3. SMS does not require users to be available in real-time, as opposed to voice calls. The receiver can then choose to respond to the sender at a convenient time [20]. This factor has helped make SMS a popular method of communication for the younger generation [2, 21]. It is an accessible way for students and young adults to keep in touch with friends [22]. Each user can regularly send messages throughout the day, providing a continuous channel of communication between the two parties without the need to send a reply immediately [23]. Furthermore, SMS messages can be sent from any location where the sending device can obtain a network signal, which provides the ability to communicate anywhere and at any time.



**Figure 2.3:** *Store-and-forward procedure for sending messages via SMS (adapted from [19])*

A further use for SMS is for machine-person messaging, whereby communication occurs between a person and a computer in order to achieve a particular objective [6]. An example of the usage of machine-person messaging is through information services. These are services which mobile phone users may subscribe to for the purpose of receiving message updates on topics of interest (Figure 2.4). Such examples include weather, traffic and news information services [11]. Over two million mobile phone users utilise SMS weather services on a daily basis in the USA in order to obtain details concerning the weather [24]. Messages are automatically created by these information service providers, which are then sent to the subscriber's device. The reverse process of person-machine messaging also exists. An example of this is where television viewers can participate by sending messages on some reality television shows. This includes sharing their views on the subject being discussed in a talk show or providing a vote for contestants in a competition [22]. These are examples of either utilising people in society or providing a service to the users, regardless of their physical location.



***Figure 2.4:** News (left) and weather (right) notifications sent from an information service provider to subscriber devices (retrieved from [25])*

### **2.2.2 Multimedia Messaging Service**

Multimedia Messaging Service (MMS) enables complex information to be sent over mobile networks. This is an enhancement over SMS and is conventionally utilised to send multimedia files as an attachment alongside a text message [26, 6]. SMS limits the user to only a small quantity of text per message. MMS ranges from basic text messages, where the text can be formatted in a variety of ways, to the inclusion of image, video and audio files (Figure 2.5). These extra features provide more interaction and control over the content that is included in the message [27]. MMS technology offers the ability to send an image immediately, which has just been captured, to another device. For example, whilst shopping it is possible to take a picture of an item and send the image file to a friend to find out their opinion on whether it would be a worthwhile purchase [28].



***Figure 2.5:** An MMS message sent to a mobile phone, containing an image and text (retrieved from [29])*

MMS has not been as successful as SMS, which is partly due to MMS not being as ubiquitous [30]. Fewer mobile phones support MMS and those that do usually require the user to configure the service themselves. Additionally, the fees for sending a message via MMS can be up to five times greater compared to SMS, depending on the network provider [31].

### 2.2.3 Electronic mail

Electronic mail (email) is an electronic communication method which started prior to SMS. In a fixed location, such as the work office or at home, it has been the most accepted method of communication since the adoption of the internet by society [32, 33]. With the introduction of wireless and mobile internet technology, mobile devices have been able to effectively include email facilities into the mobile environment. One such example is the *Blackberry* mobile phone range. *Blackberry* devices have seamlessly integrated email functionality (Figure 2.6) into a mobile environment through the use of a software application installed on the device that facilitates the sending, collection and organisation of messages on the device [9]. Emails are automatically and persistently forwarded when they are received from a server to the device, via the always-connected mobile internet [34].



**Figure 2.6:** An example email application running on a Blackberry device (retrieved from [35])

An advantage for a mobile device to communicate via email as opposed to SMS or MMS is that this method is free for sending messages [36]. In the case of a framework where a system is communicating to many devices, the system can always be connected to the internet. This would make it far cheaper to regularly send messages as there is no subsequent cost per message since the only cost is the running

of an internet connection. However, the costs of using the mobile internet can vary for a mobile user who would require internet allowance in their mobile phone contract to benefit.

In comparison to SMS, the sender of an email message has the ability to be more descriptive when creating the message, without the word limits or restrictions imposed by SMS [37, 38]. The email functionality allows any type of media file to be included as an attachment. Images, audio and video can be sent via email at no extra cost [39]. Figure 2.7 illustrates an email message containing an attachment being sent to multiple recipients. This message has also exceeded the character count of a standard SMS message, whereby in the case of SMS there would be a further cost incurred.



**Figure 2.7:** *An email message being sent to multiple recipients, containing an attachment*

People are increasingly purchasing more than one mobile device to use. SMS messages are only stored locally on one device. Since the user's complete email collection is stored on the servers that provide the email address the user is not locked into using only one device. The user can access their messages from any device, which has the necessary mobile internet and email functionality [40].

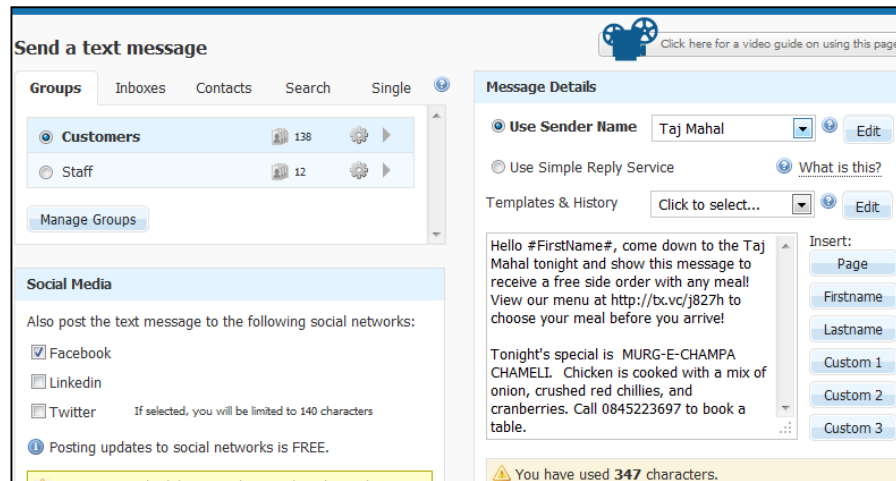
Mobile users tend to both check and respond quickly to text messages received, whereas emails are usually checked in less regular intervals [41]. For time-sensitive information this could be a crucial factor in deciding which communication method to use in order for the user to receive the relevant information. SMS is still a more universal method of sending and receiving messages over mobile phones since it is available on all devices, does not require access to the mobile internet and is standardised in the mobile environment [39].

#### 2.2.4 SMS web services

There are numerous online services that provide the facility for communicating via SMS from a fixed server to a mobile phone, by use of the internet. *TextLocal* [42] and *BulkSMS* [43] are examples of websites that offer this type of service. An SMS web service offers companies a generic method for sending and receiving messages to/from their customers. Purchasing bulk credits of SMS from the web service can reduce the cost of messaging in this way compared to individually sending messages from a mobile phone.

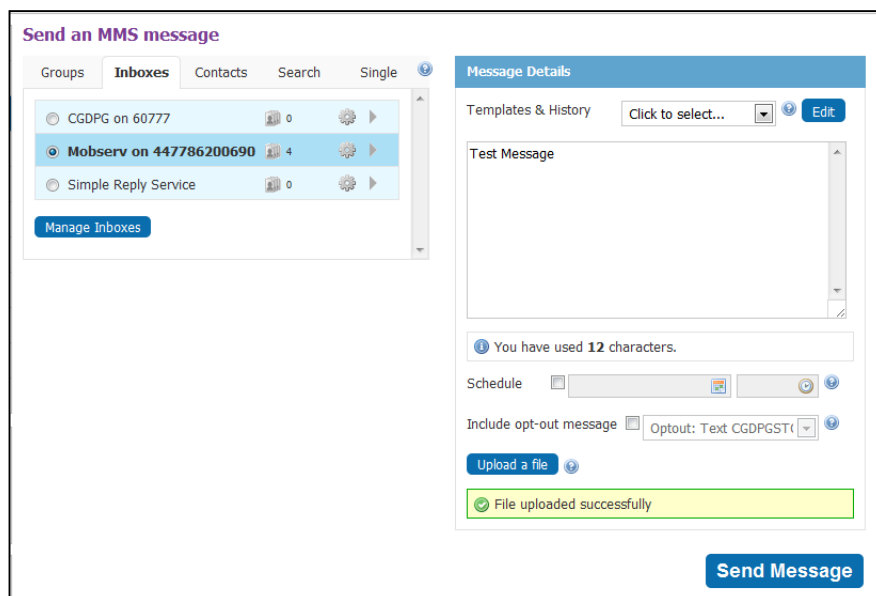
The different SMS web services have similar features to each other. They each allow their clients to bulk send a message to numerous recipients simultaneously. This is practical for broadcasting the same information to many people, for example sending alert notifications and advertising information to the client's customers, referred to as the users. Figure 2.8 illustrates a client interface for sending messages to their users, where in this example all the users would receive an advertising message. The SMS web service provides various methods for its clients to send and receive messages. One method for sending a message is the *Hypertext Transfer Protocol (HTTP) Post* method. The client makes a request via the HTTP protocol over the internet to send a message, providing the message details in defined parameter positions of the HTTP request. Another method is for the client to send an email of the message to the SMS web service, which then converts and sends it as a text message to the users. The text messages sent from the users are converted by the SMS web service and then sent as email messages to the client. An automated programme can exploit these methods to control the communication to and from mobile phones, without the need for human involvement.





**Figure 2.8:** TextLocal provides ability to send a bulk text to a group of users (retrieved from [42])

TextLocal [42] has the capability to send MMS messages in addition to SMS. In Figure 2.9 an image, audio or a video file can be uploaded and sent as an MMS message to multiple contacts. This means that multimedia files can be sent out as part of the message where textual information is not as satisfactory in describing a situation.



**Figure 2.9:** TextLocal webpage for sending an MMS message (retrieved from [42])

SMS web services can be effective generic systems that provide a means of communication for many situations and scenarios. Nonetheless they are limited due to being designed only as a facilitator for communication; with limited management of

the communication beyond ensuring messages arrive at their destination. There is no analysis of data within received messages or means to use a predefined template for a message. This handling and analysis of the communication would have to be controlled manually or by the client's own software system. However, signing up to an SMS web service makes the communication component of a system simpler and more efficient at working. This is due to the system being already in place and a tested method of communication, thereby providing a straightforward process of the exchange of messages between a client and its users.

### **2.2.5 Summary and conclusions**

The available methods that facilitate communication between mobile devices are being improved on a regular basis [44]. SMS, MMS and email have been selected for investigation in this chapter as each method has unique properties when utilised by users of mobile communication devices. The use of SMS is suited for simple text-based messages. SMS continues to be the most universal method of messaging over mobile phones since it is available across all levels of this device type [36] and is standardised in the mobile environment [39]. MMS supports enhanced features over SMS, including the sending and receiving of multimedia messages. However, it is not as ubiquitous as SMS and can be far more expensive when sending a large number of messages.

Communication via email is an effective method for users since it is a zero-cost solution to sending descriptive messages with multimedia attachments. Furthermore, the email communication method enables interactions between other types of mobile communication devices, such as tablet computers which are discussed in the next section. However, email is not universally accessible to all users of mobile phones and communication via this method requires the user to have a mobile internet data plan.

Each of the communication methods offer different benefits and drawbacks in relation to both the users and applications that are involved in the communication process. Therefore, when looking at person-machine messaging there is a need for a harmonious integration between the different communication methods. Each person may have a different communication preference or alternatively a user's preference may change at different times, which could depend on the information they want to

present in the message. Multimedia content could be more preferable to be sent by email. There are other factors, such as the capabilities of the device currently being used by a person, which would help determine their communication preference. A basic approach of communication for users is to offer only one option, for example email-to-email or SMS-to-SMS. In these cases the user is locked into just one method of communication. Integration between multiple methods of communication could include allowing the user to send information by the different methods with the system responding by one pre-set method of reply. In this example, a user may be able to send a message by either SMS or email but all the system's responses would be sent back to the user via email. The most complete integration of the multiple communication methods would be to allow the sending and receiving of each method, whereby the user determines their preference. For example, if the user sends a message via SMS future messages would be sent back to the user via SMS. If the user switches to email the system would follow suit. In this way greater flexibility is provided to the user as they are able to choose the most convenient method of communication.

## **2.3 Devices and technology**

In today's society many people now use mobile communication devices to perform an array of tasks. These tasks range from accessing the internet in a mobile environment to communicating via numerous methods to other mobile users. Unlike personal computers, mobile communication devices allow the user to work on tasks from any location [1]. Mobile phones and tablet computers are two such mobile devices. In comparison to fixed computer systems these mobile devices operate on cut-down operating systems. However, with the rapid progress in technology these devices are becoming both faster and more usable [45].

### **2.3.1 Mobile phones**

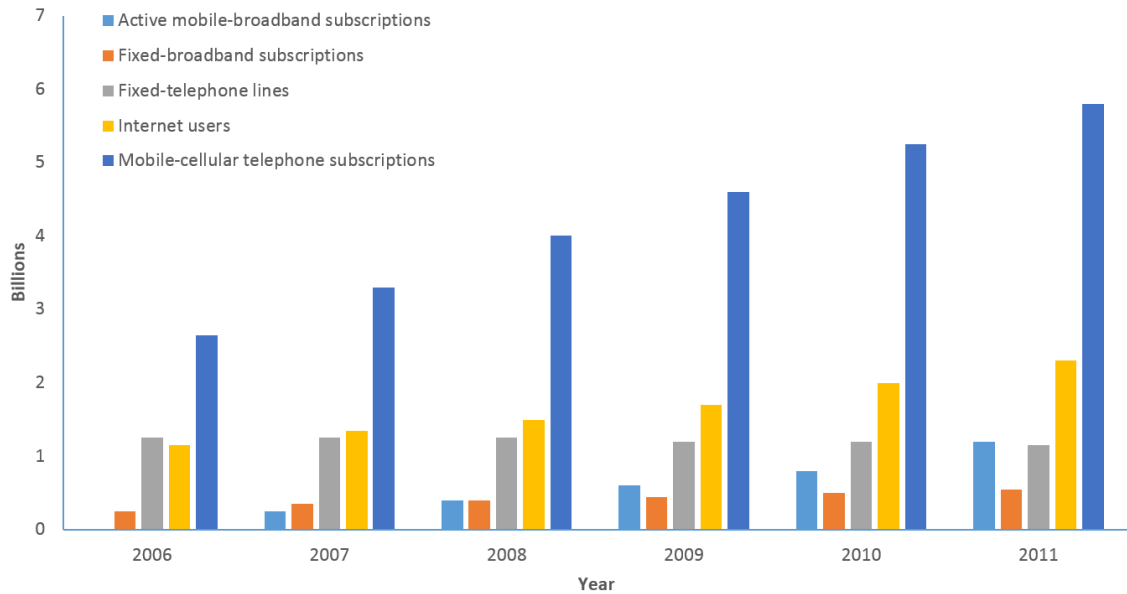
The device that has seen the largest rise in popularity in recent years is the mobile phone. The first generation mobile phone was introduced into the market in the 1980's [46]. Initially these were not easily portable as they were large bulky devices, illustrated by the device on the far-left in Figure 2.10. Mobile phones were expensive to purchase and access to the voice communication service was not ubiquitous as

there was no universal standard [6]. These factors contributed to the device not being widespread in popularity.



**Figure 2.10:** Evolution of the mobile phone from left to right (retrieved from [47])

The last two decades has seen tremendous growth in the mobile phone market with these devices being a must-have technology that the majority of the population now carries around with them at all times [2,4]. The newer models of mobile phones utilise a universal communication service, called the *Global System for Mobile Communication* (GSM). GSM provides a higher quality service for voice communication than earlier services had previously offered [48]. Additionally, new features have become available for mobile phones, including text messaging and internet access. This has allowed a global wide-reaching access to this new market, thus enabling the mobile phone to become a ubiquitous tool for communication [6]. During this time manufacturers were continuously improving the devices by decreasing the size and weight, whilst increasing battery life [2]. The size deduction is shown by the devices towards the right in Figure 2.10. Mobile phones have therefore been transformed into small multi-purpose gadgets which can easily fit into a user's pockets. The rapid growth in mobile phone subscriptions has continued yearly, as shown by the chart in Figure 2.11, growing by approximately 45% each year since 2006 [3]. The chart also illustrates that there are significantly more mobile phone subscriptions than in other available forms of communication, including fixed telephone lines.



**Figure 2.11:** *The rise of mobile phone subscribers worldwide from 2006-2011 and comparison with other forms of communication (adapted from [3])*

There are limitations with the functionality of mobile phones due to their design for mobile use. The input capabilities are not as comprehensive as those of larger fixed devices, such as personal computers. Mobile phones lack a full-size keyboard and mouse, with many requiring input via a numerical keypad. It is necessary to frequently press the same key in order to input the correct character [49]. Furthermore, the small screen limits the amount of information that can be displayed, which could make it difficult for users to clearly read the information.

### 2.3.2 Smartphones

The development of the smartphone has helped to resolve the issues arising from standard mobile phones. A smartphone is a mobile communication device that includes an advanced operating system and superior computing functionality in comparison to a standard mobile phone [50, 51]. Figure 2.12 illustrates a comparison between a smartphone and a standard mobile phone. There are many applications available to install on smartphones in order to benefit the user [52]. An example of this is satellite navigation software, which is a popular type of application that is available for smartphones [53]. This software enables users to obtain details of their current geographical position and also receive directions to locations of their choosing.



**Figure 2.12:** Comparison between a smartphone (left) (retrieved from [54]) and a standard keypad-style mobile phone (right) (retrieved from [55])

The usability of mobile devices has increased as a result of the advancements in the smartphone technology. Touchscreen interfaces allow the screen to be utilised as an input device by the touch of a finger. Larger and higher-resolution screens display a greater amount of information, as well as presenting this information in a clear way [56]. The *Apple iPhone* (left device in Figure 2.12) has received considerable praise amongst critics for its innovative display and easy to use touchscreen interface [57]. This has brought about a revolution in handset designs, facilitating a greater degree of interaction than previously possible [58]. An alternative device, the *Blackberry* smartphone, has a small *QWERTY* keyboard and trackpad integrated into the device [9, 59]. These are input components, analogous to a computer keyboard and mouse [60], to simplify the process of viewing and replying to email messages by offering quick navigation and input of text (Figure 2.13).



**Figure 2.13:** *Blackberry device with an integrated QWERTY keyboard and trackpad  
(retrieved from [61])*

A large number of mobile phones today have the hardware for accessing mobile internet services together with the appropriate software for specialised email facilities [59, 56]. The advancements in communication technologies together with the continuous improvements on processing speed have allowed mobile phones to access the internet at an acceptable speed and in a convenient way [45], as illustrated in Figure 2.14. These features are enabling people to be connected with each other, quickly and easily, wherever they are situated [62].



**Figure 2.14:** *iPhone accessing a website on the internet with the ability to zoom in on  
parts of the content (retrieved from [63])*

### 2.3.3 Tablet computers

In the past few years tablet computers have become very popular devices for consumers to own and use at a time and place convenient to them [64]. Figure 2.15 shows an image of *Apple's iPad*, which was the tablet computer that started the success of this new market [65]. Tablets are flat panel devices that combine the computing capabilities of a smartphone but with a larger touchscreen, which would usually be between 7-10 inches in size [66, 67]. The touchscreen is the input device with a large onscreen keyboard that can be easily hidden from view [68].



**Figure 2.15:** *Apple iPad tablet computer (retrieved from [65])*

A tablet can provide functionality more akin to a laptop, which includes a greater focus on content creation [69]. There are many applications available to install on tablets, which include writing word documents, creating presentations and editing images [70, 71]. These types of content creation are far more appropriate on a tablet than a smartphone due to the larger screen size and strong battery life of up to 10 hours of constant use [68]. The advancement in the availability of wireless connectivity for accessing the internet has also helped tablets to become a user-friendly method for social interaction activities amongst users [72]. Users are able to communicate with other people face-to-face, by voice or text and with the benefit of a built-in camera there are a range of communication applications available to the device [66]. All these factors have helped the tablet computer to become an attractive device for a consumer to use when they are both at a fixed location and on the move.



### 2.3.4 Integration of a digital camera

An integrated digital camera is now commonly found on mobile phones. This component allows the user to spontaneously take photographs of people, sites and other objects of interest that they experience in their daily lives [10], as shown in Figure 2.16. The first camera phones released in 2002 were of a low quality and insufficient for taking reasonable photographs when compared to standalone cameras. However, improvements in mobile technology have allowed camera phones to compete in quality. Many mobile phones now offer the same functionality as standalone cameras to take good quality photographs, such as autofocus, optical image stabilisation and a high resolution lens [73].



**Figure 2.16:** Mobile phone being used to take a photograph of the landscape  
(retrieved from [74])

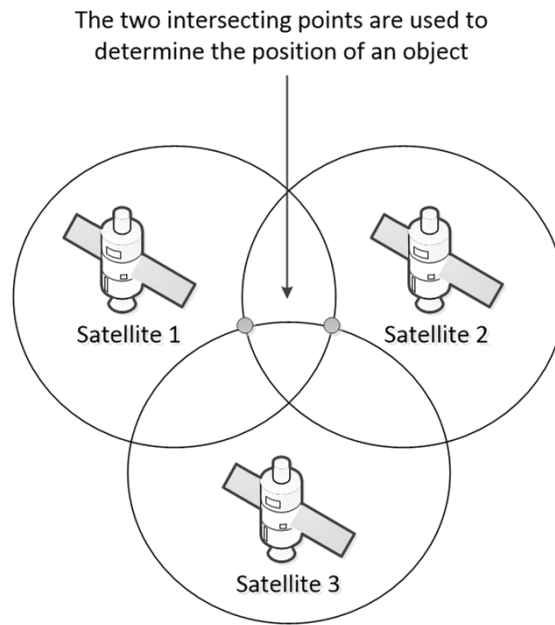
Whereas digital cameras brought the ability to instantaneously view captured images on the devices' screen, camera phones facilitate the possibility of being able to send the images to virtually anyone regardless of where they are situated [28]. Once an image is captured it is saved in the device's memory as a JPEG file, a compression technique to keep the file size small [75, 76] making it faster to send the file to a recipient. Information and events can be difficult or inappropriate to express vocally, due to having unique visual features. However, a camera phone is a device that allows not just the communication of information verbally but also visually. This facilitates the possibility of sharing visual events that the user considers to be of general interest or importance [31].

The integration into mobile phones of a good quality digital camera has had the effect that many people use these devices as a replacement to standalone compact cameras.

Mobile phones allow users to capture spontaneous events virtually anywhere in the world giving rise to new opportunities, such as amateur photojournalism. This is in contrast to standalone cameras, which a user would normally only have with them for specific purposes [10]. People with mobile phones have the ability to record anything they consider newsworthy, which can then be sent to friends or even news companies they believe would find the information of interest [10]. This has provided other uses where members of society can assist in the public interest, such as law enforcement. An example of this is where a shop owner in Sweden took a photograph with a mobile phone of someone shoplifting from their store. This photograph was supplied to the police and used to identify and find the criminal [28].

### **2.3.5 Location-based services**

*Global Positioning Satellite* (GPS) technology is a method that can locate users with high accuracy, which is now integrated into high-specification mobile phones. This is a navigational technology that utilises orbiting satellites (Figure 2.17) in order to locate the receiver device, which is based on its longitude and latitude positions [77, 78]. The user can be located with an accuracy of a few metres via GPS technology. Additionally, as the GPS hardware is integrated within the mobile phone, the location data of the device is computed and stored internally [79]. The position data can then be retrieved by navigation software that is installed on the device. This makes it possible for a mobile phone user to key in a particular address, for example a friend's house, and be returned with instructions on the route to travel based on their current location, as shown in Figure 2.18 [80]. Since a mobile phone is a small portable device that can easily be carried around by a user, it is an ideal piece of hardware to utilise GPS functionality [81].



**Figure 2.17:** GPS technology uses multiple satellites to accurately track an object's position (adapted from [82])



**Figure 2.18:** Mobile phone navigation software providing directions to reach a destination using GPS technology (retrieved from [83])

Mobile phones can now be localised using GPS technology making it possible for a centralised system to track the user's location. This allows the opportunity for location-based services, which provide information to a mobile phone user that is associated with their current geographical location [84]. These services have generated a large amount of interest recently due to both the increased number of

mobile phones on the market and the decreased cost of GPS-enabled phones. For example, a user can use a location-based service to find details of the closest hotel, cinema or restaurant, including the distance from their current position and the map location. *AroundMe* [85] is a mobile phone application, shown in Figure 2.19, which provides location-based services to help users find local businesses that are close to their current location. These services can also be used to obtain reports on weather or traffic at or near the user's current location [86]. Some GPS-enabled devices have the ability to send location data to any device it connects to. In the USA, when a call is made to the emergency services the location of the device is then sent to help locate the user quickly and direct the calls to the necessary site [78].



**Figure 2.19:** *AroundMe* application for mobile phones (retrieved from [87])

There have been experiments to investigate the collaborative use of mobile phone localisation, whereby more than one device is used for localisation to provide the possibility of cooperation between mobile users. An example system is where there is a central server that holds details of the current and likely future locations of various people, using their mobile devices and previous tracking data of their daily lives [81]. This system assigns users into groups of those most likely to be near each other, based on their expected location models at particular times. This information could then be used to coordinate and mobilise the users in a more efficient manner in order to tackle tasks. An example of this is where the system autonomously books a work meeting where the people who need to attend, according to their location models, should all be in the same building at the set time of the meeting.

### **2.3.6 Summary and conclusions**

Mobile phones and the various other mobile communication devices have helped to facilitate a new type of social interaction. People can now communicate “anytime, anywhere and for whatever reason” [2] allowing the possibility to interact and socialise with one’s friends virtually everywhere [4]. In comparison to other computing devices, such as desktops and laptops, it is common for one of these mobile devices to always be with the user in a ready to use mode [44]. Thus, with the capability to instantly connect this technology, people in a mobile society can take advantage of 24/7 access that was never previously possible. The result is a social network that is more connected worldwide than ever before [2]. Users can now carry multiple devices at any given time, each for a different purpose, with the choice of options ranging from a basic mobile phone to a tablet computer. In person-machine messaging it is a useful attribute for the system which provides services to a user to be able to cater for each device, in the most efficient manner, in order that the user can maintain communication at all times. This efficiency could include the cost of communicating or the functionality available to be able to send and receive information in the most practical way.

## **2.4 Conclusions**

The mobile phone has moved beyond a system of basic communication functionality into a multimedia and mobile computing device. These advancements in technology allow for possibilities far beyond the device’s original intent, giving the user the opportunity to partake in a variety of media interactions in a mobile environment. When users are looking to purchase a new mobile phone they now consider not only the selection of devices for voice calling but additionally whether the device has audio and video capabilities, internet technology and GPS functionality [4]. In addition to this, there are now other mobile devices and methods available to achieve communication and social interaction in the mobile environment, such as the tablet computer.

SMS has been found to be an effective and inexpensive method of communication that allows clear and concise information to be sent to users quickly, regardless of their whereabouts. However, only a limited amount of textual information can be sent to a receiver using this approach. An SMS web service can be used to handle the

transfer of messages between the system and its users. This service covers the SMS method and has the appropriate protocol in place for a client to send and receive messages. Through the use of MMS and email there are additional mediums of communication, such as photo and video facilities, which can be implemented into the communication process. These mediums can offer more informative content to the user than by text alone. For example, sending a photograph of a particular person to a user would make it easier for the user to identify them. Furthermore, a user out in the field could take a photograph of an item instead of describing it, providing enhanced details on the item.

A system that utilises people in a mobile society should integrate a wide range of communication methods in order to cover the different types of mobile communication devices. This would enable the users to interact with the system through the device they are currently using, via their own preferred method.

### **3 Review of Communication-Based Applications that Interact with Multiple Users**

#### **3.1 Introduction**

The purpose of this chapter is to investigate applications that have been developed to utilise people in order to solve a specific task. The main focus is on applications that utilise mobile communication methods, whereby it is possible to achieve an anytime and anywhere communication capability within a mobile society. There are a wide range of applications in circulation that utilise members of a mobile society for various purposes. In competitive reality television shows such as *The X Factor*, the general public are able to vote for their favourite contestant by sending in a text message during each stage of the competition [88]. Television and radio chat shows, including *The Wright Stuff* [89], allow their viewers to make comments or ask questions by sending in a text message concerning the current topic [22]. In this way it is possible to allow the general public to participate and contribute to the show. News websites, such as *BBC News* [90], encourage members of the public to send in photographs and videos taken on their camera phone regarding interesting news events [10]. Allowing members of society to participate in this way offers the news company the ability to find information from a wider range of resources, direct from people who have had first-hand experience of a news event. Groups in society have been able to utilise mobile communication methods to lobby governments for specific issues. The Indian government was forced to reopen a murder case due to activists organising an SMS petition, whereby the public sent text messages as a form of protest [91].

Several more advanced applications are discussed in this chapter that incorporate mobile communication methods for the purpose of utilising people in a mobile society. This is followed by applications which users interact with in a non-mobile environment but have valuable features and characteristics that can be included for applications that focus on a mobile environment. Therefore, both the technologies employed and the features contained within each application are identified and examined to determine their effectiveness in supporting users in a mobile society.

## 3.2 Applications utilising a mobile society

The applications discussed below operate in a mobile environment. These applications are surveyed to determine both the practical features that have been implemented and functionality that could be included to expand the applications.

### 3.2.1 *Pharmaceutical Care application*

In China an experimental scheme was established for using an SMS-based application to provide pharmaceutical care to patients [12]. According to patients there are often concerns regarding the usage of medication away from the hospital and it has been discovered that between 20-50% of the patients do not take their prescribed medication. This study looked at developing an application, entitled the *Mobile Pharmacy Service System* (MPSS), to provide individual care to patients using SMS with the aim of increasing the use of medicine whilst ensuring it is taken safely. The SMS-based application sends and receives text messages to provide a service for communication with patients. The application is linked to a database, which stores details of the patients' medication and the data regarding the various medicines. There is a user interface where the pharmacist is able to update the medication data. The application uses this data to generate individualised messages for patients, which are then sent using specialised SMS equipment. Messages are sent to patients each day consisting of a reminder to take their medication, advice on the correct methods of usage and warnings of adverse effects from prolonged usage. Patients have the ability to respond back to the application, allowing them to report on their current status or ask the pharmacist for guidance. The pharmacist has to manually reply using the application's interface. This flow of communication is shown in Figure 3.1.



**Figure 3.1:** *MPSS flow of communication (adapted from [12])*

This experiment was found to be successful amongst patients. The usage of mobile devices allowed patients to “review and remember guidance more clearly”, providing



the patients better knowledge with regard to their medication. The majority of patients were satisfied with the application, deeming that it provided a “closer link with their pharmacist”.

An issue of concern with this application is that very few elderly patients were able to participate. Poor eyesight prevented these users from reading the text messages clearly. A possible way to resolve this issue would be to provide alternative mediums of communication to allow the application to cover more people, such as picture and video messaging. Additionally, information supplied to the patient in other multimedia forms could be more effective at describing the situation. For example, a video or slideshow of the correct method of medicine usage could be more helpful to a patient than solely through text-based instructions.

### **3.2.2 Attendance Improvement applications**

An SMS-based application [92] was developed in Malaysia to investigate whether sending message reminders improves attendance rates in primary care. There are patients who regularly do not attend their appointments, disrupting their treatment and preventing other patients being able to book an earlier appointment. The wastage in time lowers the efficiency of the practice, thereby raising costs. A survey found that non-attendance was generally due to the patient forgetting or mixing up their appointment time. Therefore, the importance of this application was to provide reminders, containing the appointment details, to the patients before their appointment. This action was achieved by an employee at the practice manually sending text messages to patients 24-48 hours before their due appointment. The benefit of sending a text message is the convenience it provides to the user, whereby the user would receive the text message almost immediately. An experiment was performed with two groups of patients, where only one group received SMS reminders from the application. The results of this experiment showed that there was significantly more attendance to appointments by those using the application. The application was shown to be effective by raising the probability that each patient attended their correct appointment.

An advantage of using an application based on sending text messages is that it is possible to construct a computerised process whereby text messages are sent automatically, at a predefined time, before the appointment. This process has not been

carried out in the *Attendance Improvement* application, with the text messages manually sent by an employee of the practice. An automatic process would lessen the workload of the employee reducing the required manpower. Additionally, there is a higher chance of mistakes occurring in applications that are reliant on human input than that of an autonomous computerised system. A further advantage would be to combine this automatic process with a feedback mechanism as text reminders can be re-sent automatically to a patient if, after a predefined length of time, no confirmation message has been received from the patient.

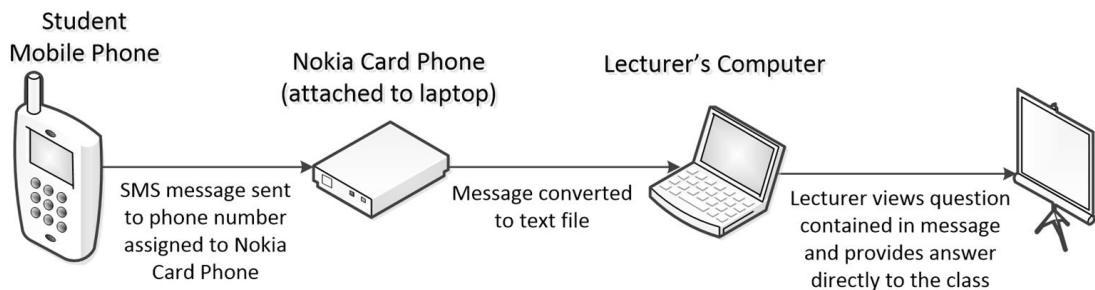
Computerised functionality was utilised in a second application, applied in a Melbourne hospital, to investigate whether sending SMS reminders improved outpatient attendance [93, 94]. The hospital's *Outpatient Clinic Scheduling System* contained the necessary data for the reminders which was uploaded and stored into a database. The database could then be queried by the SMS application's interface (Figure 3.2) to generate a personalised reminder message. The data was appropriately placed in an SMS template. On a daily basis a batch of messages would autonomously be sent to patients who have appointments due in three days. This time period, whilst helping to lower the chances of patients forgetting their appointment, gave a necessary period of time to re-fill appointments in the case of cancellations. Results showed the application was effective in increasing attendance rates to appointments.

**Figure 3.2:** *Telstar Mobile Online SMS System used to generate personalised messages (retrieved from [94])*

### 3.2.3 Interactive Learning application

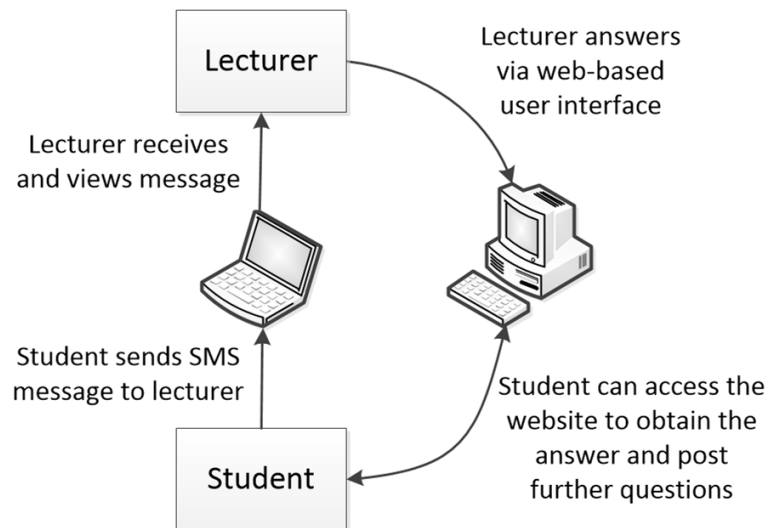
A college in Dublin has experimented with an SMS-based application [95, 96] in order to improve interactivity during lectures. Increased interaction during a lecture

between students and the lecturer has reportedly led to a more active learning environment. This has helped to motivate the students, developing a community atmosphere and supplying lecturers with valuable feedback. With the ability to generate fast and concise messages an SMS-based application was utilised to allow students to send messages from a mobile phone, in real-time, during their lectures. Students would have the ability to anonymously send a text message to a device held by the lecturer to ask questions or request the lecturer to clarify points. Figure 3.3 illustrates the process where a message, containing a question, is sent by the student and received by a *Nokia Card Phone* attached to a computer. The message is then converted to a text file for the lecturer to view it and provide answers to the entire class.



**Figure 3.3:** *Process of receiving the question from students (adapted from [95])*

This is an application where only a limited amount of financial and technical support is required, since the majority of students already possess a mobile phone along with the knowledge to be able to operate the device. Additionally, the small size of the device means it is unobtrusive in the environment. The originators of the messages are not shown to the lecturer. Normally, 38% of students never make interactions within a lecture. Providing anonymity should reduce this number, as those students who are usually too shy or fear asking questions would now have the opportunity to interact with the lecturer. This application could be utilised anywhere by the students, as the mobile phones can be carried and used outside of the lecture to send a question from any location. During lectures the lecturer can directly provide the answer whilst outside of lectures answers would be provided via the application's website. A student is then able to access the website to obtain the answer and submit a response or ask a further question. Figure 3.4 illustrates the communication process for this application outside of lectures.



**Figure 3.4:** Message loop between a student and lecturer outside of lectures (adapted from [95])

The results for the application showed that 47% of the participants sent messages during lectures. The students found that the application effectively offered the chance to interact and raise issues without disrupting the lectures. This allowed the lecturers to solve the issues at the appropriate time.

All the students who took part in the experiment had a mobile phone. However, this did not necessarily mean that the students took the devices into each lecture or had any available credit on their device, which are both essential requirements for the participants to interact in this way. If either of these requirements were not met by a student it would limit their capability within the lecture. Furthermore, even if the students had sufficient credit to send messages there is still a cost for each sent message, which could have an effect on the decision to use this method of interaction. This application would benefit from multiple communication methods being available during a lecture. Smartphones can utilise other methods of communication, such as email. It would further help the student if the college had a wireless internet connection available as email messages sent would be free for the student with no credit required on their device.

Another issue with this application is the time it would take to create a message. A large number of low-cost mobile phones have a keypad where at least three characters have to be shared over each button. This can result in the time taken to create a message being too long, especially for the inexperienced user. The result is that it is a

distraction from the tuition in the classroom. The introduction of new technology, such as touchscreen smartphones, should help to reduce the time spent texting since these allow full *QWERTY* keyboards to be displayed on the screen.

#### **3.2.4 *Smoking Cessation application***

There are currently smoking cessation services available to help smokers stop or reduce the amount they smoke [97]. However, these services are not utilised enough by the younger generation. An SMS-based cessation service would provide a relatively inexpensive method to a significant number of the population, especially the younger population where the vast majority own a mobile phone [98] and communicate via SMS [21]. In New Zealand a trial was conducted on two groups of smokers, where only the first group received an SMS-based cessation service. The second group could receive any other type of cessation services.

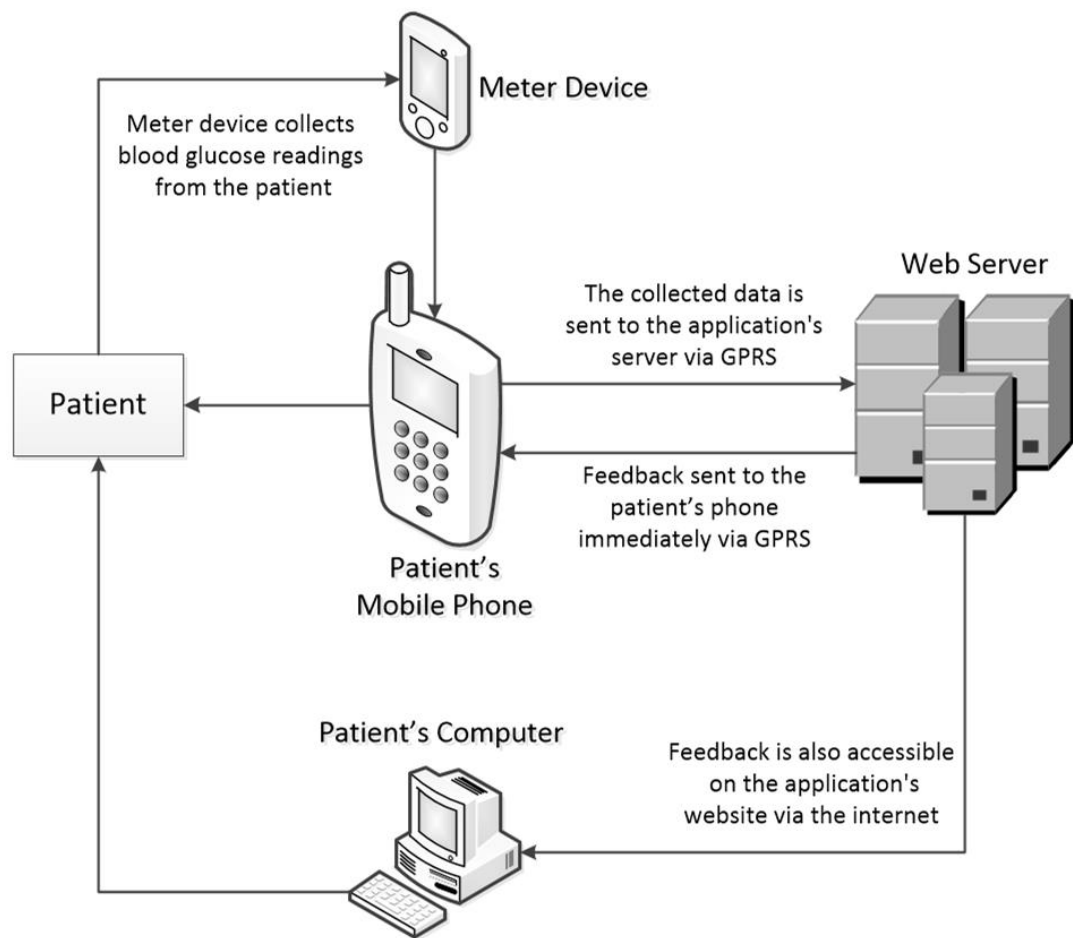
Users in the first group would receive text messages providing advice on the ways to stop smoking, support on coping with withdrawal symptoms and a range of interests to offer as a distraction. This was achieved using an algorithm that sent appropriate messages from a database that stored over a thousand messages, based on the person's characteristics. The application sent out the text messages by having functionality in place to query the database for generating the content of a message. This set-up reduced both the manpower required and chance of human error. For the first six weeks the smoker would receive regular text messages daily to help them stop smoking, then after that period the messages would slowly be reduced to keep up the cessation. Over the first 6 weeks it was found that over double the participants stopped smoking in the group that received the SMS service in comparison to the other group.

An advantage of this application is that it can reach areas of society that are hard to physically communicate with, such as people in rural areas or with disabilities. This has been made possible since mobile network services are available over a large geographical area and mobile phones are usually within the vicinity of the owner. The current geographical location of the user's mobile phone is insignificant as long as the device can obtain network signal.

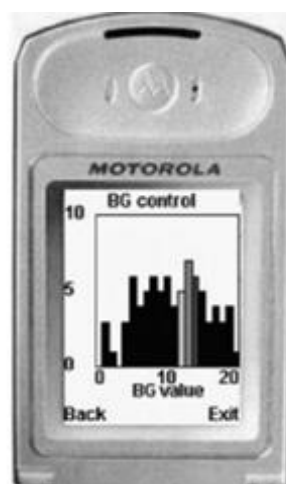
This service could be improved by utilising the new features of mobile phones, such as their multimedia capabilities. Using video and audio files can provide more effective and informative content to the user. However, this type of service could restrict certain parts of societies, for example those who can only afford an inexpensive mobile phone that is just equipped with basic functionality.

### **3.2.5 *Telemedicine Monitor* application**

An application to provide management for people suffering from the condition of diabetes [99] was developed at Oxford University. This application manages the condition, with the use of mobile phones that incorporate *General Packet Radio Service* (GPRS) technology, by sending and receiving data in real-time to a secure central server at the clinic. A meter device for measuring a patient's blood glucose level is connected to their mobile phone. A java applet on the mobile phone then takes these readings and sends them via GPRS to the server. This method enables the patient to send regular meter readings to the server daily and receive back the required continual adjustments needed for optimal support, whilst away from the clinician. Figure 3.5 illustrates the process of this communication, whereby the server provides feedback to the patient's mobile phone and an internet webpage for the user to view. With the use of the java applet, the phone has a screen suitable for displaying graphical data which is sent back from the server. One such example is a histogram that displays data on the blood glucose levels over the last day or week in graphical format. The mobile phone's colour screen is utilised with a colour-coded histogram to show the patient where the level is too low or high, as illustrated in Figure 3.6. This is to provide the necessary help to make adjustments and keep the level within the required limits. Additionally, it is possible to contact the nurse via SMS to ask questions and receive daily reminders to perform this procedure.



**Figure 3.5:** Process of communication between a patient and the application's server  
(adapted from [99])



**Figure 3.6:** Feedback histogram on display (retrieved from [99])

Results from an experiment showed that there was a substantial improvement in maintaining the correct blood glucose levels with patients using the application in comparison to those in the control group. This application has several advantages over the previous SMS-based applications. The collected data was recorded directly from a meter device (Figure 3.7), instead of being manually inputted by the patient where there is risk of erroneous data being inserted. Using GPRS means the connection to the server is always active. Consequently, the data is provided in real-time and not intermittently, resulting in the feedback from the server being relevant at the time received. Furthermore, there is the possibility of the server accessing and analysing the data almost immediately after the reading has taken place. These features make the application more effective at maintaining a patient's blood glucose levels in comparison to a manual process of inputting the data.



**Figure 3.7:** Patient's mobile phone linked to a meter device (retrieved from [99])

This scheme is only available to mobile phone users who have GPRS-enabled devices with the capability to run java software. Therefore, it has the potential of cutting off segments of society where people either have not upgraded their phones for a considerable time or can only afford an inexpensive device. There could be an alternative option to manually send the meter readings via SMS in order to allow patients with basic mobile phones to be included in the scheme.

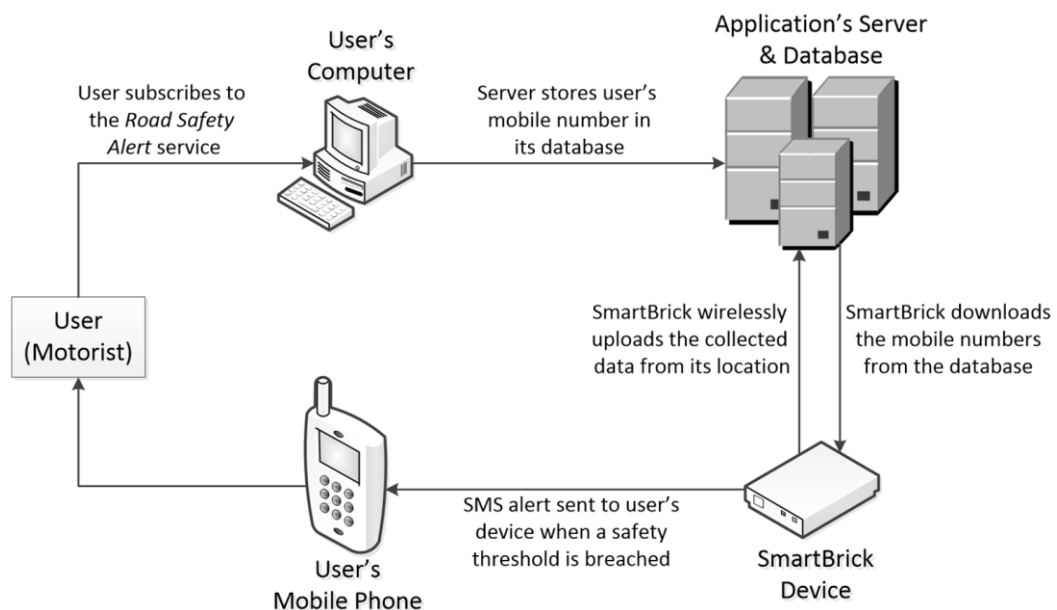
The ability to display graphical data on a mobile screen can be a useful approach for patients to monitor their conditions. However, people with impaired eyesight, such as the older generation may struggle to see the charts clearly. Having a webpage



dedicated to each patient's recorded information aids this issue as results can be viewed on a larger screen.

### 3.2.6 Road Safety Alert application

A *Road Safety Alert* application [100] has been developed at Missouri University. This application is designed both to detect a possible hazardous environment from man-made infrastructure and the natural environment, such as high-levels of water leading to a flood. Devices, called *SmartBricks*, are installed in these locations. This is a low-powered device that measures various properties of the locations, working autonomously, based on timers within the circuit board. All these devices are connected wirelessly to a remote server. Mobile phone users can subscribe to the safety alert service to receive SMS alerts of potential hazardous situations in locations that they request. The user initially subscribes via their computer over the internet, providing their mobile number to the application's server. The user's mobile number is stored in a database on the server. Each *SmartBrick* device periodically downloads measurement data to its on-board memory. For each property, safety threshold values are defined with the device raising an alert if these values are breached. The data relating to a threshold breach is wirelessly sent to the remote server, which subsequently sends SMS alert messages to the users based on the location of the *SmartBrick* device. These messages provide the users with the necessary warning details when an alert has been triggered. This process is illustrated in Figure 3.8.



**Figure 3.8:** Process of the Road Safety Alert service (adapted from [100])

The location-based features of this service mean that users are only updated about hazards that occur in locations meaningful to them. By providing this early alert to users they can be prepared for the hazard by either delaying or cancelling their journey or taking an alternative route to reach their destination. This service not only supplies the user with valuable information but also reduces the possibility of incidents occurring at the location, easing the disaster management procedure due to the hazard. These alerts are provided in close to real-time with the data sent from the *SmartBrick* device to the remote server immediately after a threshold has been breached. Sending the text messages should then occur within two minutes. The majority of mobile phone users keep their device close to them [44], thereby not making their current location an important factor when receiving alert messages. Additionally, this application can be very useful in remote and rural locations where suitable services for dealing with hazards are lacking. This is due to the devices being inexpensive to set up and to contact travellers, as well as the autonomy of the application. The *SmartBrick* devices can be left to run independently for long periods of time.

This type of application could be of further benefit if new functionality was integrated for two-way communication between the application's server and the users. The users could then work together with the application by reporting current or potential hazards. These reports could then be assessed by the application, with alerts sent to other users if necessary.

The users are only able to subscribe to this service over the internet. If the user decides to travel via a location that they have not subscribed for updates then they would be unaware of any current hazards in this location. The user would have to wait until they have an internet connection to add this location to their subscribed list. To resolve this issue a method for adding new locations to a user's subscribed list via SMS could be implemented. Users with any type of mobile phone would then be able to update their subscribed list when they are on the move.

### **3.2.7 *UbiquitousSurvey* system**

*UbiquitousSurvey* [13] is a system that enables its users to perform field studies by observing the population characteristics of various animals. The *UbiquitousSurvey* system makes use of mobile devices, such as smartphones, to gather this data from

the field. Each user out in the field is labelled a surveyor. There is a central server, which uses either a wireless connection or the mobile internet for communicating to each user's device in the field. The field data that is collected from the user's device is sent to the server and stored in its database. Figure 3.9 shows an example of field data being gathered by the mobile device.



**Figure 3.9:** Data being gathered by mobile device (retrieved from [13])

Due to the extensive geographical coverage of the mobile internet it is possible for these mobile devices to be used in areas that are vast distances from the server. Additionally, the mobile devices have software installed to perform some of the basic analysis of the field data when a connection to the server is not available. For example, this software can find the mean, minimum or maximum value from a set of collected numerical data. These features facilitate an anywhere and anytime access ability.

Another role in this application is the advisor, who sets up assignments and generates electronic forms for data entry by using a standard computing device, such as a laptop. These assignments can help the advisor when coordinating the different users by providing each user with a different role to collect the field data efficiently. Part of the advisor's role is to manually analyse the uploaded field data in the database and make judgements on how the project is progressing, identifying where any alterations are necessary.

The users in the field remotely access the server to receive their assignments, via their mobile devices. The data gathered from the field can be inserted immediately into electronic forms via a web application. This data is then submitted to the server to be

stored in the database. The device can also be used to communicate with the advisor, if issues arise.

The *UbiquitousSurvey* system requires each user's mobile device to have specific software installed in order to communicate with the server. This has the advantage of allowing the software to handle the electronic forms to send data to the server, simplifying the process for the user. However, it is necessary to invest in the mobile devices that have the correct hardware. A potential user who has a mobile device with basic communication functionality and non-compatible hardware may therefore be prevented from accessing this type of application.

### **3.3 Non-mobile applications**

The applications discussed below operate in a non-mobile environment. The functionality of each application is explored to identify features that can be beneficial for implementation into a platform, which utilises people in a mobile environment.

#### **3.3.1 *HELP* system**

The *HELP* system [101] is used in health care to assist clinicians in patient support and decision making, through the management of patient data. Patient data is collected by nurses and physicians, referred to as the users, and inputted into client computers located throughout the hospital. This is achieved by software installed on the client computers that provide electronic forms for the users to input the data, which is then sent to the *HELP* system's server for storage in its database. The patient data can later be retrieved back by a user to perform data analysis, presented by the software's graphical user interface.

The software installed on the client computers is composed of three separate layers; a presentation, business and data layer. The presentation layer controls the graphical user interface. This is where data is gathered from users, as well as displaying data back to them. The business layer contains the logic and rules for the application to run and process the received data. Finally the communication layer is where the client computer sends requests to the server. This layer handles the sending of data to and the retrieval of data from the database.

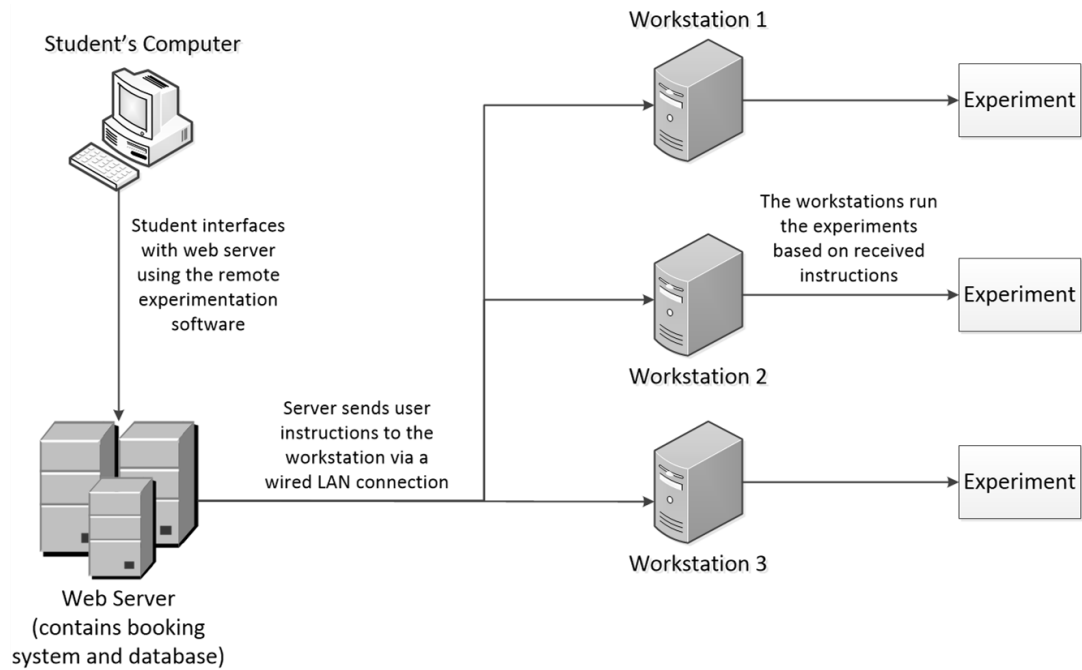
The system's server also consists of three layers. The communication layer is the counterpart to that of a client computer's communication layer, handling user requests

via communication protocols such as TCP/IP. The business layer holds the logic for the application and code to interact with the database. The database layer is composed of the data files for the database along with an interface to facilitate interactions with the database.

The *HELP* system's structure enables each client computer to perform its own tasks, with the main purpose of the server being to both hold and control the access to the system's database. This allows many client computers to simultaneously access the database in order to retrieve data or update/insert new records. Since the database is in one central location any updates are automatically reflected to all client computers. Incorporating an interface to control access to the database should reduce the chance of erroneous data being inserted, as users have to proceed through a structured process.

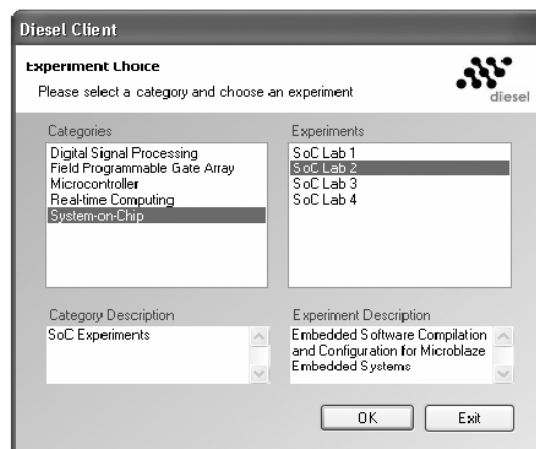
### **3.3.2 Remote Experimentation system**

A *Remote Experimentation* system has been discussed in [15] where students are able to remotely connect, from a distant location, to workstations in a laboratory to run experiments. The students achieve this remote access using their own computers, usually at their home. The laboratory workstations have the appropriate software and hardware components to control each experiment. A server controls the communication between student computers and workstations. The workstations are connected to the application's server via a wired LAN connection. Each student computer has the remote experimentation software installed to generate commands for the experiments. This software enables remote access to the experiments by sending web service requests, via the internet, to the server. Figure 3.10 illustrates the communication process for a student to remotely control an experiment.



**Figure 3.10:** Process for remote access to experiments (adapted from [15])

The system design consists of four layers; a presentation, data, business and physical layer. The presentation layer contains the applications that allow the students to interact with the system. This layer includes the *Diesel* client application, which contains the graphical user interface for remotely performing and controlling experiments. Figure 3.11 shows the user interface of the *Diesel* application for selecting an experiment to perform. Additionally, students are able to remotely submit booking times to perform an experiment via a web application. The submitted data is sent via the internet to the server's booking system.



**Figure 3.11:** User Interface of the Diesel client application (retrieved from [15])

The data layer is where the system's server resides. The server consists of a web service handler, booking system, and the database. Updates to the database are achieved via the web service handler and the booking system. The server application, for interacting with workstations, resides on the business layer. This application processes commands that have been sent from student computers, as web service requests. The application then sends the appropriate commands to the hardware for experimental procedures to be carried out, as instructed by the student. All of the hardware components that are required to perform the experimental procedures reside on the physical layer.

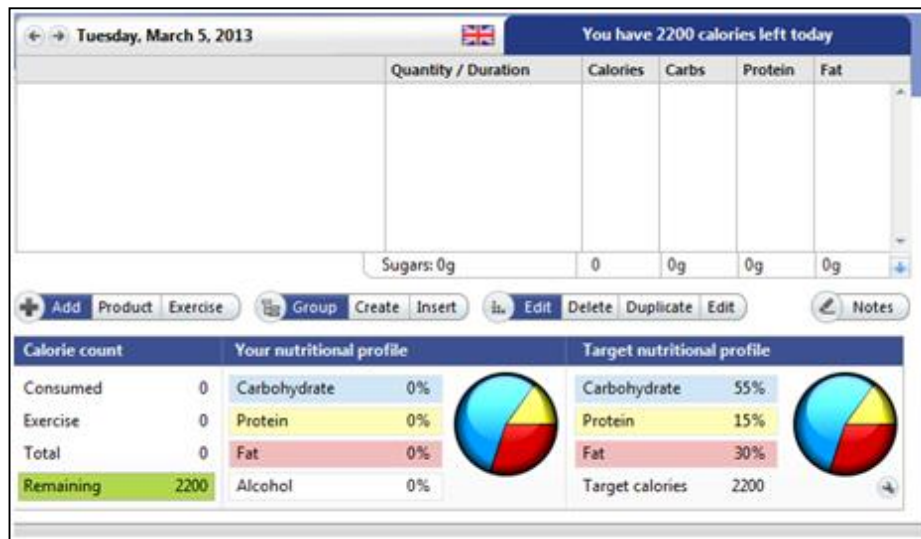
The advantages of this set-up are that the server facilitates long distance access and takes away from the student's computer much of the processing tasks. A student is able to connect to a workstation, for the purpose of working on an experiment, from any computer that has the required software installed. This can be achieved from any location where there is an internet connection. This removes the need for students to have to physically come into the laboratory, thereby making the experimentation process more hassle-free for the students. A student's computer only needs the processing power to send out experimentation commands, with the experiment being carried out via the server and separate workstations. This means that only the server-side components need the required hardware to run the application, whilst the student computers can be composed of basic hardware.

### **3.3.3 *Perfect Diet Tracker* application**

The *Perfect Diet Tracker* [102] is a software application designed to monitor the daily intake of food consumed and the daily exercises performed by a user. This is to assist the user in managing their weight change to meet a goal weight. The application calculates the necessary daily calorie intake target values, based on the target weight provided by the user.

Initially, the user inputs basic information concerning their current and target weight. The application then takes the user to the main screen (Figure 3.12), which shows a list of food products consumed and exercises performed on the current day. This screen also provides information about the calorie target for the day, including the user's remaining calorie allowance. On the diet side of the application, the user is able

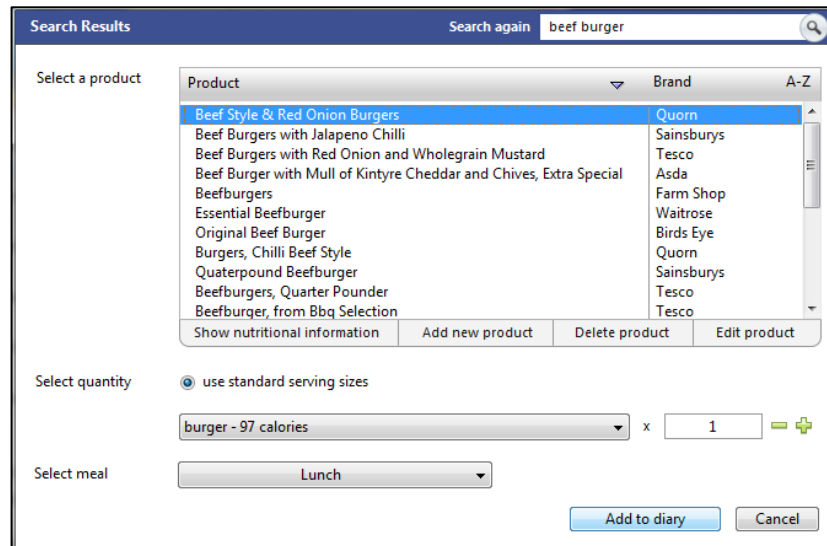
to keep track of various nutritional information. This includes the carbohydrates, proteins and fats of the products that the user consume.



**Figure 3.12:** Main screen of Perfect Diet Tracker showing nutritional information for the current day (retrieved from [102])

When the user consumes a food product they have two ways of inputting the product's nutritional data. The first method involves searching through the application's database to find out if the application already contains data on that particular product. The database contains data on numerous food products. Figure 3.13 illustrates the user searching for a specific product in the database. Once the product has been located the user inputs the quantity eaten either by weight or units consumed, for example a quantity of sandwiches. If the user is unable to find the food product on the application then the second method is to manually create a new food product and input the product's nutritional values, in addition to the quantity consumed. Each food product is then placed on the current day's diet list along with their nutritional values. The user is able to regularly check their remaining calorie intake allowance for the current day in order to stay on track with the set target. The daily calorie intake data is updated on each occasion a new product is added to the day's list.

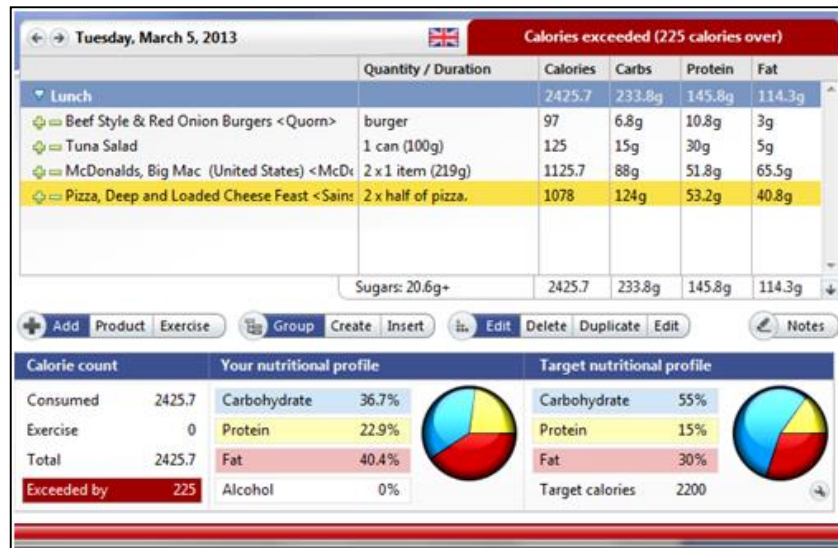




**Figure 3.13:** User searching for a “Beef Burger” food product (retrieved from [102])

To support the application’s goal of managing the user’s weight, it is also possible for the user to input data on exercises performed throughout the day. Exercises range from running and cycling to other sports, such as tennis. As with food products, the input of data can be achieved manually or by searching the list of exercises stored in the application’s database. When choosing to search through the exercise list, the user selects the correct exercise and then inputs the time taken and distance covered, if these values are required. The application uses this data to calculate the calorie burn value and update the daily total, allowing the user to view the number of calories burned and their remaining calorie allowance for the day.

The application’s database stores the user’s food and exercise data for each day, providing a record-keeping service that takes away the need for the user to store any data by their own means. This application provides real-time updates to the user by automatically analysing the data that has been inputted. This feature combined with the ability to set alert levels if the user breaches daily targets provides the user with a high level of interaction with the application to help achieve their targets. Figure 3.14 demonstrates the application’s behaviour if the user exceeds their daily calorie target, whereby the calorie information is highlighted in red to notify the user of this breach.



**Figure 3.14:** Main screen highlighting a breach in the daily calorie target (retrieved from [102])

The identified features could be implemented into a similar application where users communicate in a mobile environment via mobile communication methods. The user would communicate with a server to update their daily diet diary, whereby the server would consist of the database and analysis modules. In this way, the server would be able to store historical data on the user's diet diary and autonomously analyse new diet data that has recently been sent from the user. After receiving new food and exercise items, feedback messages could be sent to the user providing details of their updated daily calorie totals. Additionally, an alert message could be sent to the user whenever the server has calculated that a calorie target has been breached. Therefore, the user would be able to update their diet diary whilst they are mobile.

### 3.4 Social networking sites

A social networking site (SNS) is an online service which allows people to sign up as users and create their own public profile [103]. Through this profile the user can connect and interact with other users of the website or express their own views in an online community [104]. Interaction can consist of direct communication with other users, uploading content to the site in order to share information/ideas or collaboration between users in order to achieve a specific goal [105,106]. A user's social network may consist of people they know in the real world, such as friends and family. Alternatively, the user's social network can include people they have connected to online, who may share common interests with the user [107]. These

services have taken advantage of mobile communication methods, such as SMS and the mobile internet, to provide ways for users to connect to their social network and upload content while on the move.

*Facebook* [108] is an SNS with one of the highest number of registered users [109]. Each user profile contains attributes on the user, for example their location, workplace and birthday. Users can enlarge their social network by requesting friendships with other users of *Facebook* [110]. Each profile has an interactive wall, where people in a user's social network are able to post comments and share content [106]. *Facebook* has been integrated into a mobile society by the development of its mobile application on smartphones, providing functionality for uploading content while on the move. The mobile application can utilise a phone's GPS hardware with its "Check-In" feature [111]. This feature enables a user to share details of locations they are currently visiting. Furthermore, *Facebook* can offer deals and rewards to the user based on their current location. Figure 3.15 shows a deal that has been provided to a mobile phone user who has updated their current location to a particular coffee shop.



**Figure 3.15:** Facebook Deal made available on a mobile device (retrieved from [112])

Users can collaborate and coordinate with each other through the *Groups* and *Events* features of *Facebook*. *Facebook Groups* allow users to discuss ideas and share content with people who have similar interests. A *Group* may be themed on a particular interest, activity or organisation [113]. *Facebook Events* provide facilities

for organising events, which could include meetings, parties or even protests [114]. Figure 3.16 shows the web-form for creating a new *Facebook Event*. These features have been utilised in a mobile society to help organise movements. An example of this was the Arab Springs in 2011 where *Facebook* allowed a method of notifying people of protest locations and times, as well as sharing ideas about the countries with other people from afar [106].

**Figure 3.16:** *Create Event web-form on Facebook (retrieved from [108])*

SNSs share some characteristics with communication methods, such as email [115]. *Facebook's* private messaging system enables users to send private messages to their peers [116]. Users tend to send messages via email for both social and formal situations, where a message may be part of a longer conversation or a single message with no continuation from the receiver [115]. However, the purpose of communication within SNSs can be substantially different to email. SNSs require the user to construct and manage a social network consisting of both peers and people with similar interests, making these services more utilised for social communication purposes [117]. SNSs also offer alternative ways for users to connect with their network of peers [105]. As previously discussed, this could involve public discussions within *Facebook Groups* [113]. Additionally, if a user writes a wall-post publicly on another user's interactive wall, other users that are within the social network of the receiver could continue the communication by posting further comments. These comments would be displayed directly below the original wall-post. In this way SNSs support communication that is both one-to-one and one-to-many amongst the peers of a social network, enabling both private and public means of communication [117].

There are many other examples of SNSs where people in society can interact with each other on an online environment. *Twitter* [118] is a microblogging SNS where users can post messages, known as tweets, on any topic of interest to them [119]. For example, a user can tweet about their daily activities or share their views on an issue which is important to them [120]. Other users of *Twitter* can choose to follow them, receiving tweets when they are posted to the site. Camera technology in mobile phones has enabled users to capture high quality photographs. *Instagram* [121] is a photo-sharing SNS that takes advantage of this component [122]. Once a photograph is taken with a mobile phone users can upload the image file to their *Instagram* profile, sharing the image in an online gallery where connected users can view and comment on the content [123].

The SNSs discussed provide mobile applications for smartphones, offering a facility for users to stay connected with their social network whilst mobile. Some SNSs also provide features for mobile phones that do not have internet or other advanced capabilities. Users can also be alerted of new events on their online profile via multiple means. For example, a *Facebook* user can request to receive email messages for new notifications regarding their profile. An email notification can inform a user when a comment has been posted by another user on their interactive wall or on a *Group* page that they have joined [124]. A user may also receive a notification if a new person has requested to join their social network or a private message has been sent to them [116]. There are many different types of email notifications that a user can choose to opt in to receiving. *Twitter* users are able to register their mobile number to their *Twitter* account. These users can then send and receive tweets on their mobile phone via SMS. When new tweets are posted by other people that the user is following they would be sent a text message of the tweet immediately [125]. SMS is also a practical method for the user to post new tweets as each tweet can only be a maximum of 140 characters, which is less than the size of an SMS message [126]. It is now possible for people to share information, which they find newsworthy, from any location and via a vast array of communication devices, as well as interact and collaborate with other people in a mobile society through these online social networks.

↓Application	→Technology	SMS	MMS	Email	GPRS/Internet	Basic Mobile Phone	Smartphone	Tablet
Pharmaceutical Care		✓				✓		
Attendance Improvement 1 & 2		✓				✓		
Interactive Learning		✓				✓		
Smoking Cessation		✓				✓		
Telemedicine Monitor					✓		✓	
Road Safety Alert		✓				✓		
UbiquitousSurvey					✓		✓	
HELP								
Remote Experimentation					✓			
Perfect Diet Tracker					✓			
Social Networking Sites		✓		✓	✓		✓	✓

*Table 3.1: Communication and technology usage list*

↓Application	→Features	User can send field data	User can send/receive feedback	Exchange of pictorial data	Alerts and reminders	Database/record keeping	Templates for individualised messages	Autonomous data analysis	Anonymous sending	Collaboration/coordination of users	Location based features
Pharmaceutical Care			✓		✓	✓					
Attendance Improvement 1					✓						
Attendance Improvement 2					✓	✓	✓				
Interactive Learning	✓	✓							✓		
Smoking Cessation		✓			✓		✓				
Telemedicine Monitor	✓	✓		✓		✓		✓			
Road Safety Alert					✓	✓		✓			✓
UbiquitousSurvey	✓	✓				✓	✓	✓		✓	
HELP	✓					✓		✓			
Remote Experimentation	✓							✓			
Perfect Diet Tracker	✓			✓	✓	✓		✓			
Social Networking Sites	✓	✓		✓	✓	✓				✓	✓

*Table 3.2: Feature usage list*

## 3.5 Conclusions

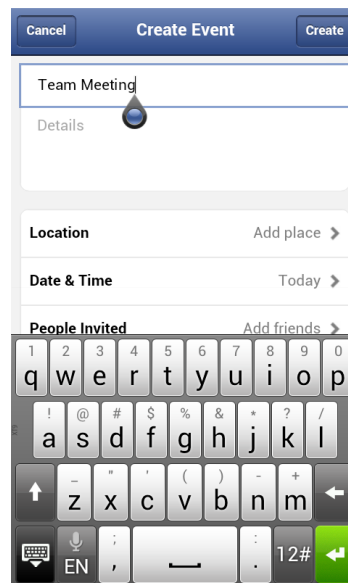
### 3.5.1 Communication methods and devices

Table 3.1 details the list of both the communication methods accessible to each application that has been discussed and the mobile communication devices that each application is suited to. The applications that have been implemented for a mobile environment focus mainly on providing support for one specific communication method. For example, the *Pharmaceutical Care* application [12] can only handle the exchange of messages via SMS. In Chapter 2 it was discussed that all modern mobile phones have the functionality to utilise SMS for the sending and receiving of messages. Therefore, it is only necessary for a user to own a basic mobile phone in order for them to be able to take advantage of this communication method. The ubiquity of SMS, together with the high popularity of mobile phone usage, makes the applications that are dependent on SMS highly accessible to any potential users in a mobile environment. However, a major downside of these applications is the limitations imposed by the SMS method, with the constraint of only being able to exchange information in a basic text format.

The *Telemedicine Monitor* application [99] helps to remove these limitations with the use of GPRS connectivity and a java applet on the user's device. These features provide a higher degree of functionality in the communication process, including a constant connection between the device and the application's server for real-time updates from both sides. The additional features of this type of application require mobile communication devices to support particular hardware and software functionality. A potential user whose device does not support these features would be prevented from accessing this type of application. Furthermore, this approach could become expensive for users who do not have a data allowance included in their mobile phone plan for using the mobile internet/GPRS. None of the applications have implemented functionality for communicating via MMS, which is possibly due to MMS not being as ubiquitous as SMS and email [30], in addition to the high expense incurred for sending a multimedia message.

SNSs, such as *Facebook* [108], go a step further by providing mobile web-forms for uploading text and multimedia content. These web-forms are tailored to request specific information, providing appropriate positions within the displayed form to

input the information. For example, a textbox can be selected at various positions on a web-form, where the user is requested to input text. Touchscreen devices would benefit from communicating via web-forms, as the user can select the appropriate textbox by touching it. This results in the touch keyboard appearing for the user to input the text. Figure 3.17 illustrates the use of a web-form on *Facebook* for inputting data, taking the *Create Event* web-form previously shown in Figure 3.16. In Figure 3.17 the user has selected the *Event Name* textbox and inputted the text “Team Meeting”. This process makes the input of data a simpler process than having to create a text message from scratch or edit details from a template. However, this communication method would not be practical for low-cost or older devices where there are limited or no internet capabilities or the processing of web-pages is slow. Users of basic mobile phones could utilise functionality provided by SNSs, which enables data to be exchanged between the service and a device via mobile communication methods. For example, Twitter users have the option of sending and receiving tweets via SMS [125,126]. Although limited in functionality, SMS could be a more convenient method for supplying an application with data in a mobile environment due to the basic requirements of this method.



**Figure 3.17:** User inputting data into a Facebook web-form (retrieved from [108])

The *Road Safety Alert* [100] and *Interactive Learning* [95] applications both enable the user to interact via multiple methods of communication. However, these interactions are limited depending on the circumstances. In the *Road Safety Alert*



application users can only update their subscribed list of hazard locations over the internet. SMS communication is limited to only receiving alerts. The *Interactive Learning* application requires communication via SMS during lectures, which can be an expensive method for the user. Whereas outside of lectures a user can send and receive messages via the application's website.

These issues demonstrate the need for an application that brings together the various methods of mobile communication with the intention of accommodating a wide range of potential users within a mobile environment. In addition, it is important for the user to have the choice of the communication method to be used for both the sending and receiving of messages. The implementation of multiple methods of communication, with the preference chosen by the user, would increase the flexibility in the way users can interact with an application. Therefore, such an application could utilise the additional functionality available in superior devices, to provide extra benefit to both the application and the user. At the same time, mobile phone users who own a less equipped device could choose a simpler communication method and thus still be able to interact with the application.

### **3.5.2 Application features**

Table 3.2 details a list of features that have been identified from the investigations of the applications in this chapter. The majority of the applications provide two-way communication in order that each user can send messages as well as receive messages from an application's server. Two key reasons have been identified for allowing users to send messages; the first reason is for providing the application with feedback on instructions previously received by the user and the second reason is for the user to provide field data to an application. The *Pharmaceutical Care* application [12] focuses on enabling its users to provide feedback on medication instructions that they have received by sending messages for further advice. Alternatively, the *UbiquitousSurvey* system [13] focuses on employing its users to collect and send field data. This application provides instructions to its users in order for them to gather the appropriate data when out in the field. The data are then sent to the application's server for analysis. This approach facilitates a higher level of interaction between an application and its users, as the users play an active role in assisting the application to reach its objectives.

The applications that receive field data from a user's mobile device include procedures for analysing the data, which are to either respond to the user or provide updates on the issues of the application. In the *UbiquitousSurvey* system, functionality is installed on each device to enable users to analyse their collected field data. However, there is limited autonomous analysis and no decision making capabilities. This application relies on a human operator (the advisor) to decide future actions, based on the supplied field data. The *Telemedicine Monitor* [99] is another application that requires each user's device to send in their field data. This application differs from the *UbiquitousSurvey* system through the implementation of autonomous analysis functionality. This functionality enables the application to process the received data immediately resulting in the updates being provided to the user in real-time. Updates sent to the user provide them with feedback, which could include instructions to make adjustments to their medication intake. This improves the efficiency of an application since it does not wait for a human operator to perform the process. Therefore, users can receive and respond to updates quicker in comparison to a process lacking in autonomous analysis capabilities. Furthermore, assuming the algorithms implemented for analysis are accurate, this approach would eliminate the possibility of human error from an operator who would be manually processing the received data. Providing functionality to analyse user data is an important factor in achieving an effective two-way communication structure. This functionality gives a greater degree of focus on user replies, enabling the user to effectively interact and assist with the objectives of an application through the sending and receiving of data in real-time.

The applications have shown that an effective method of storing communication data is by incorporating a database into the framework. For example, the *HELP* system [101] provides an interface for users to insert information on patients, which is then stored in the database. This database can be used by the application for the purposes of manipulating and analysing the data or to enable other users to access the data. Alternatively, the data stored in a database can be used by an application to autonomously send messages to users in a mobile environment. This is demonstrated by the *Smoking Cessation* application [97], whereby the application's database is composed of a table to store the textual content of numerous predefined messages. There is also an additional table to store data on each user's characteristics. The

application analyses a user's characteristics to determine and select a meaningful message to send to that particular user. The *Attendance Improvement* applications [93, 94] further develop this ability by focusing on sending personalised messages to their users. This feature relies on message templates, which are stored in the application's database, in order to provide a content structure for sending appointment reminders to users. The appointment data, also stored in the database, is inserted into the appropriate template creating a new message that is personalised to each user. This feature of merging a template with the relevant data allows a single type of message, such as an appointment reminder, to be sent on a frequent basis to numerous individuals with data unique to each person.

The *Perfect Diet Tracker* application [102] uses its database to provide functionality for sending alerts and reminders to users. When a user supplies data on a new food product consumed, the application processes this data to calculate the user's new daily calorie intake. Through this approach, alerts are autonomously generated if the current intake is greater than the target intake. The application combines autonomous analysis with the storage of historical data in a database to provide real-time alerts on a user's current diet status.

A feature that is found only in the *Interactive Learning* application [95] is the ability for a user to supply data anonymously. This feature is practical in situations where sensitive data is being transmitted or a user may not wish to be identified. Anonymity can encourage more people to be involved in issues, as shown by the results of the *Interactive Learning* application.

In the *UbiquitousSurvey* system [13] an advisor coordinates users in the field, providing each user with a different role in order that they can work towards the application's goal of observing population characteristics of various animals. *UbiquitousSurvey* utilises the collective intelligence of the user group, whereby grouping together a collection of users on a shared goal should result in faster or higher quality solutions than would be expected if only a single person was working towards the goal [127]. Therefore, implementing collective intelligence functionality is valuable for an application to incorporate in a mobile environment as the internet and mobile devices have made it straightforward for users to connect and network across large distances.

The geographical location of users can be utilised to efficiently employ people. In the *Road Safety Alert* application [100] users only receive warnings on locations that are relevant to them. If an application has up-date location data on its users then this feature could be extended to provide tasks to users that are relevant to their location. For example, a combination of location-based technology and the use of collective intelligence can enable the coordination of multiple users in solving a shared issue that is spread across a range of locations. An application that employs both these features can alert each user to the location of other users. Tasks could also be allocated in an efficient manner, which would be based on each user's proximity from the required areas.

Each application that has been investigated in this chapter is lacking in some of the features shown in Table 3.2. There is no application currently available that includes functionality to support all of these features. Creating an application that takes advantage of all the features could provide superior performance as the application would have a greater amount of options at its disposal to tackle its objectives. SNSs, such as *Facebook*, are better equipped with the functionality to use many of the discussed features. The Arab Spring example where *Facebook* users organised protests at a range of locations demonstrated how the combination of location and collective intelligence features were put to effective use. However, SNSs depend more on the intuition and direction of each user. There would need to be more focus in facilitating the application to direct users on a particular issue to achieve a desired result.

### **3.5.3 Necessity to combine features into a generic platform**

The focus for existing work is only on individual applications, where the frameworks within the applications are created only for a specific focal problem. The design and architecture for each application is therefore developed solely for that problem. For example, the *Pharmaceutical Care* application [12] focuses only on communication with outpatients to increase their medication usage. Many of the existing applications utilise similar types of components, for example a communications component to interact with users and a database to store the exchanged messages. However, the existing applications would require large changes in their design and framework to provide an effective solution to a new problem since they are developed solely for the

situation at hand, with any platform that is available to the developer. There is currently no integrated communications platform available for these types of applications to be rapidly developed and implemented for use in a mobile environment. Creating a new platform that has a framework of generic components to support multiple scenario applications would reduce the development effort for each new type of scenario.

The features that are listed in Table 3.2 could be incorporated into this new type of integrated communications platform to ensure there is a high degree of functionality for supporting different scenario applications. Additionally, implementing the capability to communicate through a multitude of mobile communication methods would allow the platform to cater for the various mobile communication devices that users may own. Each scenario application may have specific feature requirements. One application may require a small subset of the features, whilst another may require a different subset. For example, an application that focuses on managing multiple users to resolve an objective may require assignment functionality and location-based features. However, a different application that focuses on one-to-one interactions with individual users may only require functionality to send and receive basic text messages and a suitable template mechanism to personalise the messages.

A generic platform would attempt to cater for various different types of scenario applications. Therefore, it is important to consider the efficiency of the platform in handling each application type, both in the speed and ease of development and successful implementation. This involves ensuring the platform has expandable components to fulfil new types of requirements from application problems that arise in the future. A platform that has a framework of generic and expandable components, to support the features required by a scenario application, would be far more robust in reacting to new developments and issues arising in an ever changing world where new mobile devices and ways of users interacting with mobile services are emerging.

## 4 System Architectures

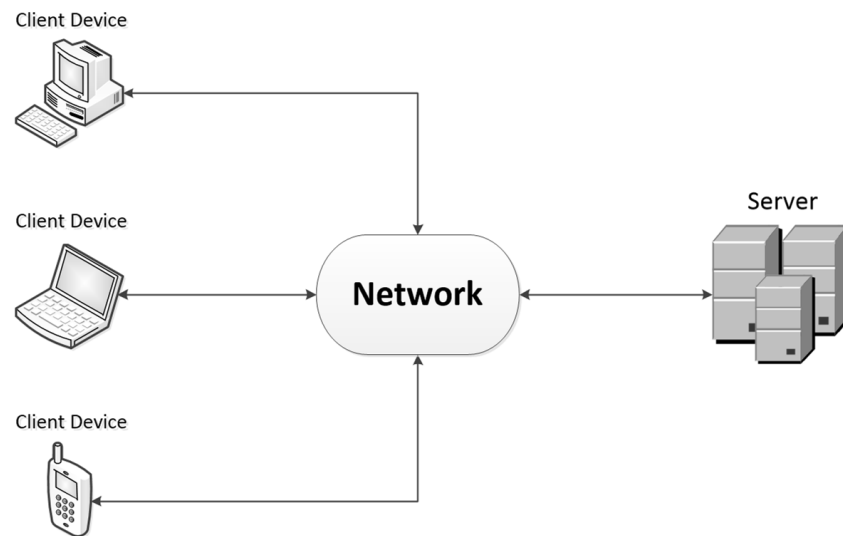
### 4.1 Introduction

Chapter 3 was concluded by discussing the need for a platform that could facilitate the running of multiple scenario applications. It is necessary for such a platform to be built based on an appropriate architectural design that can provide efficient functionality for the list of features identified in Chapter 3.

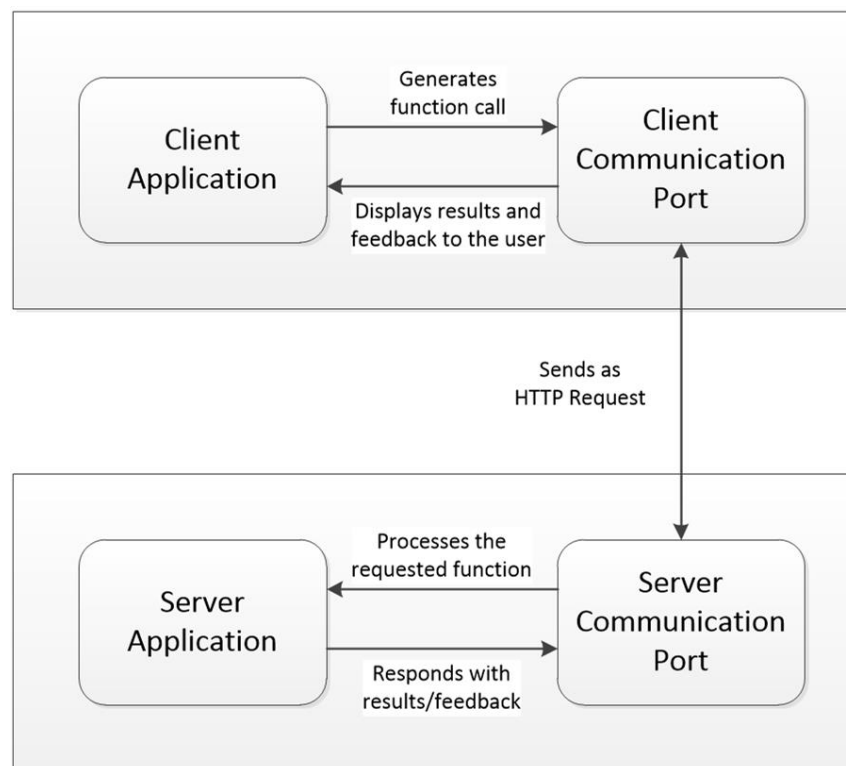
In this chapter there are four different types of system architectures being investigated for their practicality of utilising a multitude of mobile communication devices. The investigated types are the client-server, peer-to-peer, multi-agent and service-oriented architectures. There are many architectural types available that could be used to construct the platform. However, the four types that have been reviewed provide distinct attributes for enabling a platform to interact with users in a mobile and distributed environment, which are described in the following sections. The strengths and weaknesses of each type of architecture are discussed and used to determine which would be the most suitable for the proposed platform.

### 4.2 Client-server architecture

In a client-server architecture the operations are split between two types of components, client devices and a server [14]. Client devices are operated by users for sending requests to the server to perform specified tasks [128]. On the client device there is usually software installed that provides the user interface for interacting with the server [129]. The server consists of the necessary hardware and software to perform each task requested by a client device [130]. The server manages access to the various resources in the system, such as a database for data storage [131,132]. Figure 4.1 illustrates the interaction between the client devices and a server in a client-server architecture. Communication protocols are implemented in a client-server architecture to enable the sending of data between these two components (Figure 4.2). The *Remote Experimentation* system [15] is designed based on a client-server architecture. In this system a client device interfaces with the web server by sending HTTP requests in order to execute the requested functions. The server can send results to the client devices based on the consequences of a requested action via an agreed communication protocol [130].



**Figure 4.1:** Components of a client-server architecture (adapted from [131])



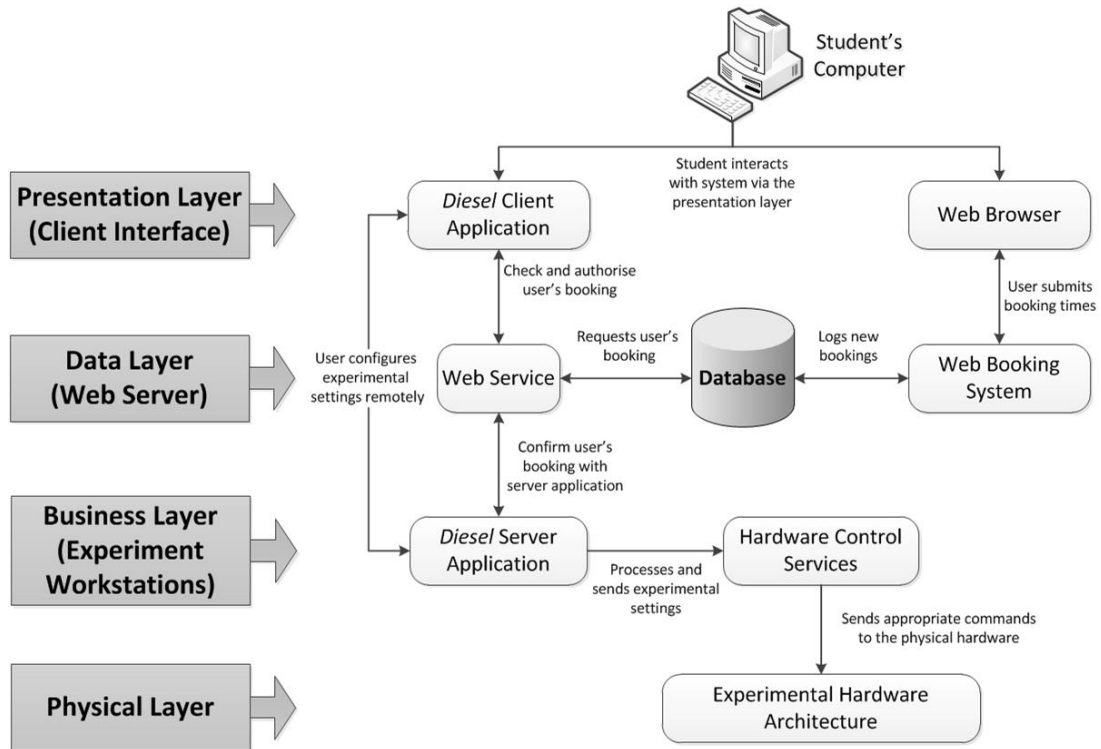
**Figure 4.2:** An example communication process between the client and server (adapted from [130])

Systems that utilise wide area networks and the internet for communication can include client devices that are a large geographical distance away from the server, with the only criteria being that the device is able to connect to the network. The

*UbiquitousSurvey* system [13] takes advantage of this aspect as client devices can collect field data in any location and send the data immediately to the server for storage and analysis. Removing the geographical limitations from a system's framework opens the way for more client devices to connect to a server.

Some systems implement a tiered communication model to facilitate the flow of data between the different components of the client-server architecture. Chapter 3 stated that the *Remote Experimentation* system [15] is composed of four layers; a presentation, business, data and physical layer. Figure 4.3 illustrates the way these layers interact with each other. The user interacts through a client application on the presentation layer, with protocols in place to connect each layer. There is a business layer to process requests that are received from users. In this way the server is performing the intensive processing of tasks, while the client device has a basic role of simply supplying data or requests to the server. The *HELP* system [101] implements a similar communication model. The data layer of the *HELP* system consists of both the system's database and the modules required to access the data-tables. In both of these systems the layers work together through a structured process to enable a seamless experience for the user, whether it is accessing and manipulating a database or configuring an experiment remotely. The end result is that the user can operate or take advantage of functionality that is not available on their own device. Server-side components are required to efficiently handle the intensive processing of requested tasks. However, the client devices can be composed of hardware that is relatively basic and inexpensive in comparison.





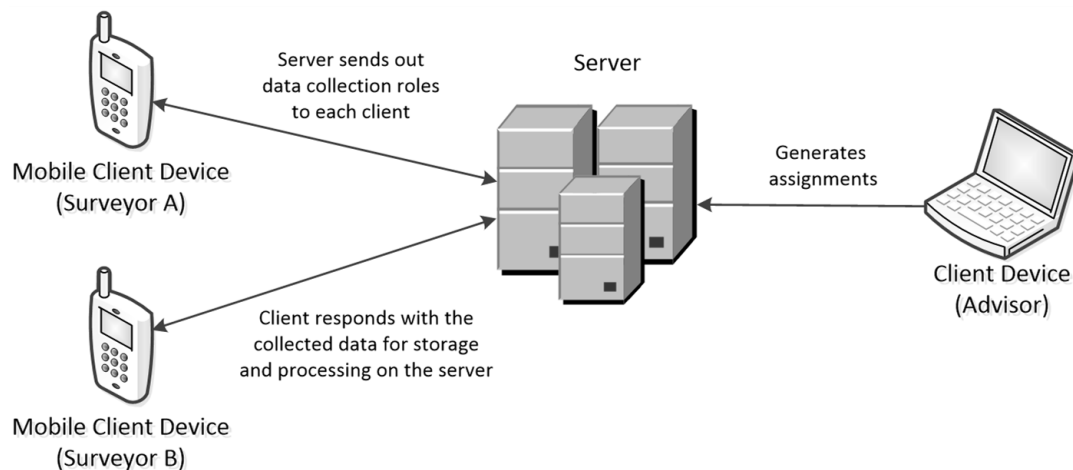
**Figure 4.3:** *Layers of the Remote Experimentation System (adapted from [15])*

The intensive processing that is performed on the server-side can involve the autonomous analysis of received data as with the *Telemedicine Monitor* application [99], whereby the application provides methods over GPRS for client devices to send collected data to the server. The server responds autonomously, based on the analysis results to alert users if any changes in their medication are required. This example illustrates how a client-server architecture can include the analysis and alert features that were discussed in Chapter 3.

An outcome of having a centralised server is that all the access abilities and data storage is controlled from one central location [133]. Multiple client devices can simultaneously access and query a server's database from any location making the client-server architecture a useful method for collecting field data from a large user group [132]. For example, the *HELP* system [99] has client computers installed at various locations where nurses and other users can access the system to add new patient data. As larger quantities of data are required to be stored only the server would need to be upgraded to handle the increased demand, which would not affect the client computers [14]. In the case of the *HELP* system, the database would be continuously expanding as new patient data is received from users. This may require

changes on the server-side to ensure there is no performance degradation in the querying of data in the database. However, this is not necessary on the client computer, as once the collected data is sent it does not need to be stored on the computer [134].

Implementing a centralised server also provides a central point to manage the coordination of multiple users [135]. The *UbiquitousSurvey* system [13] organises roles for each user and sends instructions for these roles to their client devices, as illustrated in Figure 4.4. Each client can be deployed efficiently in situations where the group of clients are required to work together to achieve a shared goal, utilising the collective intelligence of the group [127]. For example, an application on the server can ensure that each client has different tasks in order that they are not needlessly performed more than once. Furthermore, algorithms could be implemented on the server to allocate tasks based on specific requirements, such as selecting a user who is geographically closest to the task location.

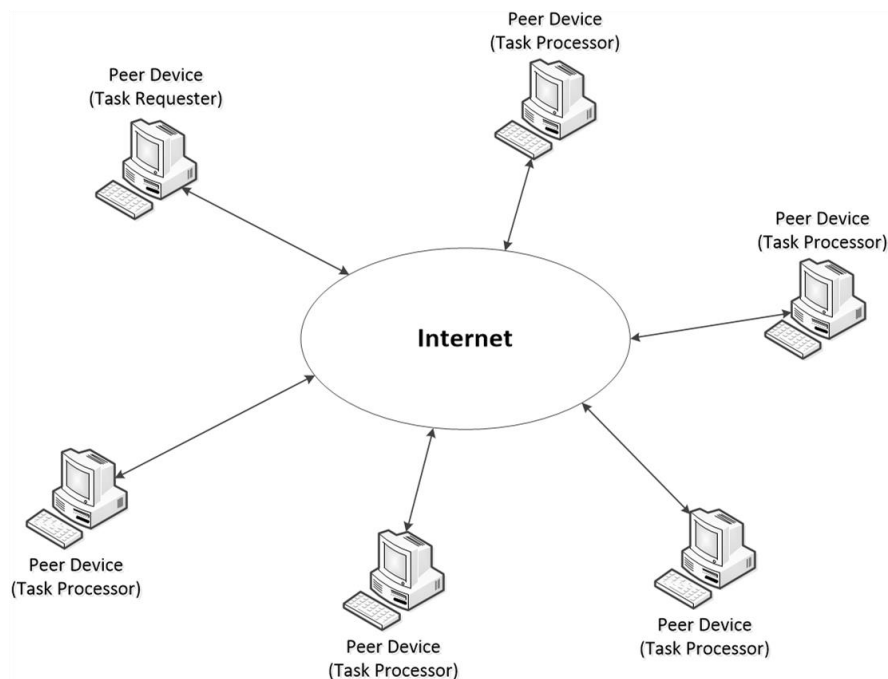


**Figure 4.4:** *UbiquitousSurvey* process of assigning data collection roles between the clients (adapted from [13])

However, there are a few drawbacks with implementing a client-server architecture. As the server is the central point of this architecture, it has to be operational at all times. A failure in the server would disrupt the entire network since all communication and requests are processed by the server [132]. Additionally, having all network traffic directed to one location can result in a congested network if clients make more requests than can be handled by the server [133]. This can then result in a delayed response time from the server.

### 4.3 Peer-to-peer architecture

A peer-to-peer (P2P) architecture moves away from a centralised framework by having a group of user computers, known as peers, connected together over a globally distributed network. Each peer acts as both server and client, sharing responsibilities and resources in the network [136]. Peers provide resources to the network, such as their computational power or file storage capacity. Other peers can utilise these resources for the processing or storage of data. Therefore, each peer shares in the cost of running the architecture and is required to collaborate with the others to access their resources for performing tasks [137]. Communication between peers is achieved by sending messages over the internet, whereby the peers can provide resources to each other [138]. Figure 4.5 illustrates the interactions between peers in a P2P architecture, where each peer can act as both a task requester and a task processor.

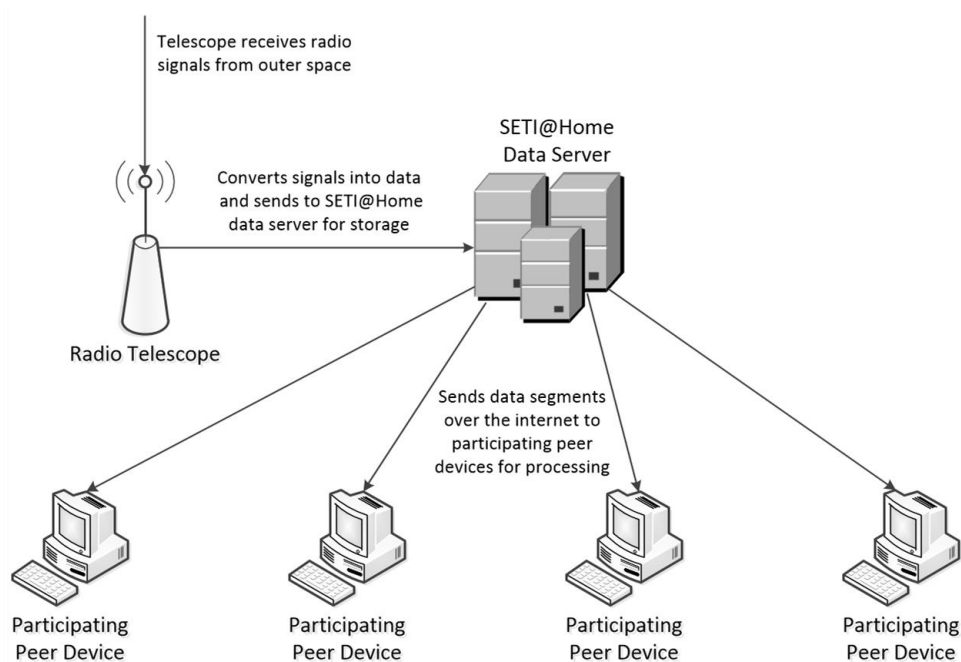


**Figure 4.5:** Components of a P2P architecture (adapted from [139])

The architecture is highly scalable as each time a new peer joins the network their resources are added to the network's pool of resources [137]. This results in an increase in the processing and memory capabilities of the network. The global reach of the internet has provided a way for countless computers to be connected, which is beneficial to the P2P architecture with the potential for near limitless resources.

The decentralised nature of a P2P architecture, with each peer acting as both client and server, prevents there being a single point of failure [140]. Each peer shares the cost of ownership within the network. Furthermore, this provides a more equal distribution of the network traffic as instead of a central server resolving all queries each peer helps towards the work.

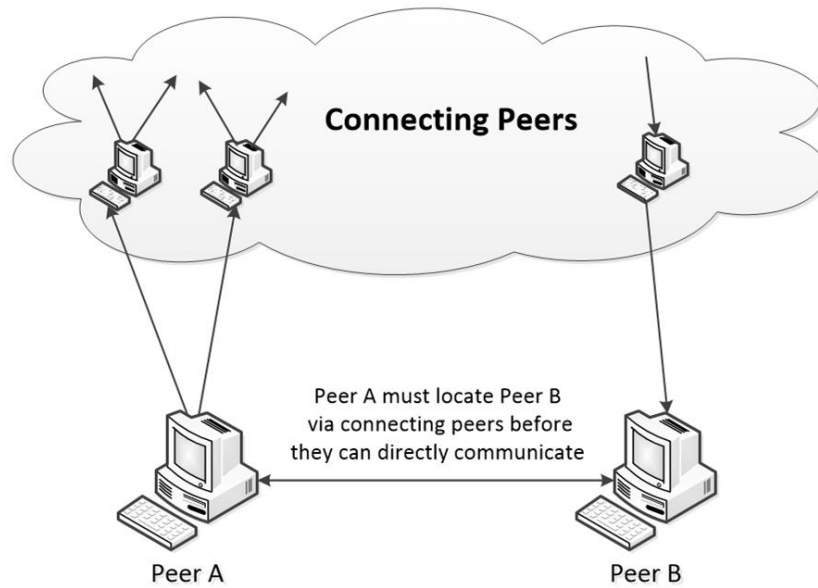
*SETI@Home* [141] is an application that takes advantage of P2P principles to benefit from distributed processing of data. *SETI@Home* utilises the combined processing power of numerous computers to analyse radio telescope data in a search for extra-terrestrial life [142], as illustrated in Figure 4.6. The computers that participate in the distributed processing of data are the commonly used internet connected devices, such as personal computers, owned by users who have signed up to the project. A radio telescope is used to receive and convert radio signals into data items. The data is sent to the *SETI@Home* data server, where it is split into small segments. Each segment is sent over the internet to different participating peer computer devices to be processed and analysed. *SETI@Home* creates a virtual supercomputer over the internet that can process data faster than a physical supercomputer, by processing millions of small segments in parallel [143]. This application shows the high scalability of the P2P principles as each time a new user joins the network, their computer's processing power can be added to the total available.



**Figure 4.6:** Framework of the *SETI@Home* Application (adapted from [143])

Peers can join or leave a P2P network at will, meaning that a peer's lifetime on the network could be very brief [139]. This issue can result in a constant change to the network's topology. The unreliable nature of peers can cause resources to become unavailable at any given time, leading to delays in the processing of data or in response to a request made [140]. Furthermore, peers tend to have a best-interest nature in this type of architecture, whereby they make the most out of the network for their own benefit whilst only providing the minimum required level of resources in return [137]. Data redundancy can help resolve these issues through both the duplication of data stored and processes performed on multiple peers within the network. *SETI@Home* provides the same data processing task to multiple users and if a user leaves the network it re-issues the task to another user, ensuring that the task is completed [141]. In situations regarding data storage, duplication of data would increase the difficulty of maintaining data integrity within the network. Updates to a specific data item would have to be reflected across all peers who store that respective data item [138].

The *SETI@Home* application still relies on a central server to collect and distribute the radio telescope data and amalgamate analysis results. In a true P2P architecture, there is no central server to organise the network's peers [136]. This means that a list of the connecting peers, together with their current processes and data storage details, cannot be stored and maintained centrally. To locate resources from an unknown location a peer is required to broadcast a query to their neighbouring peers. These peers pass on the query until the peer with the requested resource is found [138]. Only once the peer with the required resource is found can the originating peer directly communicate with them. This routing discovery process is illustrated in Figure 4.7. Broadcasting queries through this approach could create a substantial volume of traffic that is inefficient, since it can slow down the network [136]. Alternatively, each peer could hold metadata to help point to the correct locations [138]. However, this solution has the same unreliability issue as discussed earlier, as peers may join and leave the network regularly. These issues that arise from decentralised control make it difficult to incorporate data management in a P2P architecture. This includes the areas of both the peers knowing where to make data access requests and ensuring that the distributed datasets remain accurate and online within the network.



**Figure 4.7:** Peer broadcasting a query to locate another peer (adapted from [138])

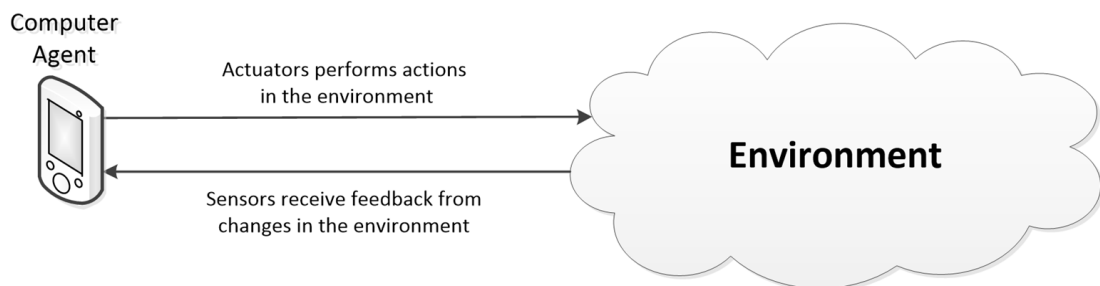
However, since no centralised server keeps a record of each peer, a P2P architecture provides a higher degree of anonymity in comparison to a client-server architecture. *Freenet* [144] is an example of a P2P application that provides anonymity amongst its peers. It is a file-sharing application, whereby the peers are able to share files with each other. Requests between peers are encrypted to help prevent determining the originator [143]. In addition, the files are encrypted locally at peer devices to provide further anonymity.

File-sharing applications have been very successful in utilising a P2P architecture. One of the most popular file-sharing applications is *BitTorrent* [145], which focuses on the storage and transfer of multimedia content amongst its users. Multimedia content, such as video files, can typically be large in size. *BitTorrent* utilises a large number of users, connected over the internet, in the exchange of these files [146]. There are three types of peers in this application, leechers, seeders and trackers [147]. Leechers make requests to download files. Each seeder acts as a server, storing a complete version of the requested file for sharing with others. Downloading a file is achieved by breaking down the file into multiple small segments, with each segment downloaded from a different seeder. A tracker contains the list of seeders that contain the requested file. As files are not distributed from a single central server, this process removes the bandwidth issues that would arise from sharing numerous large files across just one network channel. Once a leecher has downloaded the whole file it

becomes a seeder. This makes the application robust as there are constant replacements for the seeders that drop out. Additionally, data integrity is less of an issue with the sharing of multimedia content, where if errors are found in a file segment then the content can be downloaded again [148].

#### 4.4 Multi-agent architecture

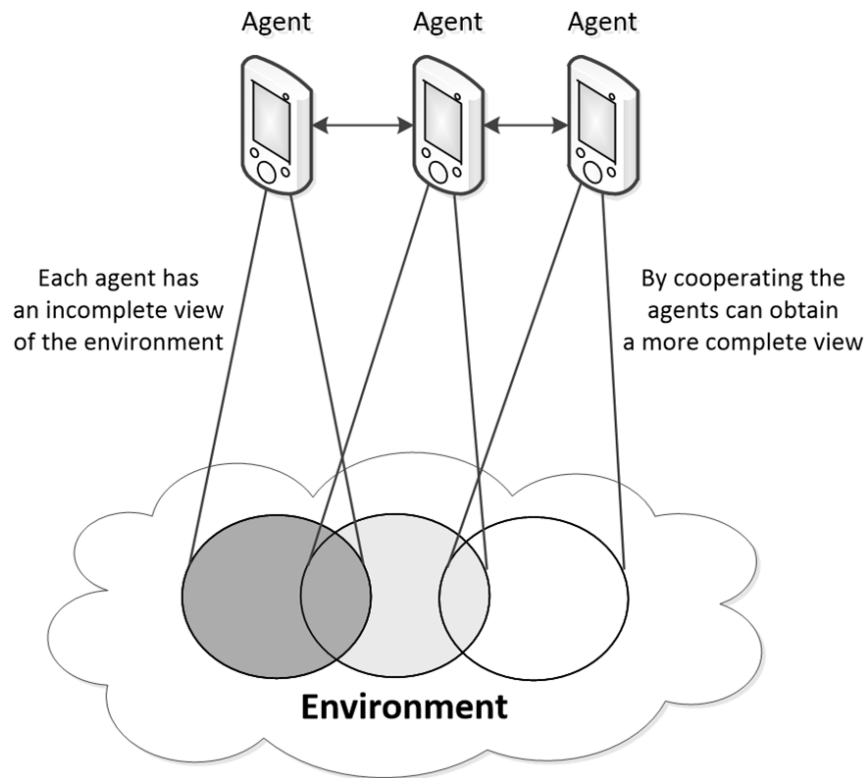
Agents are computational devices that are equipped with sensors and actuators to autonomously perceive and act on their environment [149]. Figure 4.8 illustrates an agent acting on their environment and perceiving the effects of these actions. An agent can plan, make decisions and learn in order to achieve a desired objective [150]. A multi-agent architecture consists of numerous agents that interact with each other in a dynamic environment, whereby the behaviour and actions of each agent has an effect on the environment [151]. The agents can cooperate with each other or work in parallel to achieve common objectives [152]. This is possible through communication between agents, as well as agents observing the behaviour of other agents and the changes caused to the environment. An example of a system that incorporates a multi-agent architecture is the *Intelligent Distributed Autonomous Power System (IDAPS)* [153]. The agents in this system run autonomously and cooperate together to manage the operations of electricity distribution grids. Each agent runs its operations autonomously to detect and respond to grid outages and faults.



**Figure 4.8:** Computer agent acting and perceiving on their environment (adapted from [154])

A multi-agent architecture shares a number of the advantages of a P2P architecture as both utilise computational devices that are distributed globally. A system utilising this architecture can benefit from a high quantity of computational power through the interaction and cooperation of multiple agents [155]. This architecture is also highly robust as each agent is able to share their expertise and data with other agents [156].

The agents, individually, may have an incomplete view of the environment [157]. However, by drawing on a pool of knowledge within the group of agents, it gives the opportunity for the group to reach a more complete view [158]. This cooperation between multiple agents is illustrated in Figure 4.9. Therefore, the process of interaction enables the agents to coordinate their actions and combine their resources to achieve their desired goal [159].

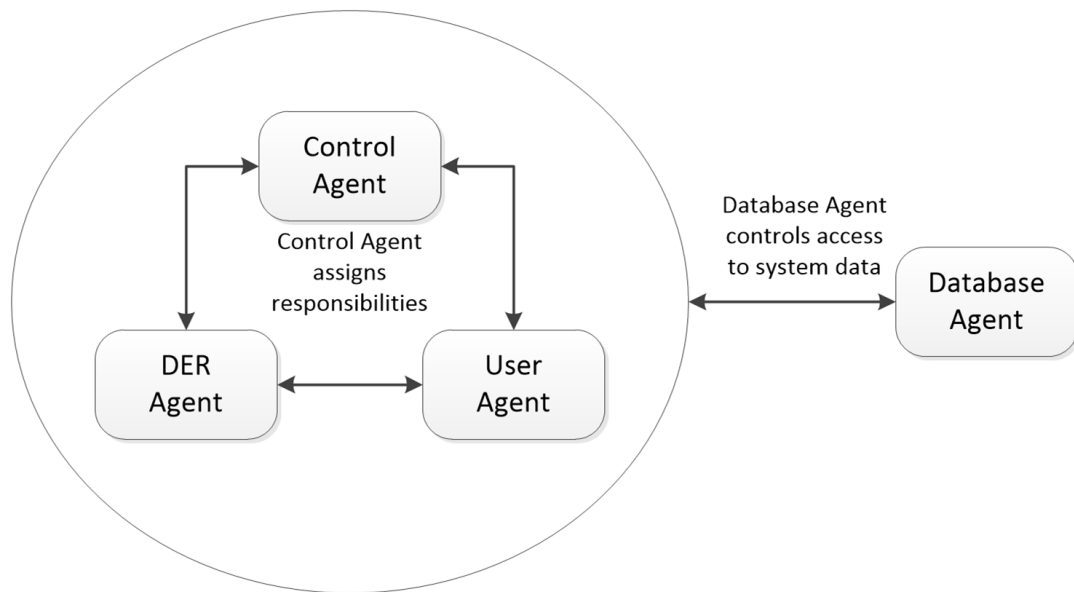


**Figure 4.9:** Cooperation between agents in a multi-agent environment (adapted from [158])

In the *IDAPS* system each agent is provided with different roles [153]. A *Control* agent assigns responsibilities to the other agent types; a *Database* agent stores and controls access to the system data; a *User* agent provides access to the grid for customers and a *DER* (Distributed Energy Resource) agent's responsibilities includes monitoring power levels. The agents only have knowledge of the areas that have been assigned to them. However, through their interactions and collaboration they can achieve together the shared goal of ensuring critical loads are secured in the event of an outage. In this way the agents are providing a level of intelligence to the grid that



would not be available without effective collaboration. Figure 4.10 illustrates the interactions between the agents in the *IDAPS* systems.



**Figure 4.10:** *IDAPS* agent interaction (adapted from [153])

In a multi-agent architecture, agents can be designed to learn new behaviours as they operate in their environment [151]. Each time an action is performed that causes a change in the environment's state, the agent would receive feedback via its sensors. This feedback can be used by the agent to help make future decisions. The agents in the *IDAPS* system could use historical data from previous faults to assist in deciding the correct action to perform when new faults occur. Therefore, the agents become more efficient at performing their actions as time progresses.

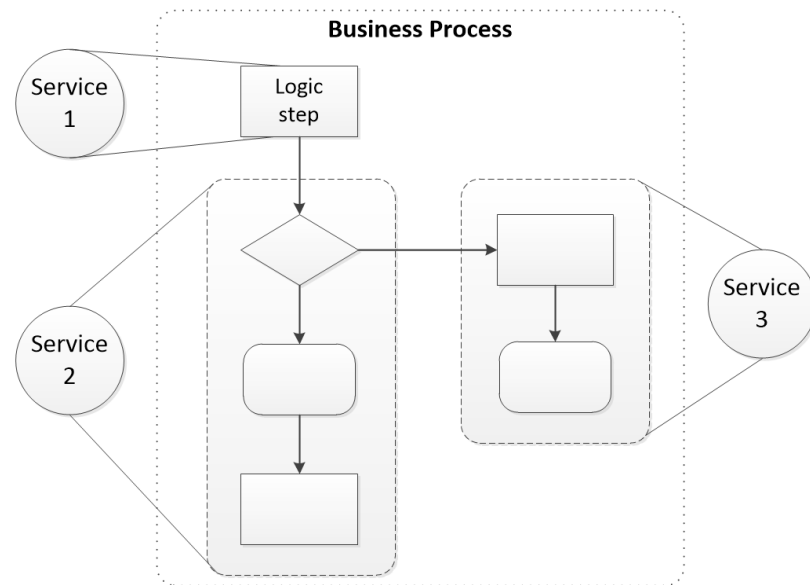
However, it is not a simple task to ensure that agents can react to their environment autonomously. An issue with implementing a multi-agent architecture in a real-world application is the difficulty of ensuring an agent can reason effectively and know the necessary requirement for each eventuality [154]. Agents also often have a self-interested nature due to their own goals which may not be in line with the collective goals [150]. This can create conflicts, preventing a successful cooperation between agents when they are running autonomously with no central controller.

Furthermore, the computational devices that act as the agents would be required to have specific components in order to perform their role adequately. For example, in the *IDAPS* system, the *Database* agent is required to have sufficient memory to store

the system data and also contain specific software in order to manage access of the data by other agents. The *DER* agent requires suitable software to monitor the grid's power levels. Therefore, this type of architecture may not be suitable in a platform that utilises mobile communication devices. As discussed in Chapter 2, mobile communication devices could range from consisting of high-end to basic components. The basic devices may not have the adequate hardware or be capable of running the required software to operate as effective computational agents in a multi-agent architecture. Therefore, a platform that utilises this type of architecture would be unable to include users of these basic devices.

## **4.5 Service-oriented architecture**

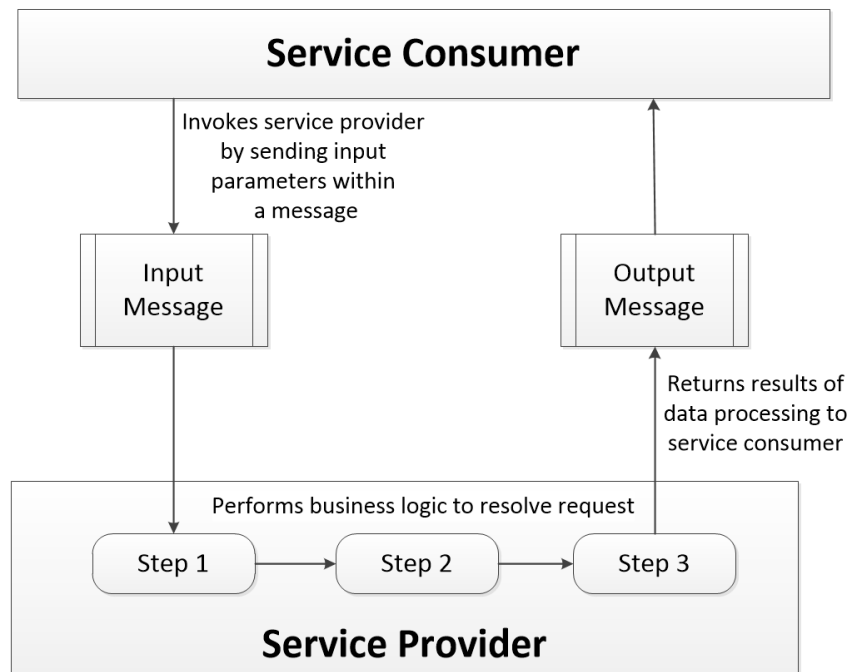
A service-oriented architecture (SOA) is an architectural design for building software systems that utilise distributed services for the running of business processes [160]. The business logic of the system is decomposed into units of varying granularity that are encapsulated by services. Each service is an independent and self-contained software entity that executes a specific and meaningful function for a calling application [161, 162], as illustrated by Figure 4.11. A service is implemented with a well-defined published interface that can be invoked by either the end-user application or another service, exposing the functions of the service to external consumption [163]. A collection of services can be invoked in a particular sequence, whereby the services would interact with each other to provide an entire business process to the calling application [164]. The service performing the requested function is referred to as a service provider and the calling service or application is referred to as the service consumer [165].



**Figure 4.11:** *Varying units of logic within the business process are encapsulated by services (adapted from [162])*

There are four general component types to a system that uses SOA principles; the services, the application front-end, a service bus and a service repository [166]. The application front-end provides the end-user with an interface to invoke the required services they need to run, together with the capabilities to view the results from the execution of the services. The service bus provides a link between the services and the application front-end. The service repository enables each service to be discovered by either the end-user or other services through storing and sharing service contracts of all the system's services [166, 167]. Each service contract contains data on the associated service to enable a service consumer to send a message and invoke the service at runtime [161].

The interface provides a description that details the purpose and behaviour of the service in an easy to understand format for all parties participating in the process [160, 165]. The interface description is available in the contract held by the service repository. This description includes a service signature containing the input parameters and the output result types, together with usage requirements and constraints of the service [164]. Therefore, services are able to establish a distinct relationship and connect with other services, which need to use them, through their interface description [162]. Services can then interact through the exchange of messages, containing the necessary input or output data. This process is illustrated in Figure 4.12.

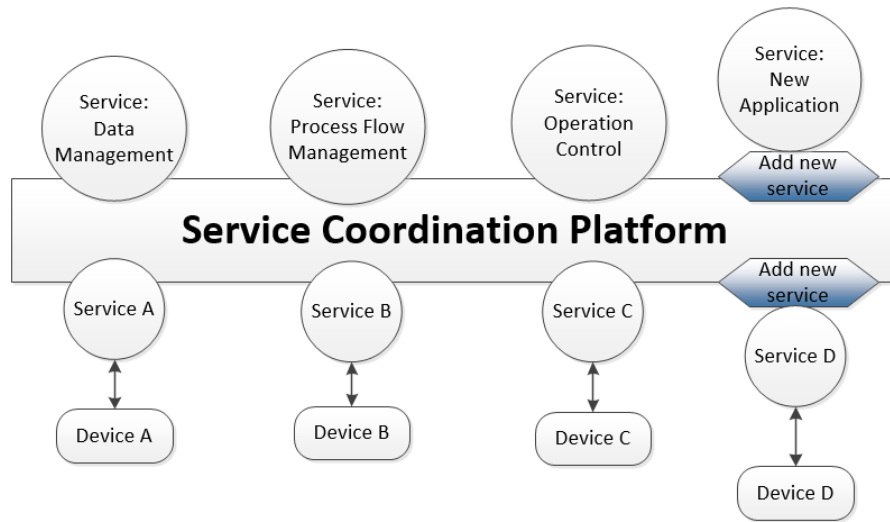


**Figure 4.12:** *Input and output messages between service consumer and service provider containing parameters and results respectively (adapted from [168])*

Each service is independent and running as a black box, whereby the procedures for performing the required function are unknown to the service consumer [164]. Therefore, in the message sent to the service provider there are no instructions on how to perform the procedures to resolve the service consumer's request [165]. This results in a loosely coupled relationship between the services of a system. A service consumer is only required to have knowledge of the name and interface of a service provider in order to invoke it [161]. Subsequently, the internal workings of a service can be modified without affecting the operations of other services or the means by which the service is invoked [164]. This results in a low complexity system in comparison to an integrated system, where changes in one module could have unexpected operational consequences in other areas of the system [169].

Loose coupling enables services to be easily added or removed from systems [164]. Existing services can then discover and establish relationships with the new services by the exchange of messages. This is demonstrated by the *Train Car Management* system [170] that employs an SOA to control the operations of a train. There is a service coordination platform that maintains the connection between all the system's services and the train crew, the users of the system, who invoke services to operate each device on the train. This architecture enables a straightforward process of

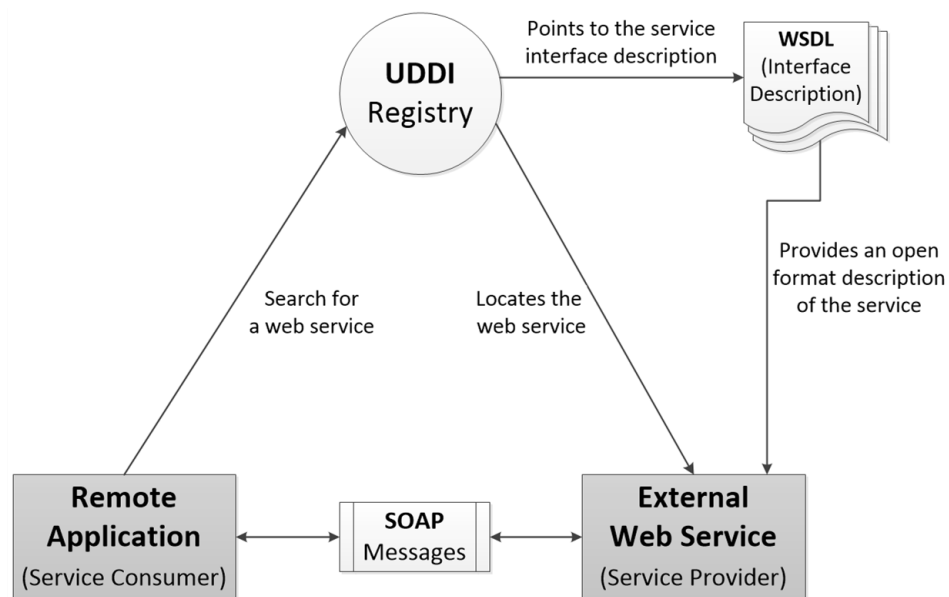
attaching new devices together with the services to operate them, as well as detaching redundant devices. The application front-end discovers the interface of the service and is then able to invoke it, allowing users to control new devices without any changes to the remaining areas of the system. This results in a system that can easily adapt to future functional requirements and facilitates for incremental development of features through the continuing interchange of services. The framework of the *Train Car Management* system is illustrated in Figure 4.13.



**Figure 4.13:** Framework of the *Train Car Management* system using SOA principles (adapted from [170])

Zimmermann et al. [168] discusses a project to upgrade the order management system of a telecommunications wholesaler to an SOA. The wholesaler provides telephone services to a wide range of companies. The order management system is required to provide two different methods of interaction, a web browser interface for small scale customers and autonomous processing and responses of web requests from the automated systems of larger scale customers. An SOA enables the re-use of services for different methods since front-end requests from either method would result in the same back-end functionality being performed [171]. In the upgrade to an SOA the order management system has deployed a *Channel Controller Layer* that acts as the connection between the methods in the front-end and service providers in the back-end of the system. Therefore, the implementation of the logic is unique within each service, where the services perform the same procedures regardless of the input method and do not have the knowledge of which method will receive their output results [168].

Services can be exposed by their interfaces over large networks to enable a wide distribution of services, where the service consumer can be a considerable geographical distance from the service provider [172]. Web services are services that can be invoked over the internet, which are simply located by a *Uniform Resource Identifier* (URI) [173]. Web services have extra constraints compared to normal services, using open internet standard for the communication between the service consumer and service provider [165, 167]. Applying open standards results in a web service being platform-independent, offering the ability for any remote web-connected application to invoke the web service [174]. There are three open standards that work in conjunction to offer web services over the internet, illustrated in Figure 4.14. *Simple Object Access Protocol* (SOAP) is the communication protocol that is used by applications to interact with a web service, defining how each message is constructed [160, 165]. *Web Services Description Language* (WSDL) is an interface standard used to define the interface description of a web service, whereby developers of a service consumer would utilise this open and machine-understandable interface format to invoke the web service [175]. Web services are registered into a *Universal Description Discovery and Integration* (UDDI) registry, which follows a universal set of rules to enable web services to be registered and retrieved in a well-defined process [176]. The UDDI registry is a service repository that enables applications to easily search for registered web services and inspect their interface descriptions [167].



**Figure 4.14:** Use of open and universal standards in web service interactions  
(adapted from [176, 177])

*Web-based Manufacturing Resource Service (WMRS)* [167] is a service discovery application that enables manufacturing companies to collaborate in the sharing of web services to provide interoperability between companies and networked product development. The web services are distributed across multiple manufacturing websites. Users of WMRS can search through their portal to locate and use the web services. These users are able to integrate the web services into their existing systems, leveraging the assets of third party business logic, as illustrated in Figure 4.15.



**Figure 4.15:** *Interoperability between different companies via their registered web services (adapted from [178])*

An example use of WMRS is in the construction of car tyre moulds, where the manufacturers can utilise web services through WMRS to integrate external resources. These services include software for computer aided engineering analysis, process design and workflow monitoring. The services can then be configured to meet the needs of the consumer's system. The wide distribution of external web services provides the consumer, in this example a manufacturer, with a high degree of flexibility in extending their system to meet its requirements, where it is possible to invoke a service regardless of location. This increases the agility of the system's framework through having the technology at their disposal to respond to new requirements rapidly [169].

The infrastructure of WMRS, together with the open and universal standards that define web services, results in the opportunity for services to be re-used by multiple separate entities for different situations [171]. The benefits from the re-use of web services include decreased development efforts and costs due to a reduced amount of business logic being coded internally [179]. Furthermore, utilising third party services helps to reduce the complexity of a company's own product, resulting in less maintenance required after the product's release.

There are drawbacks to implementing a system with an SOA. A distributed system that relies on services over a wide geographical network would have a higher latency in the communication compared to an integrated and centralised system [180]. This is due to the increased time taken to transfer packets of input and output data between the locations of the service consumer and the service provider. A local function call is considerably faster, taking a few nanoseconds to reach the required function in the code library [181]. Applications that implement a graphical user interface (GUI) would not be suited to high latency in the communication process with an external service provider. GUIs tend to demand high levels of data exchange, whilst also requiring rapid responses to ensure the user perceives instant responses from their actions [182].

The web services that are offered to systems, such as the manufacturing services for the WMRS application, would need to manage concurrent invocations from multiple different service consumers [180]. This can be difficult to manage in a large distributed environment where numerous requests may pile up, causing the connection to the service provider to reach its limits [181]. This could delay or prevent subsequent request messages from being received by the service provider. The service provider must also ensure that each request is handled in an appropriate order to prevent conflicts, particularly in the cases where a service consumer is sending more than one request or multiple consumers are collaborating, for example in networked product development. Therefore, the service providers need to be designed to handle concurrent requests sufficiently, where time delays are limited and conflicts in data are prevented.

Reliability can be an issue for a distributed system since there can be a large number of potential points of failure. Each service, router and other connected devices within the network may fail, thereby disrupting the entire running of the system [181]. Furthermore, it can be difficult to ascertain the reason behind the failure of a system process, whether it is the fault of a remote service provider or the network connection. A software bug may negatively impact both the service consumer and service provider. Dropped network connections would interrupt the communication process between the two parties. The components of a system need to be developed in a way



that the points of failure would be correctly handled at runtime to ensure the system can respond appropriately and continue running [180].

## 4.6 Conclusions

The system architectures discussed in this chapter each provide desirable attributes depending on the situation that they are applied to. In a client-server architecture the central server controls and organises the components of the system. This architecture is beneficial in organising the coordination of a group of client devices, as illustrated by the *UbiquitousSurvey* system [13]. The clients operating in the field can be provided with different tasks, whereby the field data that each of them collects is sent to be processed in a central location. Additionally, data management is handled from this central location. This is illustrated by the functionality of the *HELP* system [101], whereby users input data into client computers which then send the data to the server for storage.

A P2P architecture utilises the free processing and storage resources of connected user devices to accomplish tasks. This architecture has been shown to be effective in applications that focus on content sharing, especially that of large multimedia files. The *BitTorrent* [145] application enables peers to download small segments of these files from multiple seeders simultaneously, thus dividing the work effort amongst the connected devices. However, it is harder to achieve coordination between peers compared to a client-server architecture since there is no central coordinator in place.

A multi-agent architecture focuses on either having agents working together or reacting to the actions of other agents in a shared environment, thereby making this architecture beneficial for collaboration between devices. In the *IDAPS* system [153] each agent is provided with different responsibilities, where the agents work together to achieve their shared objective. Furthermore, agents can learn from the results of previous actions in their dynamic environment. However, as agents are running autonomously there is difficulty in ensuring that agents react correctly to the information they receive and have successful interactions with other agents.

Both these decentralised architectures require the participating devices to consist of adequate hardware to perform their processing tasks. This can be problematic when utilising mobile communication devices, such as those listed in Table 3.1 of Chapter

3. A basic mobile phone may not have the ability to perform tasks other than that of sending and receiving communicated data. Additionally, peers can join or leave a network whenever they desire.

An SOA enables systems to utilise independent and autonomous services to perform their business logic. The loosely coupled relationship between different services in a system allows new services to be easily added and old services removed, with minimal change to the rest of the system. The open standards that define web services offer a straightforward method for systems to leverage existing services from external third parties, thereby reducing the development effort of the system. However, the distributed nature of an SOA can result in performance and reliability issues. High latency in the communication between services increases the time taken to perform business processes. Each server running a service, as well as the network connections between the servers, are potential points of failure that can disrupt the running of a system.

Further to the discussion of developing a new platform at the conclusion of Chapter 3, this chapter has shown that a client-server architecture would be a suitable framework for developing the platform as it supports the list of features in Table 3.2. The only difficulty arising would be that of user anonymity. This feature would need to be factored into the development of the platform to enable the inclusion of potential clients who wish to remain anonymous. However, a central location for processing and data management is ideal to ensure the other elements are correctly handled. Central coordination ensures that each client is provided with the correct tasks, enabling objectives to be met by each user.

Designing the platform based on a client-server architecture would take the processing and data storage requirements away from the user, thereby removing the barriers to entry for client devices to join the platform. Client devices are not required to have specific hardware to assist in these areas. The *Remote Experimentation* system [15] illustrates how a client-server architecture is effective in this type of situation. The server handles the functionality to control the experiments. Client devices only require an interface to interact with the server.

Furthermore, an SOA facilitates a different way of looking at the composition of a system's components, whereby its principles can be applied to systems that are based on an alternative architecture, such as the client-server architecture. The platform should apply particular principles of an SOA. A loosely coupled relationship between the different components of the platform would facilitate a rapid integration of new components as further requirements are identified. External web services could be utilised for various aspects of the platform, including the use of SMS web services that were described in Chapter 2. The platform should be designed with unique and distinct business logic in each component, whereby there is minimal duplication of code, in order to reduce the complexity and simplify the maintenance required.

## 5 Design and Implementation

### 5.1 Introduction

The previous chapters have identified requirements for the development of a new integrated communications platform that can support both interaction via multiple mobile communication methods and the integration of a range of different scenario applications. A client-server architecture has been selected as this enables the platform to effectively interact with users in a mobile environment.

In order to be able to operate a range of scenario applications the proposed platform needs to be developed with a framework of generic components that support the feature-set which was identified in Table 3.2 (Chapter 3). This includes the need for a communication component that supports two-way communication between multiple users and the platform. The communication component is required to support the exchange of pictorial content and have the capability for users to send messages anonymously. Messages that are sent by either the platform or users are to be based on a template layout. This is to provide a content structure for each message that can then be individualised for the situation at hand by adding relevant field data to the template.

The platform should employ a versatile analysis component that can efficiently process and react to received messages autonomously and in real-time. A database is required to handle the storage of the communicated data and record the responses performed by the platform. The analysis component and database should collaborate to provide functionality for sending alerts and reminders to users based on previously received and processed information. The platform needs to incorporate assignment allocation procedures to handle scenario applications that require multiple users operating concurrently, in order to coordinate the users effectively. Location-based functionality needs to be implemented into the platform, whereby up-to-date location data can be recorded on each user to be used for assignment allocations based on their location.

This chapter discusses the design and construction of the new platform, labelled the *Connected-Mobile Platform*, which is developed based on the requirements that are detailed above. The first section discusses the high level design elements of the

platform's framework. This includes the components of the client-server architecture, the different methods of communication available to the platform and the implementation of the features described in Table 3.2.

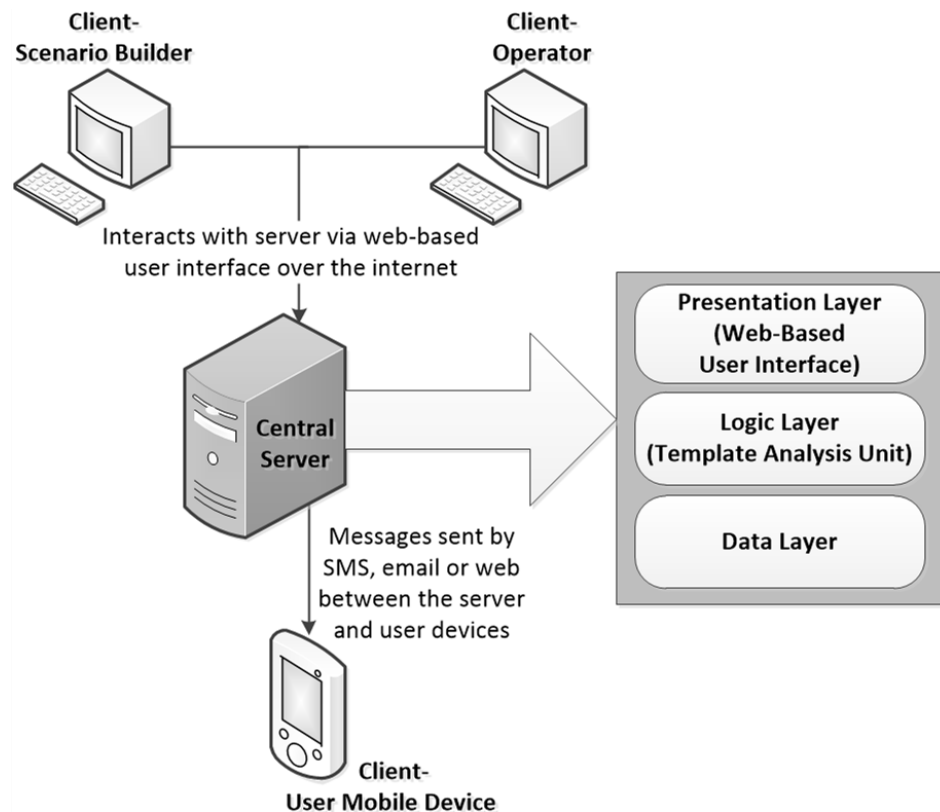
The next section describes the procedure to analyse and respond to user communication. The platform is designed to have a high level of autonomy, whereby the server has the ability to automatically communicate with each user as well as process information received without the need for human intervention, by an operator.

The process of integrating new applications is discussed. The platform has a framework of generic components to facilitate the development of a range of different types of application problems in a structured process. Each application is labelled as a scenario and the platform is designed to support a range of scenarios. The implementation of the platform and its experimental setup for running case studies of different scenarios are then described in detail.

## **5.2 Design of the *Connected-Mobile Platform***

### **5.2.1 The platform's client-server architecture**

The platform's design is based on a client-server architecture, which is illustrated in the overview layout of the platform in Figure 5.1. The central server is a fixed computing device that houses the core application of the platform and the databases for data storage. The core application handles the flow of communication between the server and its clients, as well as the analysis of the data collected within the received messages.



**Figure 5.1:** Overview of the platform's client-server architecture

There are three types of clients within the architecture; a scenario builder, an operator and users of mobile communication devices. The scenario builder's role is to create each scenario and define their analysis criteria. This is carried out prior to a scenario's initialisation. The purpose of the operator is to supervise each scenario whilst it is active, administering to issues that arise. Each user in the field is a client with their mobile communication device being a client device utilised to interact with the server.

As identified in Chapter 4, applying a client-server architecture to the design of this platform enables the server to solely handle the intensive processing tasks [99]. The only requirement of a user's mobile communication device is to send messages containing field data and to receive messages containing instructions or other relevant information. Therefore, no additional hardware or software is required to be installed on their device. For each scenario all communication is controlled from the server, sending commands and updates out to users. This means the management and coordination of users can be handled from one location [13]. The anytime, anywhere characteristics of mobile phones mean that the location of the server is irrelevant, as

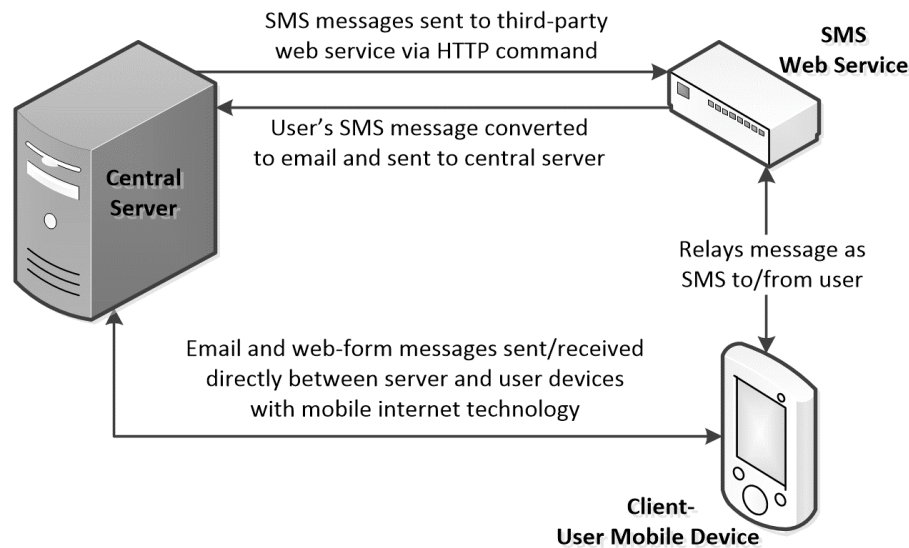
long as there is a stable connection to the internet [132]. Data storage is also controlled from this central location. The data sent from a user's mobile device is retrieved and then stored on the database. The client device is not required to store any data once it has been sent to the server. As large quantities of data may need to be stored, only the server would require maintenance and changes to handle storing this data [134].

The platform is split into three separate layers; a presentation layer, a logic layer and a data layer, as illustrated in Figure 5.1. These three layers were decided based on the investigation of the *Remote Experimentation* [15] and *Help* systems [101] in Chapters 3 and 4. The three layers have been implemented to provide a structure for the communication between the server and its clients. The presentation layer, discussed in further detail in Section 5.2.7, houses the web-based user interface. This is where the operator and scenario builder can interact with the server to manipulate scenarios. The logic layer houses the *Template Analysis Unit* (TAU) where the processing and analysis of field data, received from users, is carried out. The logic layer controls the collection and sorting of the collected data and autonomous actions in response. The data analysis process performed in the logic layer is presented in detail in Section 5.3. The data layer, described in Section 5.2.10, is composed of a set of tools to effectively store, retrieve and update data within the platform's databases. There is a database, labelled as the *Principal* database, to store data associated with the core entities of the platform; including scenarios, messages and users of the platform. For each scenario there is an additional database to store data, which is only associated with that individual scenario.

### **5.2.2 Integrated architecture to support multiple communication methods**

Each user's mobile communication device interfaces with the server by sending or receiving messages wirelessly. The server utilises communication methods available over the mobile networks and the internet to communicate with users' mobile devices. Users can send messages either via SMS, email or using a web-form within the web-based user interface. These three communication methods have been chosen for the platform due to each method having distinct benefits that have been highlighted in Chapters 2 and 3. SMS is a ubiquitous mobile communication method that is supported by all mobile phones and therefore enables the platform to be

inclusive to all types of users. Email and web-based means of communication are also supported as they offer further features in the communication process including the exchange of pictorial content and the support of new types of portable devices, such as tablet computers. Enabling a user to choose between these three communication methods allows the platform to interact with a wide range of mobile communication devices. Therefore, the preferred communication method is chosen by the user based on their own preferences and the capabilities of their mobile device. Users are able to communicate with the server from any location where a network reception or wireless internet connection is available. Figure 5.2 illustrates the communication methods that allow a user to interact with the server.



**Figure 5.2:** Interactions between the server and a user

For SMS messaging the user's device does not require specialised components as virtually all current mobile phones support communication via SMS. Therefore, any potential user owning a mobile phone would be able to interact with the server via this method. For this reason SMS has been chosen as the primary communication method within the platform. SMS communication is achieved indirectly via *TextLocal* [42], a third party SMS web service. The web service relays messages to and from the server and its users. The server sends the message as an HTTP command to the web service, which forwards the contained data as an SMS message to the user. SMS messages sent from the user are relayed by the web service as an email message and sent to the server. *Microsoft Outlook*, an email client, has been integrated into the



platform to receive email messages from the web service. These messages are stored in the *Outlook Inbox* folder until the server retrieves them for processing.

Communication via email is accessible to users with internet-enabled devices, thereby benefiting from the added features of this method, as discussed in Chapter 2. This includes sending messages at no cost and the ability to attach multimedia files to a message. The server can forward these multimedia files to other users whose devices also have email capabilities, enabling the exchange of pictorial information between the server and its users. Communication via email is achieved directly between the server and the user via *Simple Mail Transfer Protocol* (SMTP) commands over the internet. Email messages sent from a user are also collected and stored in the *Inbox* folder.

Chapter 4 discussed the benefits of leveraging external services provided by third party companies in a service-oriented architecture. Although the platform has been designed based on a client-server architecture, both *TextLocal* and *Microsoft Outlook* are external services utilised by the platform. These services are employed to reduce the time and effort in developing the platform since additional components would otherwise have to be constructed to perform the same functionality.

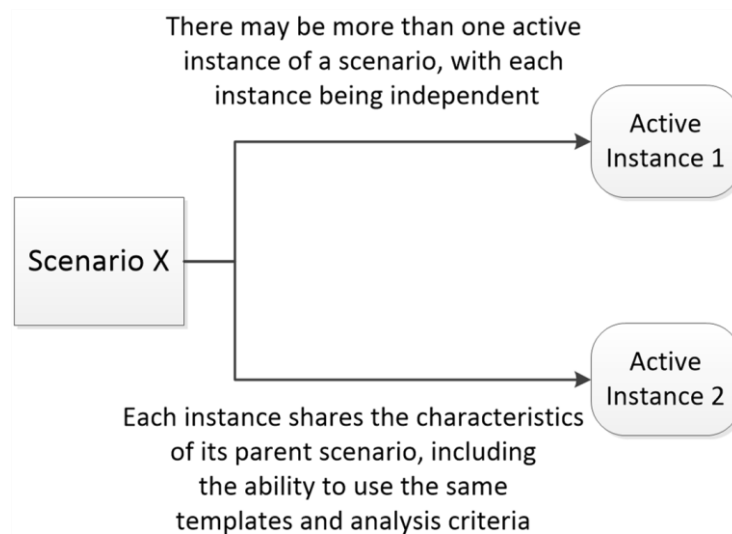
The users of smartphones and tablets can utilise the web-based user interface to send and receive messages. Each user is provided with a unique *PersonID* attribute to access the website. The user then has the ability to send a message from the *UserMessageSend* web-form and view previous messages, both sent and received, from the *UserMessageView* web-form. User messages sent via this method are instantly received by the server, bypassing the need for the server to interface with an SMS web service or an email client. Messages that initiate from the server are created and stored in the *Principal* database. When the user requests to view the message the web-based user interface collects and displays the message data from the *Principal* database.

Message interfacing between the server and its users consists of two stages: output and input. Output consists of messages sent from the server to users. This enables the server to provide new instructions or tasks to a user, respond with feedback on an earlier message or provide an update concerning a situation. Output can either be

achieved autonomously via the TAU or manually by the operator. Input to the server consists of messages sent by the user after a task is fulfilled with the data gathered in the field. Alternatively, the input message could be a request for further information or instructions. The server has the ability to receive numerous messages simultaneously, whereby each message would then be processed individually. Additionally, messages can be sent from the server to multiple users simultaneously to enable multiple users to interface concurrently with the platform.

### 5.2.3 Generic framework to manage multiple scenarios

The platform's architecture has been designed based on a generic framework to facilitate a smooth integration process with different types of scenarios. The platform's core functionality covers the features required for the running of each scenario. This includes the ability to send messages to users and algorithms for handling and processing received messages from users. The platform contains modules for the management and storage of the various shared entities and their attributes. These entities include the scenarios, users and messages exchanged between the server and its users. Each occurrence of a scenario is defined as an instance. When a scenario is requested to be run for a particular problem, an instance of the scenario is created. Each instance relates to a unique problem, as illustrated in Figure 5.3. It is possible for a scenario to have many separate instances running concurrently.



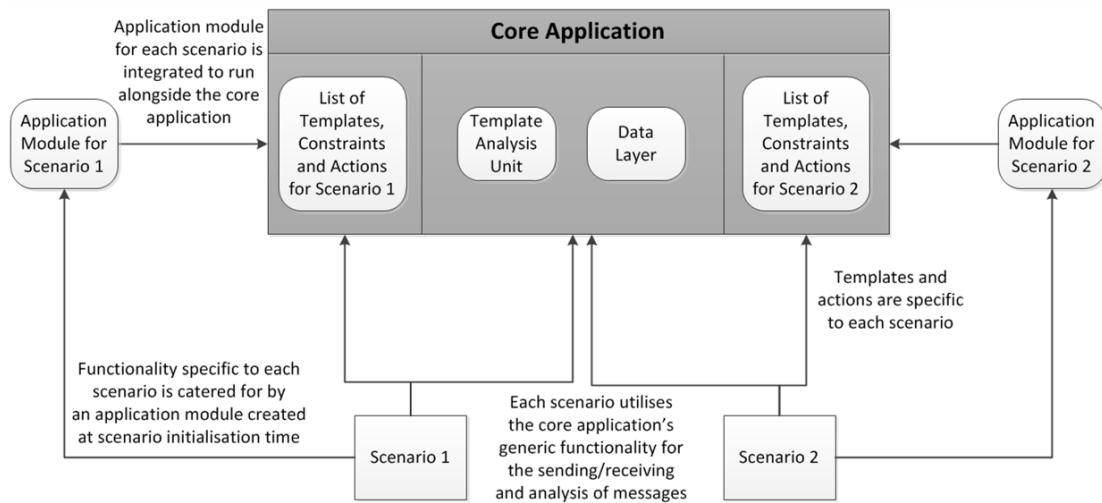
**Figure 5.3:** A scenario with two active instances

The platform provides tailored functionality for each scenario in order that the specialised characteristics of a scenario can be supported, which is achieved via a custom-made template mechanism. A template contains a pre-defined layout of text which is used to create a particular message. Its purpose is to provide a structure for the information exchanged between the server and its users. Each message is created, for both input and output, based on a pre-defined template where each template is exclusively associated to one scenario. However, there can be numerous templates created for a scenario, enabling the communication structure of each scenario to be unique and specifically devised to meet the scenario's objectives. A template has associated rules, constraints and response actions to aid the platform's analysis process and enable instances of the associated scenario to run autonomously.

Each scenario may have dissimilar attributes and characteristics due to their differing purposes and objectives. Therefore, the core functionality may not be sufficient for some aspects of a scenario. To handle data heterogeneity between scenarios, the platform has been developed to provide each scenario with its own unique database to store the field data gathered by users. Field data is only relevant for the parent scenario of the instance it concerns. Therefore, providing a database specifically designed for each scenario means that the data items can be stored in appropriate locations within the database. Additionally, each scenario has a bespoke application module integrated into the platform's logic layer to run data analysis that is specific to the scenario. This application module has the ability to retrieve and manipulate the gathered field data in the associated scenario's database.

This design enables the platform to support a range of scenarios at any given time, as well as being easily expandable to cover the unique functionality required for new scenarios as they arise as only minimal changes are needed for the main elements of the platform. A scenario could focus on its users individually with a separate instance for each user and no association between users. This type of scenario would have a heavy emphasis on one-to-one communication. A different scenario may require multiple users per instance, with the coordination of users from the central location of the server necessary to fulfil tasks. This type of scenario could also incorporate broadcast communication to provide updates to groups of users as an instance progresses. The platform is designed to handle each type of scenario with the same

underlying communication and data analysis structure. Figure 5.4 illustrates the expandable structure of the platform's generic framework.



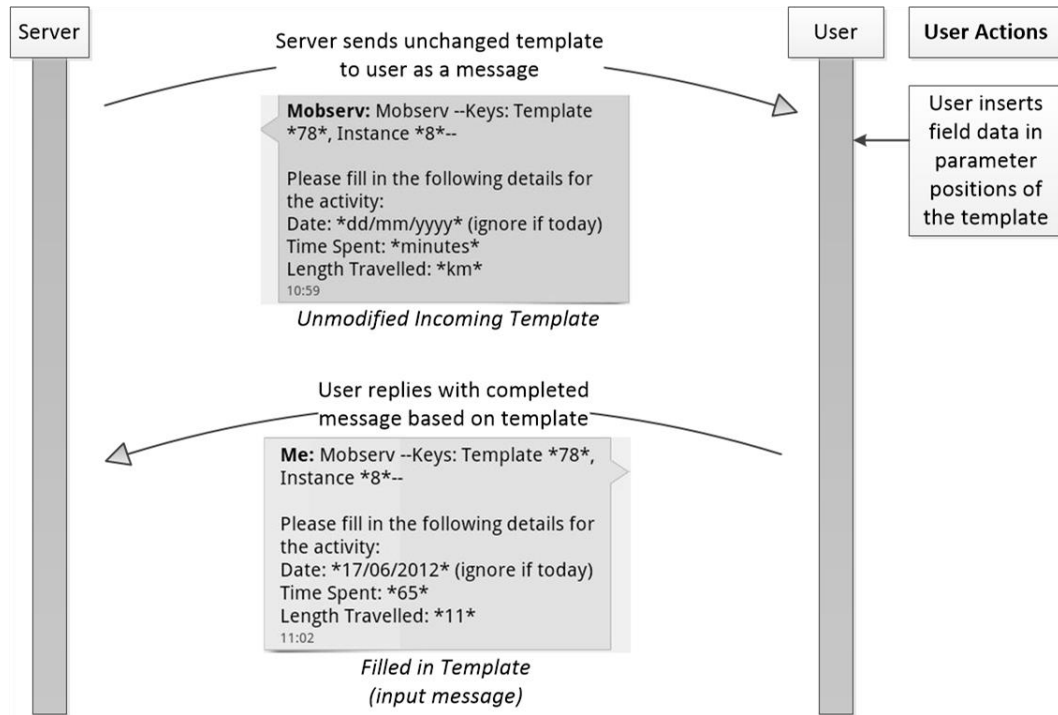
**Figure 5.4:** The platform's generic and expandable framework

#### 5.2.4 Implementation of templates for autonomous operations

Pre-defined *Outgoing* templates are used by the server to autonomously create and send messages to users. *Incoming* templates are designed for users' responses to the server, allowing users to provide instance updates or make new requests. *Outgoing* and *Incoming* templates are designed to work with each other to facilitate the flow of communication. Output messages based on *Outgoing* templates are sent by the server as a response to analysis performed on the data of a received input message. The *Outgoing* templates can then be associated with a new *Incoming* template. This *Incoming* template would be supplied to the user alongside the output message to assist their response. To create a new input message to be sent to the server, the user would only be required to insert field data into the supplied *Incoming* template.

A parameter is a pre-defined position in a template that has been created to facilitate the input of field data. An *Incoming* template can have multiple parameters defined within its textual structure, with each parameter uniquely defined by its position in the template. When the server provides the relevant *Incoming* template to the user it is sent unchanged as a text message. The user's role is to insert field data into the parameter positions of the supplied *Incoming* template, transforming a template text into an informative message concerning the matter at hand. The user would then send this new message to the server.

The predefined format of *Incoming* templates enables the TAU to extract the field data once the input message is received, providing the method for autonomous analysis and processing of each message. This communication process is illustrated in Figure 5.5. A *Parameter* entity has attributes for the position number within the template and the expected data type of the inserted field data. These attributes are designed to assist the TAU in analysing the field data.



**Figure 5.5:** Server and user exchange of communication via templates

Parameters are also available for *Outgoing* templates. Autonomous functionality in the TAU generates new output messages from an *Outgoing* template. Data from the *Principal* database or a scenario's database is inserted into each specified parameter position.

For the purpose of SMS and email messages, within the template text of an unmodified *Incoming* template, each parameter position consists of temporary data enclosed between two asterisks that separate the parameter from the rest of the message. Temporary data is provided in the parameter positions of *Incoming* templates to assist the user in supplying the correct field data. This temporary data could be a list of data items, whereby the user is instructed to retain only the required item and delete the remaining items. Alternatively, the temporary data could specify

the name, description or format of the field data that is required to be inserted. For example, if a date is requested then the supplied temporary information would be “*dd/mm/yyyy*” to guide the user to insert the data in the correct format. The user can then replace the temporary data, enclosed between the two asterisks, with the requested field data. Using asterisks to contain data items enables the analysis process to identify where a data item commences and finishes within a message to easily extract it. The unmodified *Incoming* template in Figure 5.5 contains temporary data in the parameter positions.

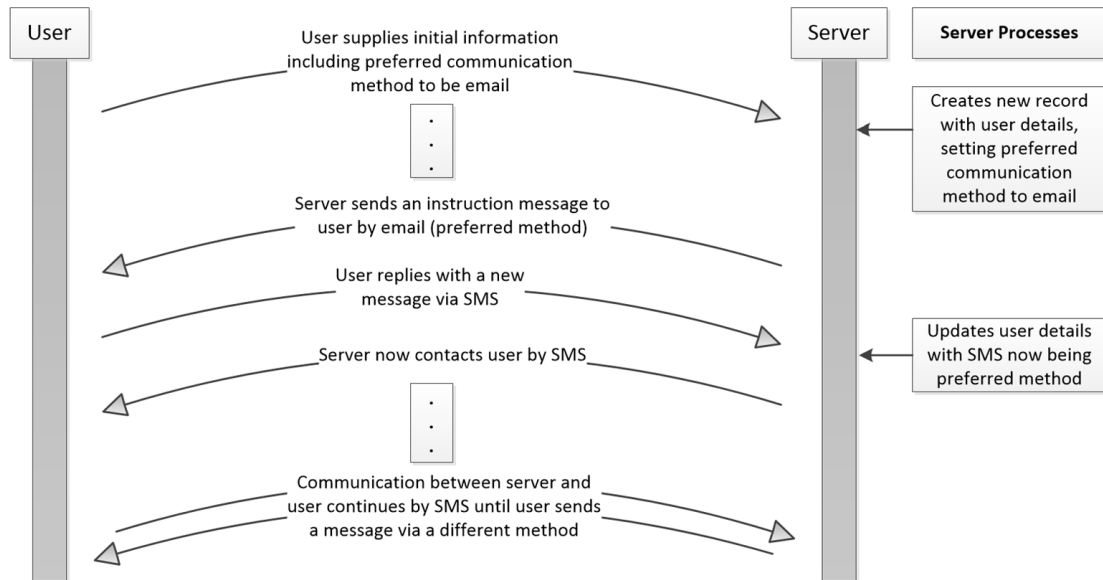
A *Request* template is a further type of template available for communication. This type of template is a fixed template implemented within the platform independent of any scenario. A *Request* template provides a method for users to request updates that are unrelated to a particular scenario or make general requests concerning the entities in the *Principal* database. There are various *Request* templates available for the user to either request a list of scenarios, a list of templates associated with a scenario or the actual text of a specific *Incoming* template. In addition, users can provide updates to their personal details via a *Request* template.

### **5.2.5 User preferences**

Each user is represented by a *Person* entity with their attributes stored in a *Person* record within the *Principal* database. This record includes details of the user’s mobile number and email address for contacting the user via SMS and email respectively. The record contains a unique identifier attribute, the *PersonID*, which is also the user’s login identifier for accessing the web-based user interface to communicate via its web-forms. There is a location attribute for each user in their *Person* record, utilised by scenarios that have location-dependent characteristics. Details of a user’s location can be regularly updated by the use of the “Member Update” *Request* template.

There are some user preferences stored in the *Person* record. Each user can choose their preferred method of communication. This preference is automatically updated whenever the user changes the method used to interact with the server. For example, if the server emails an instruction to the user and the user responds with an SMS message this would then result in future communication being sent via SMS, until the user changes to another method. This feature, illustrated in Figure 5.6, provides

flexibility as the user can simply switch between the available communication methods and the server will match the user's choice, maintaining a continuous flow of information exchange.



**Figure 5.6:** Communication process for changing preferred method from email to SMS

The only exception is when multimedia files are attached to a message. Under these circumstances when the user's preference is SMS the server would send the message as an email to include the multimedia file since this is not possible via SMS. However, the user will be notified by SMS regarding this occurrence. Future correspondences would depend on the user's method of response to this communication. Users who have their preferred method set to SMS are able to send multimedia files via email without altering this preference.

When a user initially joins the platform it is possible for them to choose the amount of personal data that is stored relating to them. This is to enable the user to remain anonymous, if so desired. Figure 5.7a shows the attributes stored in the *Person* table of the *Principal* database. If a user does not request to remain anonymous then all the attributes of this table are utilised to store information on the user. However, it would still be the user's choice to decide whether or not they would wish to share their location.

Alternatively, the user can select between two levels of anonymity. The first level is *Part Anonymity*, whereby the user only supplies the server with their contact details. In this case the user's name and any other details are not requested by the server. However, the user is still recorded as an individual record in the *Principal* database, which contains their email address and/or mobile number. This option allows the server to continue a conversation with the user, whilst their personal details remain anonymous. Figure 5.7b shows the attributes of the *Person* table which are used in this situation.

The second level is *Full Anonymity*, whereby the server does not keep any information on the user. In this case if a user submits field data concerning an instance of a scenario, only the field data they have supplied for that particular instance is stored. This would be linked to a default record in the *Person* table, which is utilised for all users who request to remain fully anonymous (Figure 5.7c). This feature allows for greater flexibility on the user-side since the user can easily choose the amount of information the platform stores regarding them, whilst still being able to participate in scenarios.

Person Table (Complete)	Person Table (Part Anonymity)	Person Table (Full Anonymity)
PersonID Name Email Number Location PreferredContact Type	PersonID Email/Number Type (set to PartAnonymity)	PersonID Name (set to Anonymous) Type (set to FullAnonymity)

**Figure 5.7:** *Person table- a: (left-side) Record with all user's attributes; b: (centre) Record for Part Anonymity with only the user's contact details; c: (right-side) Default record for Full Anonymity with no personal details of a user stored*

### 5.2.6 Assignments for collective intelligence

*Assignment* entities enable the platform to keep track of the current task a user is performing for an instance. Each task supplied to a user is considered as an assignment. There are two types of assignments; standard and location-dependent assignments. Each assignment that is available to a scenario is stored as a record in

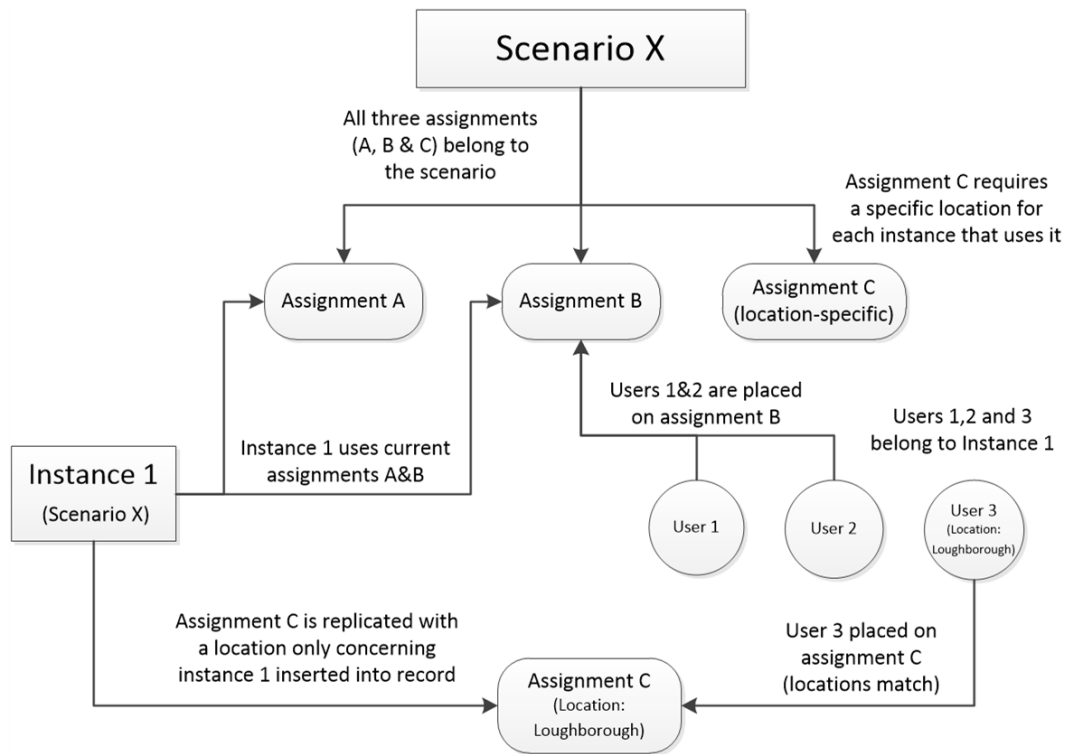


the *Assignment* table of the *Principal* database. An *Assignment* record holds attributes for the name, description and, if required, the location of an assignment.

A location-dependent assignment relies on the *Location* attribute to select users close to the specified location to be allocated to the assignment. The location of each user may need to be regularly updated in order to effectively provide them with assignments in the correct area. Users who do not share their location will not be able to participate in location-dependent assignments.

Each assignment is associated with a scenario, where the scenario may have many assignments linked to it. An instance uses the standard assignments of its parent scenario. However, location-dependent assignments are replicated when needed by an instance. This allows the location data to be added to the new assignment, which is then linked only to the instance that created it.

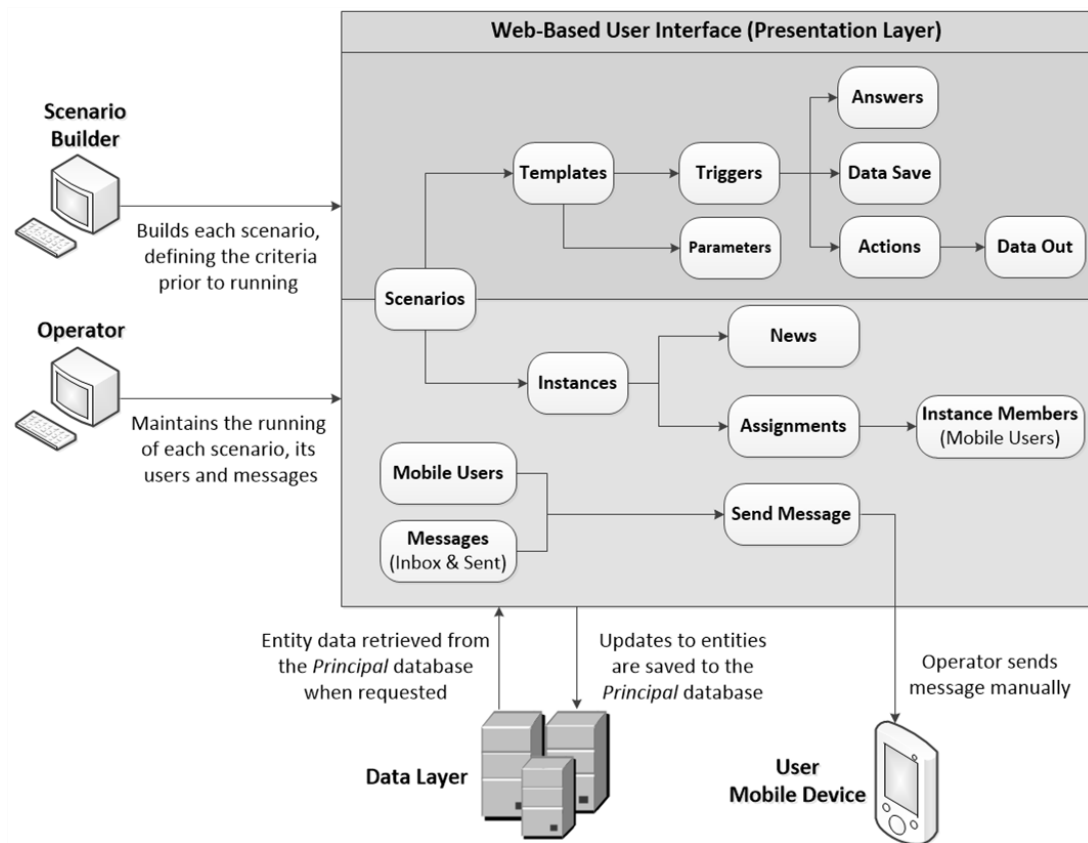
The key reason for implementing assignments is to facilitate coordination between users when there is a range of different tasks available for an instance. The purpose of this feature is to utilise the collective intelligence of the user group through effective coordination of the users in order to take advantage of their combined skill and abilities in tackling a shared issue. Each user can contribute field data regarding different aspects of the instance. This offers the opportunity for other users to be updated with new relevant information and the provision of tasks to the most suitable users. The platform can keep track of the tasks performed and currently being carried out by each user for a particular instance. Having the option of location-dependent assignments enables the provision of assignments to users in an efficient manner, based on their close proximity to the issues of the assignment. Therefore, the platform can take advantage of the number of users available for working on the issues of an instance as well as the different locations of each user. Figure 5.8 shows the structure of how a scenario and its instances utilise assignments.



**Figure 5.8:** Provision of assignments in scenarios and instances

### 5.2.7 Roles of the operator and scenario builder

Figure 5.9 illustrates the various entity modules that the operator and scenario builder have control over via the web-based user interface. This interface is accessible from standard web browsers making access possible from any modern computer with an internet connection. The role of the operator is to maintain each scenario implemented on the platform and supervise the progress of active instances. *News* records enable the operator to follow the progress of an instance from initialisation to its current status, providing them with up-to-date information on the instance. These *News* records, created by the TAU, become available as users provide field data on an instance, which are then stored in the *Principal* database.



**Figure 5.9:** The web-based user interface is split into two parts, providing access for the operator and scenario builder

The operator can manage the users assigned to an instance by providing them with assignments, updating their details or removing them from the instance. The operator has the ability to keep track of the progress of each user's assignment. This enables the operator to identify any potential issues that may arise and resolve them by intervening with the instance. For example, the operator can manually send a message to a user or update the user's requirements for a specific assignment. The web-based user interface provides the facility for the operator to view the dialogue of conversation between the server and each user, as well as for sending further messages.

Although the platform is designed to be autonomous in the communication process, the operator may be required to manually resolve issues that arise as errors could occur in the analysis of extracted message data. Some errors can be handled by the TAU and others would result in sending a notification to the user, requesting them to resend the data correctly. However, a template's criteria can also be to place an alert

value on a *Message* record if there are errors in the extracted message data. This provides a way of notifying the operator to investigate the matter further.

Prior to the initialisation of a scenario the builder defines the necessary procedures and criteria for the scenario's unique autonomous operations. As the flow of information is controlled by the use of templates, it is the builder's responsibility to create each *Incoming* and *Outgoing* template. Furthermore, the builder constructs appropriate rules and constraints for each template. *Trigger* and *Answer* entities define these rules and constraints. The TAU uses these entities to analyse the field data within an input message that is based on an *Incoming* template. The builder also defines each action that the TAU can perform, based on the analysis decisions made. For example, an action could be to instruct the TAU to reply to a user using a specific *Outgoing* template. These entities need to be created by the builder and tailored to the requirements of a scenario in order for the scenario to be active and ready for use in the platform.

#### **5.2.8 Message entity**

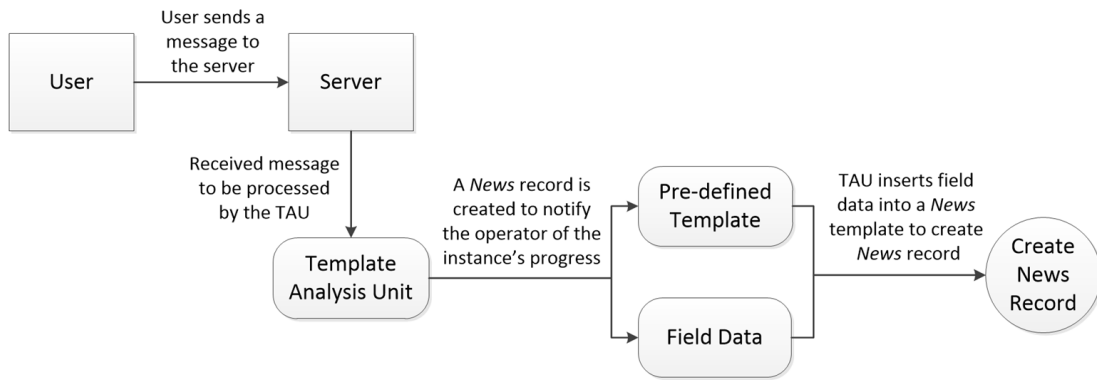
Each message is stored as a record in the *Message* table of the *Principal* database. Attributes of a message stored in this record include the sender, title and text of the message. The title contains key information regarding the message, such as the template and instance it refers to. The rest of the message contains the main text constructed from a specified template with inserted field data. If a multimedia file is attached to the message then this file is saved to the server. The *Message* record stores the filename and the location of the folder in which the attachment has been saved. Each *Message* record can have various alert statuses to illustrate their importance. The purpose of these indicators is to notify the operator of an issue if it is required to be manually resolved. Figure 5.10 shows the *Message* table containing each of these attributes.

Message Table
MessageID
TemplateID
InstanceID
PendingID
Title
Text
DateSent
Alert
AttachmentLocation

**Figure 5.10:** Message table and its attributes

### 5.2.9 News entity

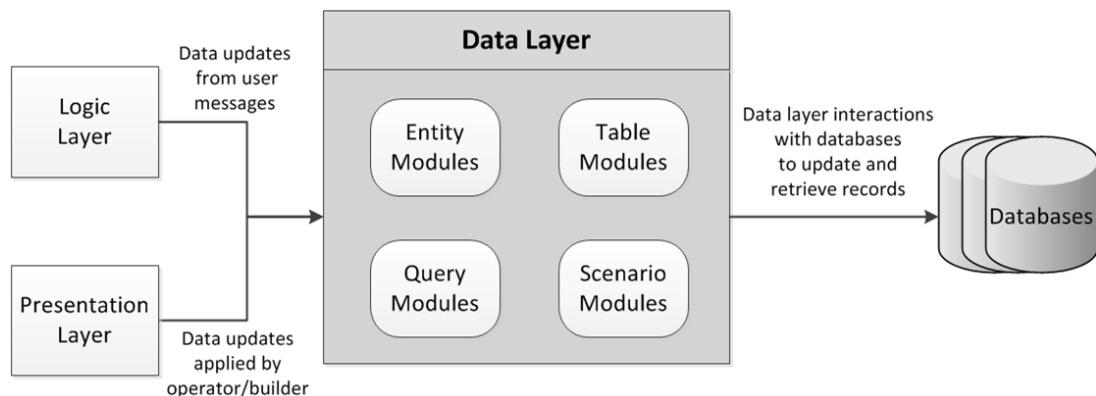
Messages sent from a user may trigger a significant event or status change for the associated instance. Should this occur, the TAU is able to create a *News* record to provide details of the issue. *News* templates are utilised by the TAU to create *News* records autonomously. The *News* record is created by merging a *News* template's text with existing data regarding the instance. This is carried out in the same format as the creation of output messages from an *Outgoing* template. The inserted data can either be current field data from the received message or prior data relating to the instance, which is stored in the scenario's database. Alternatively, the operator has the ability to manually create *News* records for an instance. For example, a received message may be flagged by the TAU to be manually resolved. Once the operator has resolved the issue they can create a *News* record to explain the consequences of the data. Each *News* record is stored in the *News* table of the *Principal* database. The purpose of these *News* records is to enable the operator to easily follow the progress of an instance from its start point to the current situation, providing up-to-date information on the instance. *News* records could also be used to broadcast updates to users as an instance progresses. Figure 5.11 illustrates the procedure for creating a *News* record when a message is analysed by the TAU.



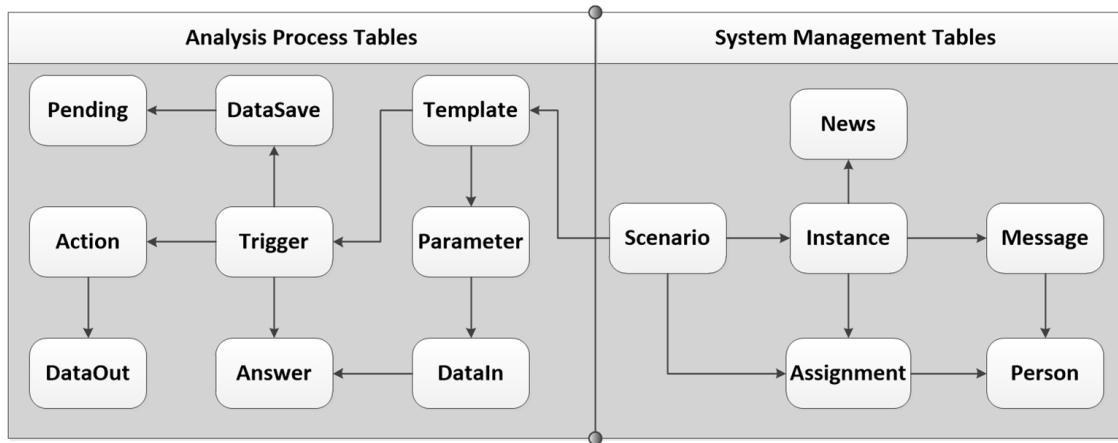
**Figure 5.11:** Process of creating a News record

### 5.2.10 Data layer of the platform

The data layer (Figure 5.12) consists of modules that interface with the platform's databases to facilitate the storage and retrieval of data, as well as performing updates and alterations. The *Principal* database is designed to store data on core entities of the platform, necessary to operate each scenario. This database, illustrated in Figure 5.13, is split into two sections. The right-side of the *Principal* database represents the tables used by the operator for the management of users and scenarios. There is a table to store records of each scenario and others to store records of a scenario's active instances and assignments. Additionally, there are tables to store records of the registered users and the messages that have been exchanged between the server and its users. The left-side of the *Principal* database represents the tables used by the TAU for the analysis procedures. These tables hold the data on entities which are defined by the scenario builder. This includes tables for the *Incoming* and *Outgoing* templates, the template parameters and the analysis entities, such as the *Trigger*, *Answer* and *Action* entities.



**Figure 5.12:** Design of the data layer



**Figure 5.13:** Tables of the Principal database

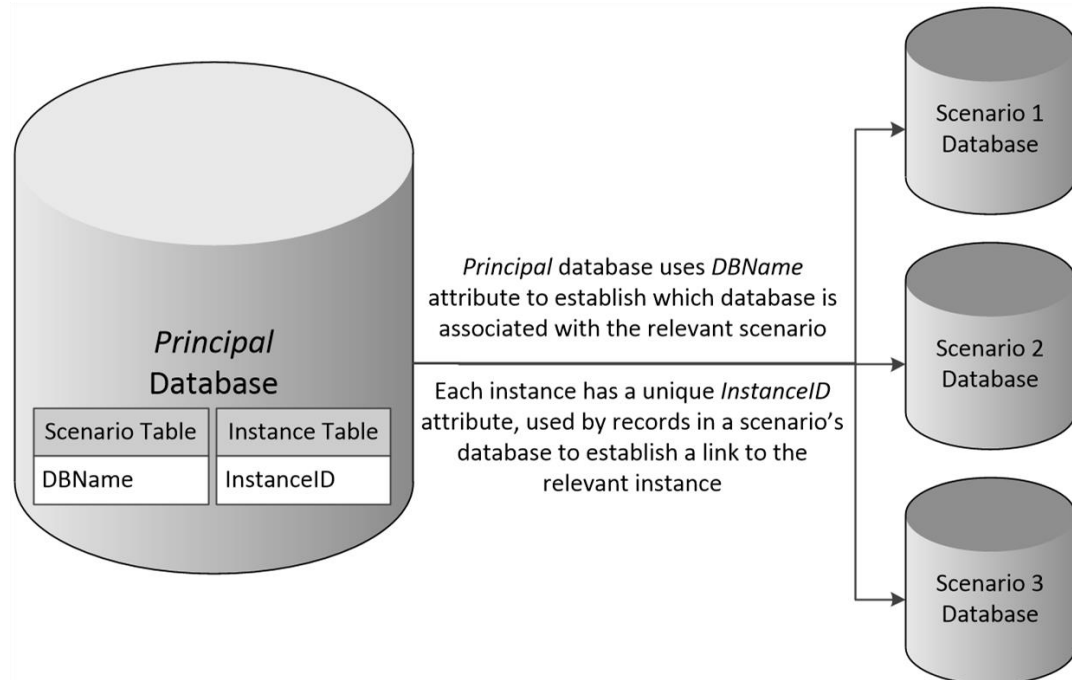
There is an entity module for each table within the *Principal* database. These have attributes to reflect the fields in the associated table, therefore all the data from one record in the table can be transferred to an entity module. This functionality enables the data to be manipulated and modified with ease by processes on all the platform's layers. An entity's values can then be saved back to the same record in the *Principal* database, updating its data.

Query modules provide functionality for the modification or collection of data across multiple fields within a record, multiple records in a table or data from multiple tables. The algorithms in these modules can be accessed from all layers of the platform for straightforward interactions with the *Principal* database.

Table modules are utilised to populate a table of data within the web-based user interface. For each table in the *Principal* database where there is a web-form in the web-based user interface that displays a list of records in that table, there is a table module to retrieve the stored data. This functionality allows the operator to view multiple records at once, with the ability to filter and sort the records. For example, the operator can view a list of messages in the server's *Inbox* and then filter the messages based on a specified instance.

The data layer provides access to the *Principal* database and each scenario's database. A scenario's database is designed and configured uniquely for its respective scenario, storing field data that is gathered by users or resultant from analysis of data previously collected. Scenario modules in the data layer have the functionality to

collect and store data in these databases. As the platform is designed with a generic framework it follows that these modules need to cater for each possible type of scenario database. On each occasion that access to a scenario's database is required, the database, table and field names are collected by procedures in the logic layer. These values are passed to the data layer, with a tailored query generated to successfully access and manipulate a scenario's database. The records in a scenario's database are linked to their associated instance by the instance's identifier attribute (Figure 5.14). Additionally, the name of the associated database is stored in each *Scenario* record in the *Principal* database, ensuring that a connection is established with the correct scenario's database.



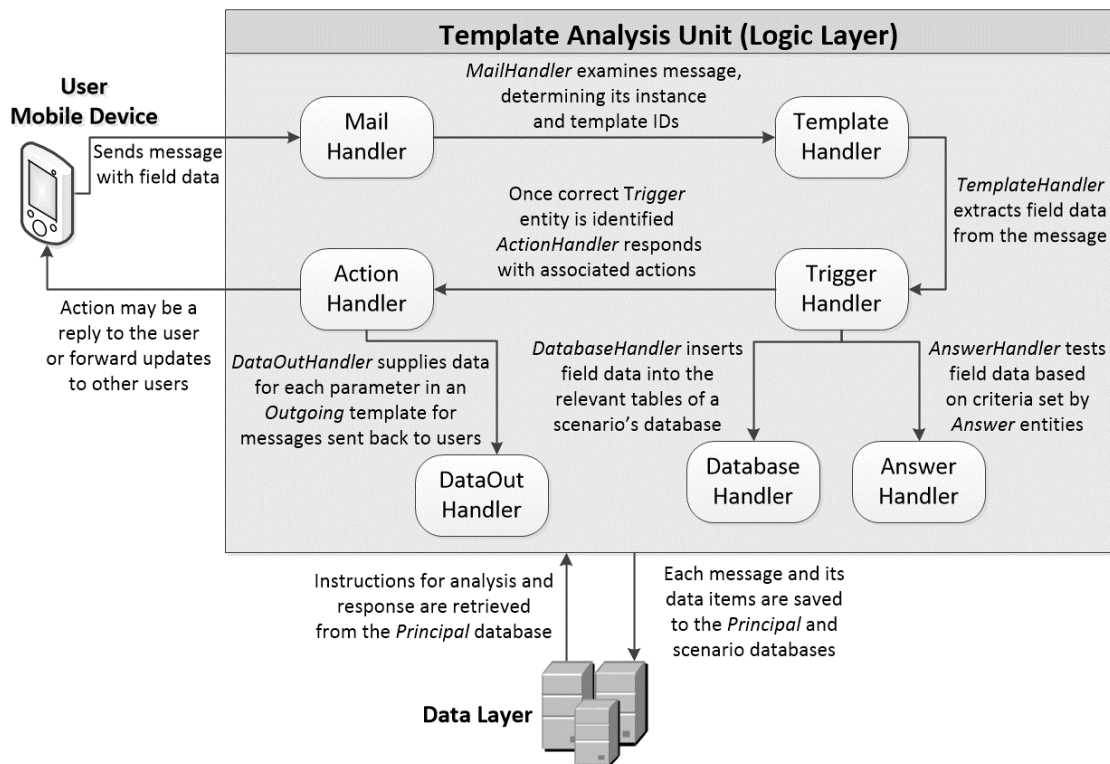
**Figure 5.14:** Relationship between the Principal database and each scenario's database

### 5.3 Data analysis process

The processing and analysis of a message is performed over a number of stages in the TAU. Each stage is split into a class, which is a construct that stores the methods and variables needed to effectively perform the operations of the analysis stage [183]. The programming language used to develop each class is C#, which is discussed in further detail in Section 5.5. Initially, messages are collected in batches at a regular time interval and then processed individually. Attributes of the message are retrieved and



the field data contained within the message text is extracted. The field data is tested against criteria defined in the associated *Trigger* and *Answer* entities of the template that the message is based on. Finally, a number of actions are performed based on this analysis, using *Action* entities. Figure 5.15 illustrates each of the stages performed within the TAU. Each class has algorithms and routines which are independent of the individual scenarios. The purpose of the TAU is to enable the server to communicate with users in real-time and update the progress of each instance with minimal human involvement.

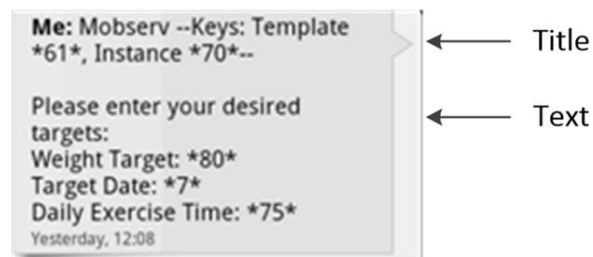


**Figure 5.15:** Message analysis stages of the Template Analysis Unit

### 5.3.1 MailHandler class

*Microsoft Outlook*, the email client installed on the server, retrieves email and SMS messages sent from the users. The *MailHandler* class interacts with *Outlook* to collect each message from the *Inbox* folder. Whether a message originates as an email or an SMS message it is received to this *Inbox* folder in the email format. Messages are collected from the *Inbox* folder at regular intervals. The message's attributes are gathered and saved as a new *Message* record in the *Principal* database. This includes the title, text, sender, timestamp and the method used to send the message. Determining the method of communication is achieved by examining the sender

address. Email messages contain a separate attribute for the title, whereas for SMS messages the message body contains the title and text of the message. Therefore, in SMS messages the title is identified by being between two “--” strings to separate it from the message text. Additionally, an SMS message begins with the word “*Mobserv*”, which instructs the SMS web service to forward the message to the platform’s email client. The *MailHandler* class checks each of these factors to distinguish the originating message type. An example of an SMS message sent from a mobile phone is shown in Figure 5.16.



**Figure 5.16:** An example SMS message sent by a user

In the case of an SMS message, which has been forwarded by the SMS web service as an email message, the email address that has been provided contains the originator’s mobile number. This number is extracted and used to find the record of the user who has sent the message.

When a user sends a message via the web-based user interface this automatically triggers the *MailHandler* class to process the message. The message attributes are similarly stored in a new *Message* record. However, the process of determining the communication method is skipped.

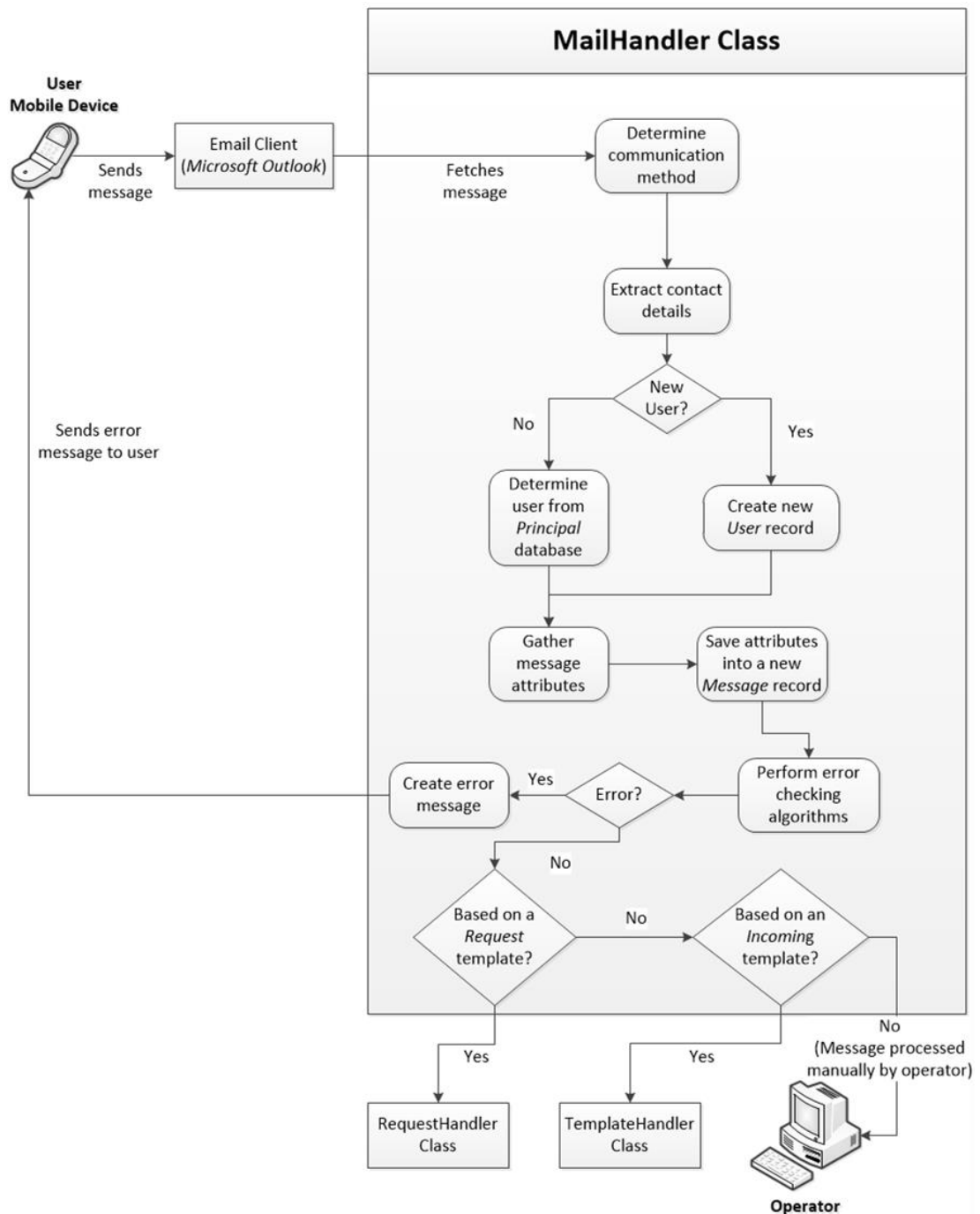
The key information contained within the title is designed to aid the *MailHandler* class in identifying how to proceed with the message analysis. Initially, it distinguishes whether the message is based on a *Request* or *Incoming* template; these are the two types of messages that can be analysed by the TAU. If a message is not based on either of these types of templates then it needs to be handled manually by the operator. In the case of *Request* templates the title begins with “*Req:*”, followed by the name of the request. The data within the message text is passed to the *RequestHandler* class for analysis based on the request name. The *RequestHandler* class is the final stage for *Request* messages, resolving the request.

*Incoming* templates are utilised by the TAU in the analysis of messages associated with an instance of a scenario. Each message based on an *Incoming* template contains key identifiers within the title. These identifiers are pointers to entity records within the *Principal* database that the message relates to. The two main entities are the originating template and the associated instance of the message. Providing these identifiers enables the *MailHandler* class to find the correct entities. These identifiers are extracted from the message and saved as the *TemplateID* and *InstanceID* attributes within the new message’s record. These identifiers would have been prefilled into the title by the TAU when the template text was initially sent to the user. This would only occur for an instance if either the message is sent as a response to a previous message concerning that particular instance or the user is already assigned to that instance. Therefore, the user can reply with the correct information pre-inserted into the message. The user is then only required to insert the necessary field data in the body of the message. However, if the instance is unknown, then the user would be required to add this information manually.

The *MailHandler* class passes each of the key items within the title through error checking algorithms to ensure they are valid keys that exist within the *Principal* database and do not conflict with each other. For example, the *InstanceID* and *TemplateID* must be associated with the same scenario. Two further checks are

performed to ensure the actual message text is based on the *TemplateID* provided. The first check tests if the beginning text in the message, before the first parameter position, matches that of the original template text. The second check tests if the total count of parameters in the message match the count of the originating template. The parameter count is calculated based on the amount of asterisks in the message. Once these error checks have passed successfully the TAU can continue processing the message's text. If the *MailHandler* class picks up any errors, a message is sent back to the user detailing the errors and an alert status is set to notify the operator. Figure 5.17 illustrates the main operations of the *MailHandler* class.

The *MailHandler* class acts as an intermediary between the remainder of the logic layer and the communication methods that interact with the user, whether this is via SMS, email or the web-based user interface. The functionality contained within the rest of the TAU is performed without knowledge of the front-end communication method used, applying the principles employed by the telecommunications order management system [168] discussed in Chapter 4. This is to ensure there is no duplication of code within the class methods that would otherwise perform the same business process.



**Figure 5.17:** Operations of the MailHandler class

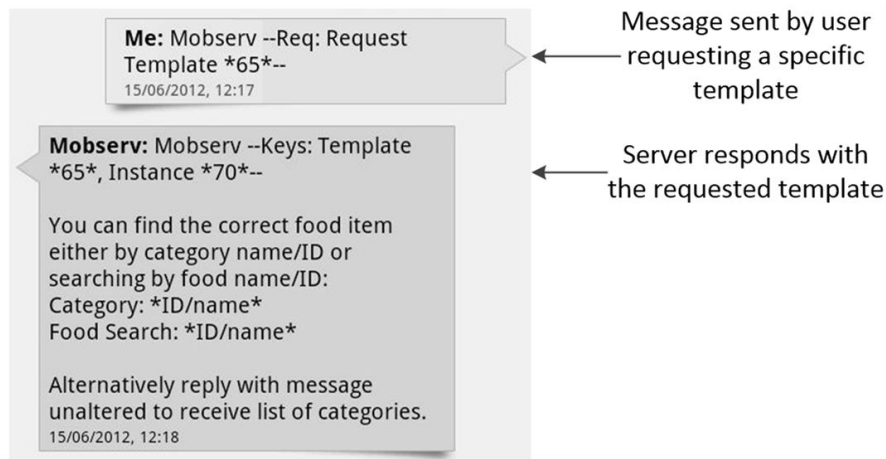
### 5.3.2 RequestHandler class

An example of the use of *Request* templates is for the joining of new users to the platform. If a message is received from a new user then a reply is sent to the user containing the “Member Update” *Request* template. This template contains parameter positions for the user to provide their personal details. Once the user replies with this information the *RequestHandler* class extracts the data from the new message to

create a record of the user. The “Member Update” *Request* template is also used by current members to update their details. This process does not apply for users who have requested *Full Anonymity*. These users are required to type the phrase “*Mobserv Anon*” at the beginning of each message that is sent to the server in order to bypass the requirement of providing their details.

Some *Request* templates are designed to enable a dialogue between the server and a user. The user can make a request to the server for a list of scenarios. The server would then reply by sending a message containing the name and unique identifier of each active scenario. Additionally, the user would receive the “Select Scenario” *Request* template, which would allow them to request starting a new instance of one of these scenarios. The user is only required to provide the selected scenario’s identifier in their response. Each scenario can only have one template for starting a new instance, referred to as an *InstanceCreator* template. This template is used to create a new instance of a scenario and requires the user to fill in key information concerning this new instance. Therefore, in addition to the operator manually creating new instances, *InstanceCreator* templates provide extra flexibility in allowing users to initiate instances.

Users can also request to join currently active instances. The outcome of this request may depend on the user meeting particular requirements of the scenario. For example, the user may be required to be currently in a particular location. Once a user joins an instance they can make a request for a list of available *Incoming* templates associated with the instance’s scenario. The server sends a response containing the name and identifier of each *Incoming* template. Using this information the user can make a request for a specific template, whereby the server would send the unmodified template text. This is illustrated in Figure 5.18 by an example of a *Request* message sent by a user with a response from the server. The user would then only need to insert the requested field data in the parameter positions of the received *Incoming* template. Alternatively, the user can request for a list of assignments for an instance they are joined to, followed by a request to be allocated to a particular assignment.

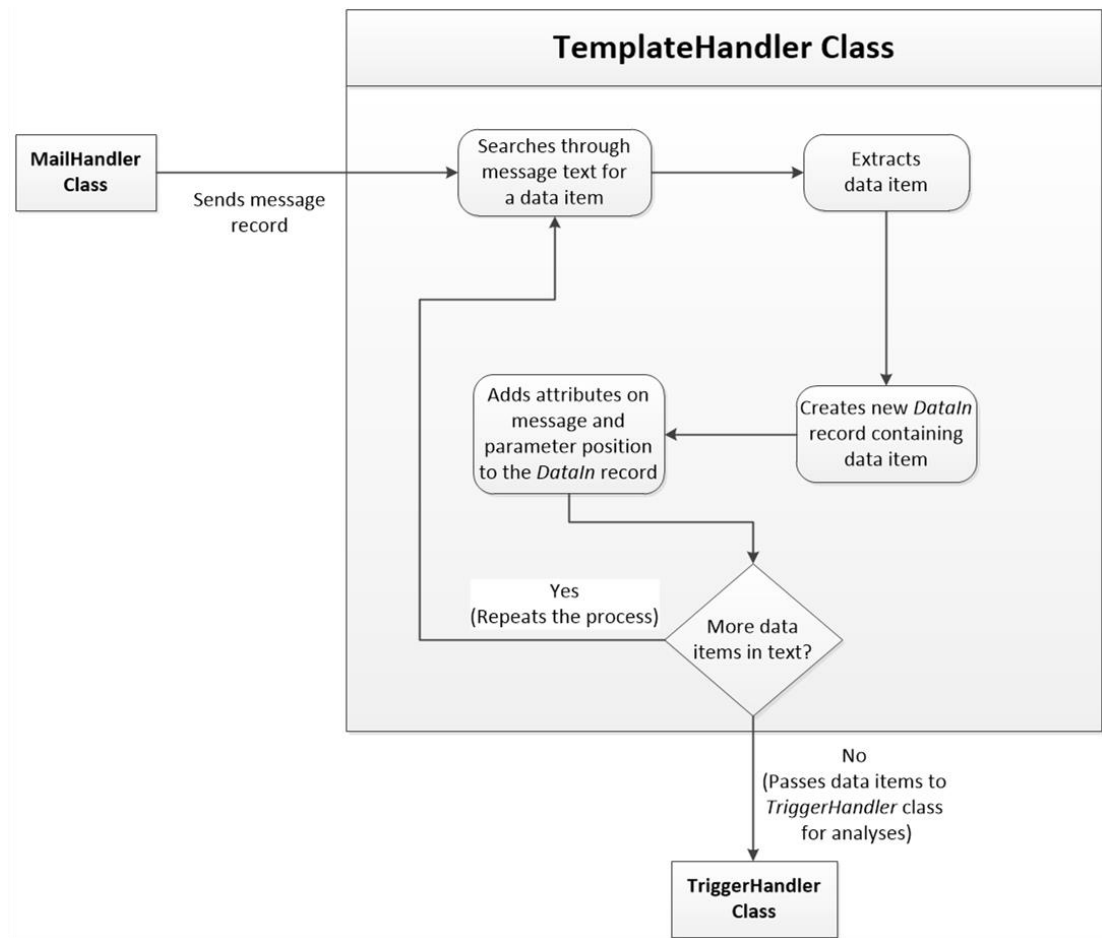


**Figure 5.18:** Request message sent from a user, followed by the reply from the server

### 5.3.3 *TemplateHandler* class

The next step is to obtain the field data from the message. In order that the appropriate actions are taken when a message is received it is necessary for the TAU to individually analyse each data item provided in the message body. The first stage of this process is to extract each data item from the parameter positions defined within the original template. The *TemplateHandler* class searches through the message text to find the data items. This process relies on locating each pair of asterisks in which a data item is enclosed between, enabling the *TemplateHandler* class to determine where each data item commences and finishes within the message text.

The *DataIn* table in the *Principal* database stores every data item retrieved from the users. Each string, representing a data item, is extracted from the message and saved to a new record in this table. The *DataIn* record contains attributes on the data item, this includes the data text; the identifier of the message and the parameter position within the template where it was extracted. These data items are the only parts of the message text that are required for analysis. The remainder of the message is constructed from a template to help the user insert field data accurately. Therefore, extracting and storing these data items separately enables them to be processed in a straightforward manner, along with the simplicity of locating the data items for future analysis tasks. Figure 5.19 illustrates the main operations of the *TemplateHandler* class.



**Figure 5.19:** Operations of the *TemplateHandler* class

### 5.3.4 *AnswerHandler* and *TriggerHandler* classes

Once the data items are extracted from the message they are processed by the *AnswerHandler* and *TriggerHandler* classes. These two classes perform the core data analysis for all types of scenarios. The *AnswerHandler* class uses *Answer* entities to analyse individual data items from a message. Each *Answer* entity is associated with one parameter position of a template and it contains criteria which the data item in the corresponding parameter position is required to fulfil.

There are various types of criteria that could be defined, each being tested on the data item. It may be necessary for the data item to be of a particular data-type, such as a boolean, integer, string or date. All data items are stored as strings in the *DataIn* table. Therefore, in this situation an attempt would be made to convert the data item to the specified type. This conversion attempt must be successful to meet the data-type criteria.

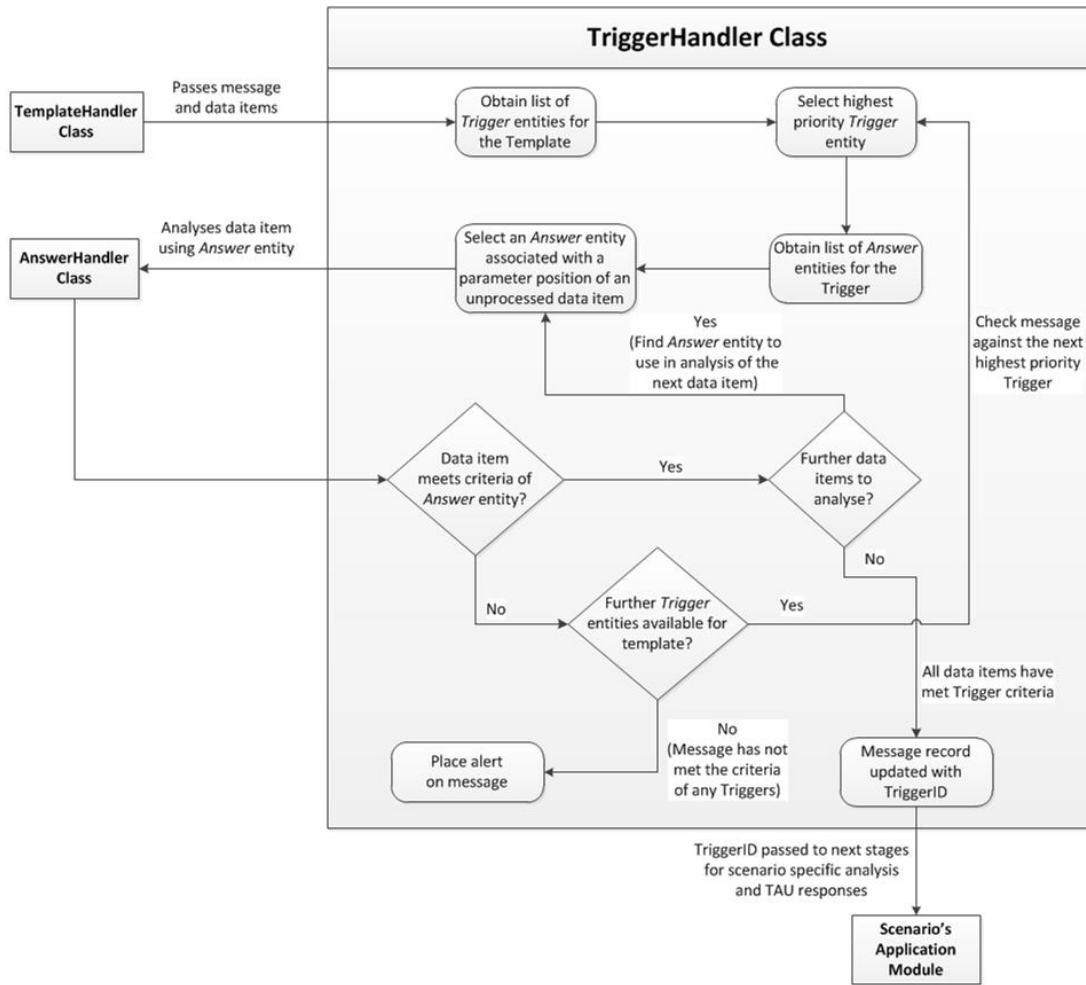


The data item could be required to either match or not match a specified answer. If the data item is a string there is an option to test whether the data item contains the text of a supplied answer. Alternatively, the data item could be checked against multiple answer strings where it only has to match one of the answers.

For numerical data there can be criteria for the data item to be greater or less than a value or lie within a set range from a value. In these situations the answer could be a fixed value. Alternatively, the answer could be a pointer to an attribute of the instance, person or message or to a field in the scenario's database. If any of these conditions are set by an *Answer* entity then the associated data item is tested to observe whether it passes or fails.

There could be more than one possible *Answer* entity for each data position in a message. A *Trigger* entity is used to determine which *Answer* entities the message's data items are examined with. A *Trigger* entity groups a set of *Answer* entities together. There must be at least one *Answer* entity corresponding to each parameter position of the originating template. Each *Trigger* entity is related to an *Incoming* template, with one of its attributes being the template's identifier. It may be possible for users to respond with differing field data that has the same meaning. There can be multiple *Answer* entities for a data item to assist in covering each possible type of the same response. If there is more than one *Answer* entity in the group for a particular parameter then each *Answer* entity is examined until the data item successfully meets one of their criteria.

If a template has more than one *Trigger* entity then each one is given an order of priority. The *TriggerHandler* class works through the answer set of each *Trigger* entity, starting with the highest priority. This continues until a *Trigger* entity is found in which the message's data items have met all the criteria of its answer set. This *Trigger* entity is passed to the *ActionHandler* class for the final stage of determining actions to perform. This process is designed in order that the TAU can perform different responses depending on the field data provided. Figure 5.20 illustrates the main operations of the *TriggerHandler* class.



**Figure 5.20: Operations of the TriggerHandler class**

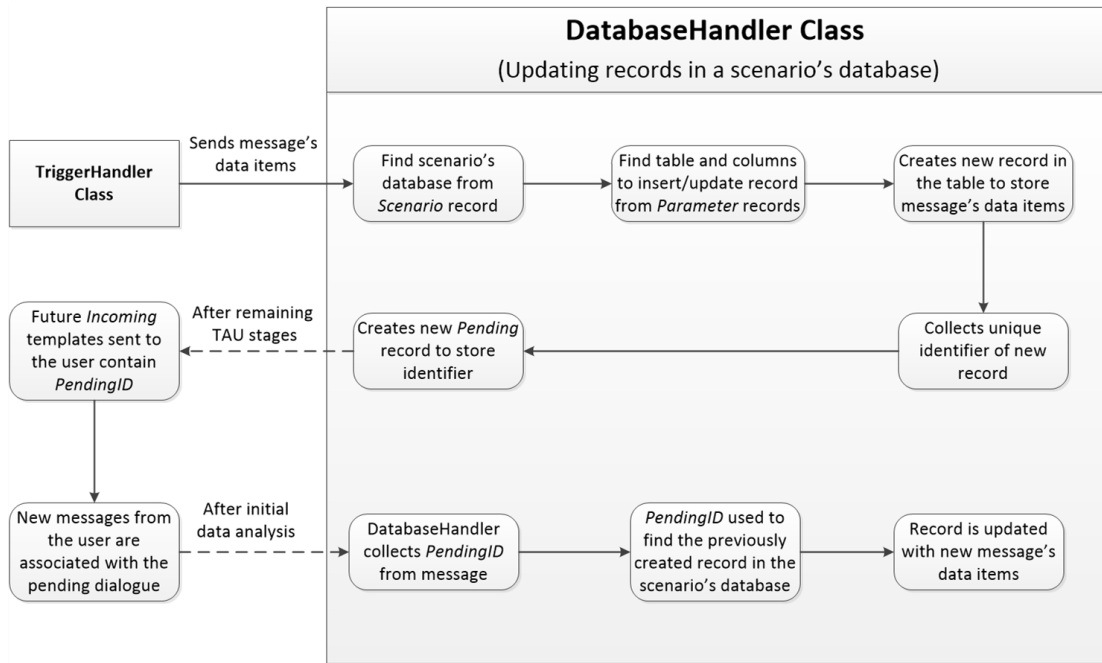
As each scenario is different it may need extra functionality that is not covered by the core analysis functionality. As previously mentioned, each scenario has its own application module for analysis of data specific to that scenario. The message's data items may be passed to this module for additional analysis of data prior to the TAU performing actions. The application module can alter the final responses to be carried out and utilise further error handling algorithms to ensure a message's data is accurate.

### 5.3.5 DatabaseHandler class

Before the TAU performs any actions a message's extracted data items are saved to meaningful locations in the scenario's database. The TAU would then be able to access this data during the analysis of future communications. The scenario builder selects each data item based on its parameter position and specifies the table and column of the scenario's database to insert the data. A record in the *DataSave* table of

the *Principal* database stores these details. It is possible for the *DatabaseHandler* class to either insert a new record into a table or update a current record. This depends on whether the table allows more than one record per instance and whether any records for the instance currently exist in the table. Some tables in a scenario's database would only have a maximum of one record for each instance. In this situation a check is made on whether there is a record in place yet for the instance. Where no record is found a new record, containing the field data, would be inserted into the table. Otherwise the record is updated with the new field data in the relevant columns.

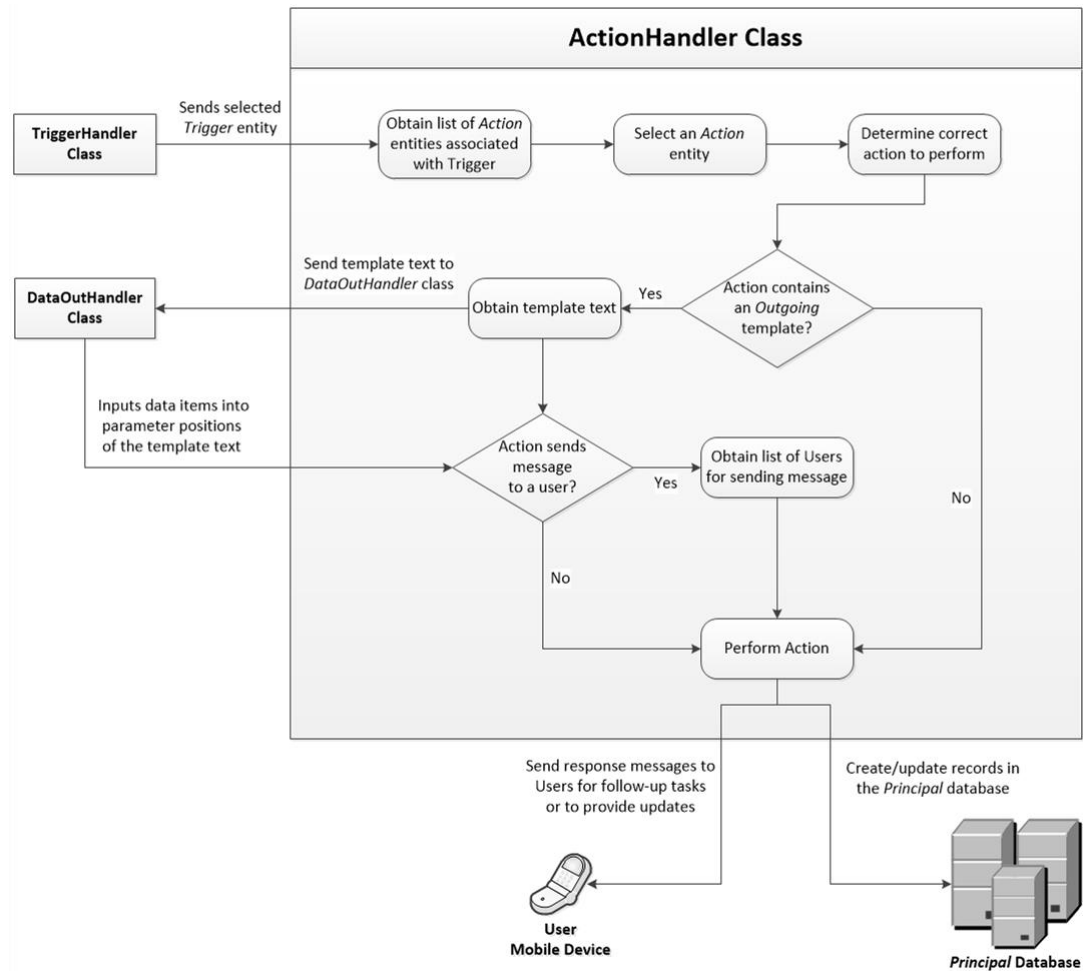
Other tables in the scenario's database may be able to hold multiple records for each instance. In normal circumstances, a new record would be created to store the new field data for these tables. However, there are certain actions that may require a user to update the details of a record over multiple messages; if this is the case the *DatabaseHandler* creates a new *Pending* entity. A *Pending* entity enables multiple messages, which are exchanged between the server and a user, to be associated together. This feature allows the *DatabaseHandler* class to update a record in one of the tables of the scenario's database, where multiple records may exist for a particular instance. The *TableID* and *TableName* attributes of the *Pending* entity are used to locate the specified record, effectively allowing the *Pending* entity to point to any record in the scenario's database. Therefore, the field data of any future messages linked to the *Pending* entity can be used to update the specified record of the scenario's database. Figure 5.21 illustrates the process where the *DatabaseHandler* class creates and updates these records within a scenario's database.



**Figure 5.21:** Process of the DatabaseHandler class for handling tables that contain multiple records associated to one instance

### 5.3.6 ActionHandler class

An Action entity defines a specific action that can be performed by the ActionHandler class. The Trigger entity, which is passed to the ActionHandler class, points to a set of one or more Action entities that have been defined by the scenario builder. When a trigger's criteria are matched by all the data items within a message then each of the actions are carried out. There are a variety of action types available to the TAU. Figure 5.22 illustrates the process where the ActionHandler class performs actions; both for replying to users and updating the Principal database.



**Figure 5.22:** Operations of the ActionHandler class

The *Reply* action is used for sending a reply message to the user. This action makes it possible to request new data from the user, provide instructions on a new task or update the user on changes in the instance's status based on the information that they have supplied. The significance of this action is that it enables the server to maintain a dialogue with the user without the involvement of the operator. When the action is initially created the scenario builder is required to choose an *Outgoing* template for the response. The text in this template is used to create the new output message. A title is generated to provide key information concerning the message, including the template and instance identifiers associated with the message. To facilitate a continued dialogue, the TAU may also send an *Incoming* template to the user alongside the reply. The purpose of this *Incoming* template is to enable the user to provide a further response back to the server in the correct format. The user can then simply insert any newly requested field data into this *Incoming* template and send it as a new input message to the server.

The *MessageSend* action works on the same basis as the *Reply* action but is used to send a new message to other users participating in the same instance. Furthermore, a message can be sent to groups of users in the same instance. These response groups include all users participating in the instance and users either on or off assignments. This helps the server to broadcast messages out to potentially many users. For example, the server could broadcast an update to all the users participating in an instance when the instance's status is changed due to the analysis results of a received message. Alternatively, this technique could be used to ask multiple users to carry out the same data collection task.

The *Forward* action forwards the user's message to other users in an instance. This action uses the same broadcast feature of sending a message to multiple users. However, with this action it is just the original message from the user that is sent to other users. This action is used to update other users with the data that a user has provided, notifying them of the discovery and providing helpful details.

A *Notification* action is available to send a confirmation message back to the user. This is a form of providing feedback to the user regarding their recently sent message. The user is informed that the message has been received and successfully processed without errors. A *Notification* action does not require an *Outgoing* template. It is created based on a fixed message template and the title of the received message.

An *InstanceNew* action is triggered when a user sends a message based on an *InstanceCreator* template, initiating a new instance of the scenario. A scenario may define an optional restraint on the user's ability to initiate an instance. If this is the case then the new instance is initially placed on a *Preliminary* status. The operator would be required to confirm the instance in order to make it active. The reason for this would be to prevent erroneous instances being created, using up the server's resources. The scenario builder could prevent users from initiating an instance of a particular scenario by not defining an *InstanceCreator* template.

Users can manually request to join an instance via a *Request* template. Alternatively, the TAU could automatically add a new user to an instance with the *UserAdd* action. This would occur if the new user has been referred to by another user of the instance. The TAU could also select users based on a specified location to add to an instance.

In the situations where the TAU adds a user to an instance, the user would be notified but remain inactive until they reply with a confirmation message. This provides flexibility by enabling both users and the server to add more users to an instance. This action can also add new users to the platform if their details are not currently stored, kick-starting the exchange of messages to retrieve a new user's details.

A *StatusChange* action updates the instance to a new status. Each possible status an instance goes through is defined by the scenario builder during the implementation of the scenario. Updating an instance to a new status is a way of the instance progressing, whilst showing its current circumstances. This can result in users being provided with new assignments and old assignments ending early if they no longer need to be fulfilled.

The analysis of a message may raise a new issue regarding the associated instance. In this case the *AssignAdd* action can create assignments and place users, who are currently joined to the instance, on the new assignment to resolve the issue. In a dynamic scenario where new assignments are continuously arising, the *AssignAdd* action helps to progress the instance by efficiently allocating the users. This can be achieved by selecting the closest user to the location of an assignment, if it is location-dependent. This action could be used in combination with the *MessageSend* action to provide the user with details of the new assignment they have been allocated to. Alternatively, once a user has finished providing useful information for a particular assignment or achieved the assignment's objectives they can be removed from it. The *AssignRemove* action is utilised to remove users from assignments. An automatic notification message can be sent to the user to notify them of this change.

There could be certain messages of high importance that require an operator to view and then decide how to proceed. Alternatively, it may only be necessary that the operator is aware of a new situation. *Alert* actions flag messages for these situations. There are two types of alerts, whereby the alert attribute in the message entity indicates the type that applies. An *Alert* action places the received message on an alert status of *HighPriority*, which notifies the operator of its importance, allowing them to easily find the message. They could then manually perform any actions that are necessary. There is also an *ErrorDetected* alert status that a message is set to if the TAU detects erroneous data in the analysis stage.

Each instance has a list of *News* records to enable the operator to keep track of the progress. The platform is designed for *News* records to be created when a user reports back significant events occurring out in the field. If a user's message meets the specified criteria then a *NewsInsert* action would be used to create a *News* records. This action requires a *News* template to obtain the text for the *News* record. If there are any parameter positions within the template these are updated with the relevant data, depending on the message received and the instance. In addition, data from the *News* record can be sent out as a message to the original user or a group of users to keep them up-to-date with the instance's progress.

The *TemplateList* action instructs the *ActionHandler* class to send a list of available *Incoming* templates to a specified response group. This action can be used to supply the list to a user after the setup data of a new instance has been collected or to send to users who have recently been added to an existing instance.

### **5.3.7 *DataOutHandler* class**

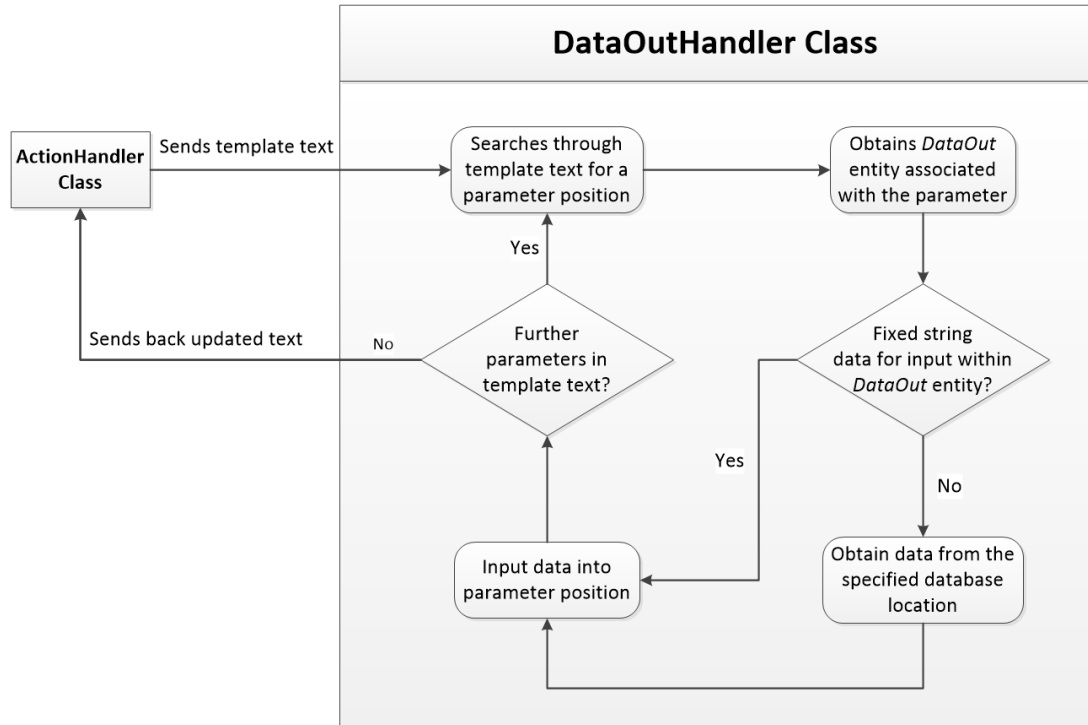
*Outgoing* templates used for actions may have parameter positions that need to be filled in with the appropriate data. This is achieved by the *DataOutHandler* class. Each action that requires an *Outgoing* template points to a list of *DataOut* entities, one for each parameter position within the template. A *DataOut* entity instructs the *DataOutHandler* class on which data item is to be inserted into its corresponding position in the template. This could be a fixed string provided by the *DataOut* record, in this case the string is inserted into the text location of the parameter position.

There are other locations where the appropriate data can be obtained. The *DataOut* record may point to the original message's text, with its *FieldID* attribute indicating the parameter position to extract the data from this text. The *DataOut* record could also point to an attribute of the *Message* entity, *Person* entity (the sender of the original message) or the *Instance* entity. For example, the *DateSent* attribute of the message entity may be selected in this way.

A further location that is used to retrieve data is the scenario's database. In this situation, the *DataOut* record would point to a field in the database, based on the table and column names provided. The scenario builder would have defined these details in the *DataOut* record.



The range of locations in which the *DataOut* class is able to retrieve data enables the TAU to be flexible and accurate in providing details to a response. Figure 5.23 illustrates the operations performed by the *DataOutHandler* class.



**Figure 5.23:** Operations of the *DataOutHandler* class

## 5.4 Developing a scenario application

The implementation of a new scenario involves a three-stage scenario development process. Once these stages have been completed by the scenario builder the new scenario is ready to be run by the platform.

### 5.4.1 Stage 1: Creating the scenario's database

The first stage involves developing the scenario's database. The builder constructs the database using *Microsoft SQL Server Management Studio* [184], a software application for designing and managing relational databases. The builder is initially required to determine and lay out the components of the scenario. These components are defined as entities of the scenario. A table is required in the scenario's database for each entity, to store data on the entity's attributes. A new record needs to be created in the *Scenario* table of the *Principal* database. This record provides a pointer to the scenario's database as well as the name and description of the scenario.

### 5.4.2 Stage 2: Template and analysis criteria

In the second stage, the builder creates the *Incoming* and *Outgoing* templates that will be used in the communication structure of the scenario. This may include an *InstanceCreator* template, which would enable users to initiate a new instance. Figure 5.24 illustrates the *Template* web-form of the web-based user interface. On this web-form the builder is able to create new *Incoming* and *Outgoing* templates for a scenario. The builder inputs the name and the text of the template, as well as defining other attributes such as declaring whether the template is an *InstanceCreator* template. For each template that is created the parameters are automatically defined based on the locations of the template text that are enclosed by two asterisks. The builder can provide a name and alter the temporary text of each parameter afterwards. The name is used to easily identify the parameter when creating the analysis entities, such as an *Answer* entity.

**Template Form**

Scenario: 7: Diet Diary  
Show Templates for: Incoming  
☐ Select All  
Delete Refresh

	TemplateID	Name	Type
<input type="checkbox"/> Select	43	New Diet Diary Instance	0
<input type="checkbox"/> Select	58	Input Diet Details	0
<input type="checkbox"/> Select	61	Input Auto Targets	0
<input type="checkbox"/> Select	63	Input Manual Targets	0
<input type="checkbox"/> Select	65	Auto Food Search	0

1 2 3

Save New Delete Create/Edit Triggers Parameters

ID:   
Name:   
Template Structure:   
Parameter Count: 0  
Response Template: None  
Instance Creator: ☐ Yes ☒ No  
Category: Setup  
Allow Attachment: ☐ Yes ☒ No  
Attachment Table:   
Attachment Column:

**Figure 5.24:** Template web-form within the web-based user interface

At least one *Trigger* entity is required to be associated with each *Incoming* template. The first *Trigger* entity is created automatically and set as the default trigger. The

builder constructs an *Answer* entity for each parameter position, requiring analysis, in the *Incoming* template. An *Answer* entity is created in the *Answer* web-form, where the builder defines the required data type and value criteria that the data item would be checked against. If an answer value is not fixed the builder would define a pointer to a database or entity location where the value is stored. In the *DataSave* web-form the builder defines the location, within the scenario's database, to save each data item of a message based on the template.

The *Action* web-form is where the builder defines the response actions for each trigger. Each action is created by defining the action type. Actions that involve sending a message require the builder to specify the response group and the *Outgoing* template to use. The other action types have alternative selections to choose from, for example the alert action requests the builder to select the appropriate alert type. For actions that require an *Outgoing* template the builder would define the fixed string or a data location to use as input for each parameter position of the template. A new *DataOut* entity would then be created for each one.

### **5.4.3 Stage 3: Application module development**

Finally, the builder creates a bespoke application module if additional analysis is required for the scenario. The application module interacts with the *TriggerHandler* class by a defined set of input parameters and a returned output variable. This module has a loosely coupled relationship with the classes of the TAU, whereby only the input and output data is known by these classes. No internal workings need to be established with the classes of the TAU, ensuring that the application module can be easily integrated into the platform.

When a message has fulfilled the criteria of a *Trigger* entity then the trigger's identifier value is sent to the application module as an input parameter to check if there are additional calculations or analysis to be performed. Additional input parameters include the associated *Message* and *Person* entities. The builder is required to link each method in the application module to a *Trigger* entity associated with the scenario. The method can then use and manipulate the message's data. The application module has access to the core entities of the *Principal* database and the scenario's database tables via the modules of the data layer. As output, the method would return a value to the TAU to indicate the trigger identifier. Therefore, the

method has the ability to keep the same trigger as before or instruct the TAU to perform the actions associated with another trigger. Once the application module has been developed and integrated into the platform's logic layer, the scenario is ready for active use. The operator is then able to take over the running of the scenario and each of its instances.

## **5.5 Development environment and software tools**

The *Connected-Mobile Platform* has been developed in the *Microsoft Visual Studio* integrated development environment (IDE) [185]. This is a software application that provides a vast array of tools, which are contained within a library of classes known as the .NET framework, for developing new applications that run on *Microsoft Windows* operating systems. Within the *Visual Studio* IDE the platform's presentation layer utilises ASP.NET (*Active Server Pages*), a server-side framework for web applications [186], in order to create and display the web-based user interface. Each element of the web-based user interface has been designed using the *HyperText Markup Language* (HTML) to display the content in a client's web browser, which includes the web-forms for the operator, scenario builder and users of the platform. The ASP.NET framework contains a set of graphical tools to assist in constructing the HTML content of each web-form. The platform utilises the class libraries available from the ASP.NET framework to collect data sent from a client device via the web-forms for processing in the logic layer. The ASP.NET framework is also used to collect and display processed data on the web-forms on client device.

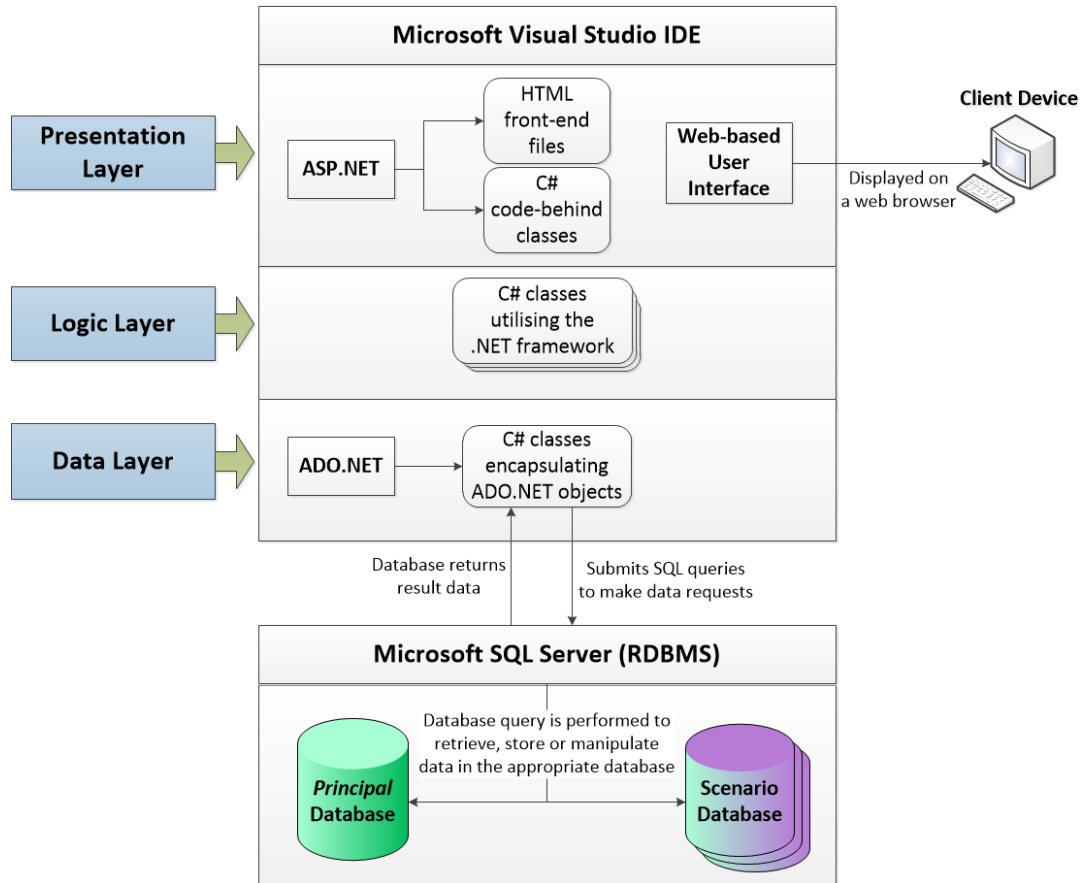
C#, an object-oriented programming language, is used to code the business logic and rules of the platform. Each component of the platform's logic layer is coded into a C# class within the *Visual Studio* IDE. These classes are the blueprints that store the methods, event procedures and variables necessary to perform the application processes [183]. There is also a C# code-behind class for each ASP.NET web-form that holds the server-side code for reacting to user events during interactions with the web-based user interface.

The C# classes in the platform's data layer utilise the set of ADO.NET (*ActiveX Data Objects*) classes within the .NET framework. The functionality provided by the ADO.NET classes enables the data layer to access the platform's databases for the storage and manipulation of data [187]. Additional requests are made to the databases

for the retrieval of data which would then be stored in ADO.NET objects, such as data-row and data-table objects.

The classes in each layer of the platform interact with each other to pass data along. For example, information received from a user in the web-based user interface would be passed to the classes of the TAU in the logic layer for processing. Results would be passed to the data layer for storage. Also, additional data may be retrieved from a database to assist in processing a message. Analysis results would also be passed to the presentation layer to be displayed to the user.

The platform's databases are stored and handled by *Microsoft SQL Server*, a relational database management system (RDBMS) that structures data into database tables consisting of fields and rows [188]. Data requests to a *SQL Server* database are performed via SQL queries. Each time a class within the platform's data layer needs to interact with a database an ADO.NET object is created, containing the appropriate SQL query to make the required data request. The design of each database has been achieved using *Microsoft SQL Server Management Studio* [184], which provides a user interface for developers to construct and maintain *SQL Server* databases. Figure 5.25 illustrates the development environment and software tools used to develop the platform.



*Figure 5.25: Development environment and software tools for developing the platform*

## 5.6 Experimental setup for operating scenarios in the case studies

Chapters 6 and 7 discuss case studies of two different types of scenarios. Experimental runs of each scenario were performed with the experimental setup described below. The *Connected-Mobile Platform* that has been described in this chapter was installed on a computer that had *Microsoft Windows Server 2008* running as the server operating system. The computer's hardware consisted of a 3.2 gigahertz (GHz) dual-core central processing unit (CPU) and a 4 gigabyte (GB) unit of random-access memory (RAM). This computer acted as the server for the experiments and had the .NET framework and *Microsoft SQL Server* installed to facilitate the running of the platform in order to handle the application processes and maintain the databases respectively. The server operated continuously to maintain the running of the modules in each layer of the platform in order that clients could interact with the server and newly received data could be processed without interruption.

The scenario builder had direct access to the server to create the databases and develop the application modules for each new scenario, utilising *SQL Server Management Studio* and the *Visual Studio* IDE respectively. Defining template content and analysis criteria for each scenario was achieved either directly on the server or on a separate client computer connected via the server's local area network (LAN). The operator used the same LAN connection to maintain scenarios. Both of these roles rely on interacting with the server via the web-based user interface. The web-based user interface was only accessible via this LAN connection for the purpose of the experiment on the prototype of the platform. However, the web-based user interface would be published to a website that is accessible from any location for real-world use.

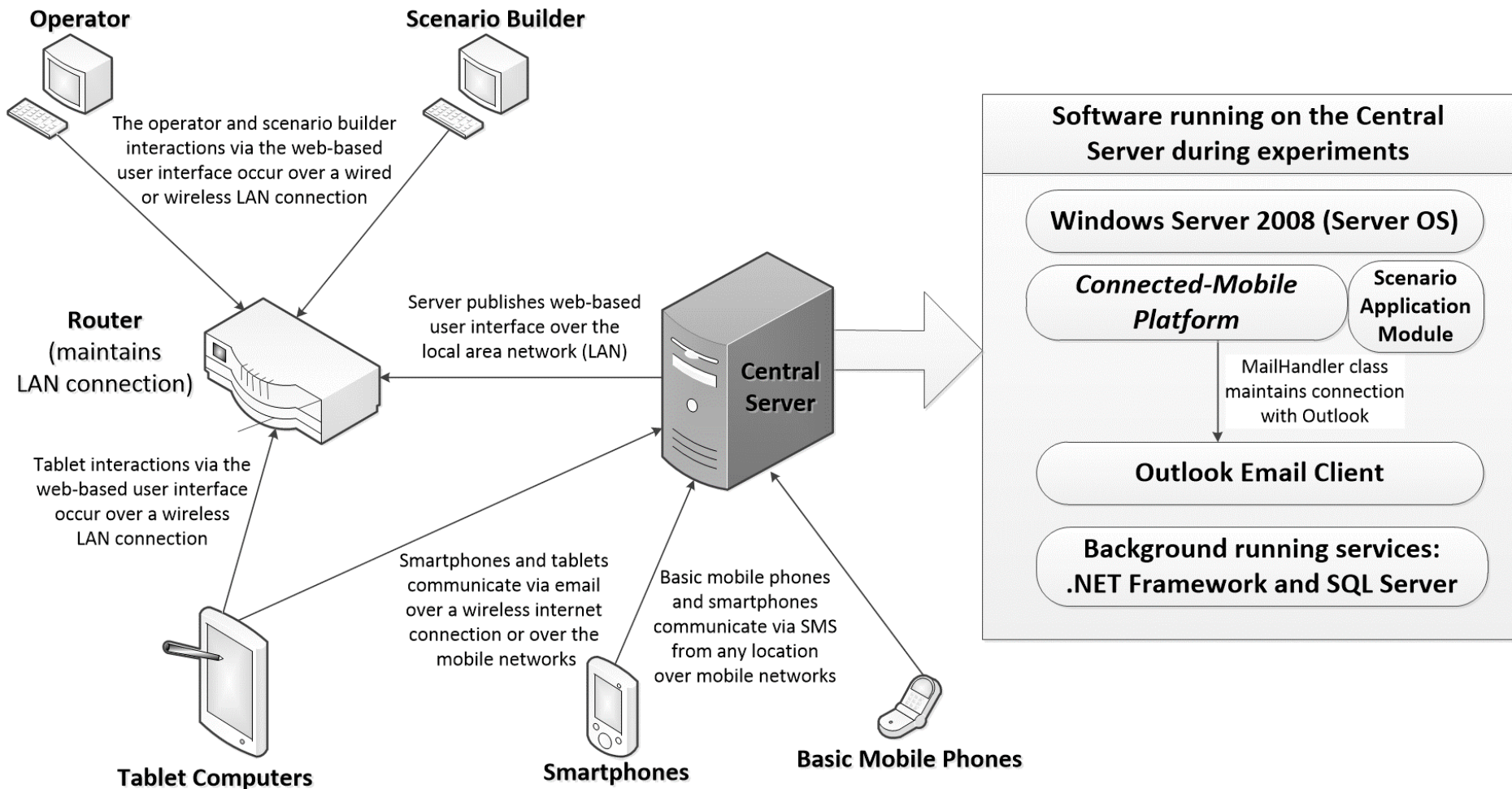
*Microsoft Outlook* runs concurrently on the server and is configured to receive all emails from the platform's email account, which is *mobservs@googlemail.com*. A "Send/Receive" request, within *Outlook*, is set to run automatically every minute to retrieve new email messages from Google's email server. These messages may either be sent directly from a user's device via email or an indirect SMS message that has been converted to email via *TextLocal* [42], the SMS web service. A C# *Timer* object, within the platform's core application, automatically runs to check *Outlook* every minute for new messages that need processing.

The collaboration between the platform and *Outlook* has been established for the experiments to ensure messages sent by a user either via email or SMS are retrieved within a short timeframe in order that the server can respond in real-time. This has been tested within the experiments by observing the average length of time it took for a user to receive a reply from the server after they had sent a message and whether there was a substantial delay if multiple users send messages to the server simultaneously.

The users, within the experiments, interact with the server via three different device types; basic mobile phones, smartphones and tablet computers. All SMS messages are sent by a user operating either a basic mobile phone or a smartphone. The server's responses are received by the same user device that sends an SMS message. All screenshots for SMS messages were taken from smartphones as this facility was not possible from the basic mobile phones used.

Users operated both smartphones and tablets for sending and receiving email messages. Server replies via email are stored by an email server that holds the user's email account. Therefore, email messages were accessible by any internet-enabled device the user owned during the experiments. Tablet computers were used to demonstrate the user interactions via the web-based user interface (using the LAN connection). Each user would have a unique login identifier that enabled them to view all previously exchanged messages and send new messages on any internet-enabled device. Each device type was included in the experiments to make it possible to assess how user-friendly the interaction process is between a user and the server in terms of receiving the relevant information and inputting field data into templates to create new messages. Figure 5.26 illustrates the experimental setup for the case studies.





*Figure 5.26: Experimental setup of the platform*

## 5.7 Conclusions

The aim of the research was to investigate how people in a mobile society could be utilised for different scenario problems, by including multiple methods of interaction. The proposed platform and functionalities address the identified requirements by incorporating multiple methods of communication in order to support numerous scenarios, whilst offering a guided development process for the fast implementation of each scenario.

The usage of mobile communication methods in the communication process between the server and users has allowed the platform to employ users in any location and to contact them at any time, enabling scenarios to be tackled in a mobile environment. Text messaging via SMS is the prime technique utilised for server and user interactions. Additionally, email technology has been accommodated to handle the exchange of multimedia files and a web-based user interface with data entry via a web-form provides a user-friendly method of supplying data for smartphones and other internet connected devices. The user is able to choose their preferred method of communication. The user can switch at any time to another method of communication. The server would then respond via this new method, thereby continually meeting the user's choice in the communication process.

The platform has been developed based on a novel framework that is built on a client-server architecture. This novel framework has generic properties in order that it can support a range of different scenarios. The core functionality covers the features required for each scenario to facilitate the sending of messages to users and the processing of received messages. The key to this is the template mechanism that defines the structure of each message exchanged between the server and its users. The template mechanism enables messages to be uniquely tailored to each scenario in an ordered process. The *Principal* database provides a central location where data can be stored that relates to the core components of the platform such as the registered users, the exchanged messages and details of each template.

The template analysis process performed by the TAU has been designed to be flexible and robust in responding to field data. The template mechanism enables the TAU to operate on messages for each scenario. The TAU is composed of generic functionality, designed to work regardless of the scenario. However, a template's

associated analysis can again be tailored to meet the requirements of an individual scenario. The features provided by *Trigger* and *Answer* entities enable the scenario builder to choose from a wide range of variables in deciding the results of data processing, with many different types of results possible due to the limitless number of triggers for each template. These features have been put in place to provide the platform with an effective analysis component.

*Action* entities have provided the ability for a wide array of responses. One important facility that the *Action* entities offer is to enable the server to engage in a dialogue with its users, without the need for operator intervention. This can provide an efficient mechanism of supplying users with instructions, as well as reacting to collected field data in a timely manner. Each user can be allocated to different assignments, which can interlink with assignments provided to other users. This assignment allocation can be based on user location if the assignment is location-dependent, providing a means for the effective coordination of users. Therefore, a user can cooperate with other users to help progress and tackle issues and objectives within a scenario. At the same time the users can be informed of the progress of other users via the broadcast of updates. The responses made available by *Action* entities can provide details in great length to the users with the ability of response templates to re-use data from previous messages. Actions also enable users to initiate operations by offering the users the opportunity to request the creation of new scenario instances.

The core functionality implemented within the platform has been designed as a framework for developing societal systems that have the ability to mobilise a potentially large workforce scattered widely in a surrounding area of a location. It is important to evaluate the platform's ability to handle different scenarios. Two important factors to consider include the simplicity and speed of constructing a new scenario and the effectiveness of the platform once a scenario is active with participating users. This chapter detailed the three-stage process for developing new scenarios into the platform. This process is in place to simplify the integration of each scenario. The generic functionality of the platform is designed to support the requirements of the different scenarios whilst they are running with active instances. The next two chapters consider the actual implementation of two different types of

scenarios, each with unique characteristics and requirements, by applying the three-stage scenario development process.

## 6 Case Study 1: *Diet Diary* Scenario

### 6.1 Introduction

To evaluate the functionality of the *Connected-Mobile Platform*, two scenarios having different requirements and differing objectives have been developed. The first scenario, entitled the *Diet Diary* scenario, tracks both the daily food consumption and the activities performed by a subscribed user. The purpose of the scenario is to manage a user's daily calorie intake in order to help the user reach a desired goal weight over a specific period of time. This scenario has only a single user assigned to each instance, with a focus on one-to-one communication between the server and the specified user. This makes it possible to concentrate on demonstrating the platform's effectiveness in receiving messages, processing the field data and performing the appropriate responses.

The *Diet Diary* scenario utilises the core components of the platform, including the communication functionality to interact with users and the *Principal* database for storing messages. This scenario makes use of a number of the features that have been discussed in Chapter 3 and has subsequently been incorporated into the platform in Chapter 5. These features include the platform's ability to analyse field data and the use of templates to control the communication structure. The employment of these features into the *Diet Diary* scenario is described in this chapter.

It is important for the platform to enable the scenario to be developed simply and smoothly. Part of this process is the building of a database to cater for the scenario's unique attributes. This database contains tables unique to the *Diet Diary* scenario, which enables the platform to hold data efficiently on each user's *Diet Diary* instance. There is also a separate application module, specific to this scenario, which is easily integrated with the platform for analysis and calculations. These components are part of the three-stage scenario development process being used to implement the *Diet Diary* scenario, which is described in this chapter. This is followed by a discussion of the time savings attained from applying the scenario development process, whereby the *Diet Diary* scenario's development time is compared to creating an application from scratch.

This chapter investigates a running instance of the scenario from the user's point of view to demonstrate the platform's ability to support an active scenario and provide a user-friendly approach with its clients' interactions.

## 6.2 Details of the scenario

The *Diet Diary* scenario in this chapter aims to include features similar to those implemented within the *Perfect Diet Tracker* application [102], which was discussed in Chapter 3. In this case it is not necessary to develop the functionality from scratch as the scenario takes advantage of the platform's components to provide for its features. Each *Diet Diary* instance enables one user to monitor their daily calorie intake based on food and drink consumption and calorie burn rate from performing strenuous activities. This information can be used for weight management, whereby the user can provide a weight loss or gain goal over a timeframe. A *Diet Diary* instance would provide daily calorie intake and burn targets to assist the user in meeting their desired overall weight goal. User data on food and activity items would be analysed by the *Template Analysis Unit* (TAU) to observe whether the user is meeting their respective daily quotas.

The *Diet Diary* scenario has been chosen since it requires many of the features discussed in Chapter 3, which the platform aimed to provide support for in Chapter 5. The scenario focuses on data entry, enabling the user to regularly inform the server of new food and activity items. For example, each time the user consumes a food item this process can be performed by providing the item's details. The user is able to input all the details of the item manually, including the quantity consumed and the calories of the item. Alternatively, the user can choose to select from a list of items stored by the scenario. This feature is in place to simplify the input process as the user would only be required to provide the quantity consumed.

The scenario utilises the platform's autonomous data analysis components to determine the user's chosen method and subsequently calculates the calorie intake from the food item. Data analysis is also performed to update the daily values of the user's *Diet Diary* instance. This information is sent to the user as feedback, notifying them on the amounts required to meet their daily targets. There is an alert feature in this scenario, which notifies the user if their daily calorie intake or burn targets are breached when new food or activity items have been sent to the server. An alert limit

can also be set when the user is nearing a calorie target value. This feature enables the user to be informed and then react to the situation. The autonomous analysis component is used to identify these target breaches on each occasion that the user's daily calorie values are updated.

The template mechanism that was discussed in Chapter 5, is utilised for each of these features. Template messages are created in the scenario's development process to cover each type of interaction between the server and the user. *Incoming* templates provide a means for data entry by the user, which includes providing initial diet details to update an instance when the user consumes a food item or performs an activity. From the server aspect, there are *Outgoing* templates to provide the necessary feedback to the user, either to simply confirm the successful entry of the user's message or to provide alerts on the nutritional values for the day. Feedback and alert messages sent to users are based on *Outgoing* templates which have been individualised with the relevant data, meaningful to the user. The scenario has a database developed and tailored directly to meet its characteristics. This database provides record-keeping of each user assigned to an instance, storing the data on food and activity items, weight goals, calorie targets and daily summaries. The TAU transfers the field data supplied by the user into the scenario's database.

The scenario has been designed to utilise the core components of the platform in order to provide the features discussed. The communication components implemented within the platform handles the interactions between the server and users assigned to *Diet Diary* instances. This has the effect that the user can communicate via SMS, email and the web-based user interface and switch easily between these three methods. The platform controls the sending out of messages from the server, determining the appropriate communication method to use. The platform also handles the retrieval of user messages, regardless of the communication method employed. There is no need to develop specific new communication handling components for the scenario.

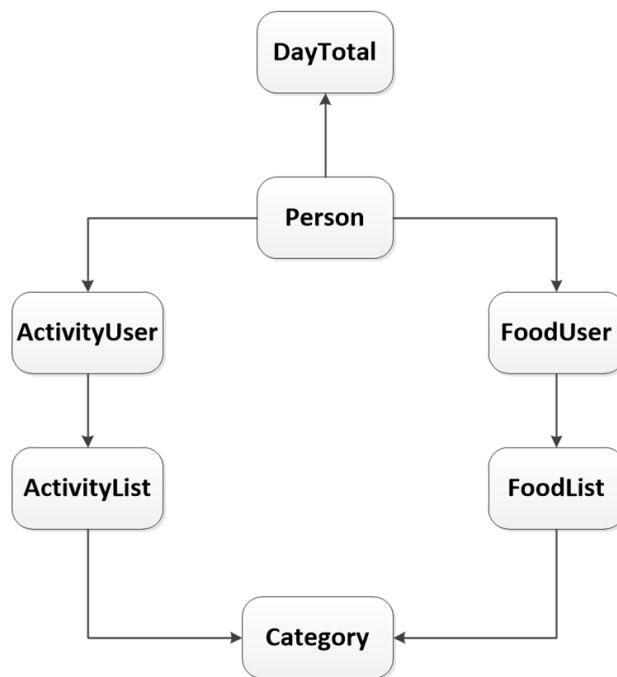
The platform's template mechanism is utilised by the scenario builder to create each *Incoming* and *Outgoing* template message that is to be used as part of the communication structure of the scenario. The builder defines the text and rules for each template tailoring them to the requirements of the scenario, via the web-based

user interface. The TAU analyses each input message, based on an *Incoming* template, which is sent from a user. This analysis is based on the template rules that have been defined by the builder. The data layer provides access to the *Principal* database for storage and retrieval of the scenario's templates, as well as users and messages for each instance.

## 6.3 Implementation of the scenario

### 6.3.1 Database entities

There are several components to the *Diet Diary* scenario. Following the three-stage scenario development process each of these components is defined as an entity of the scenario, whereby the scenario's database has been developed with a table to represent and store data for each entity. Figure 6.1 illustrates the scenario's database, signifying the relationships between the scenario's entities.



**Figure 6.1:** *Entities of the scenario's database*

The *Person* entity of each *Diet Diary* instance represents the user for whom an instance is created. There is a *Person* table within the scenario's database where there is a record for each user assigned to an instance, which stores details on a user's current attributes such as their height, weight and gender. Additionally, objective data are held within this table, which includes the daily calorie burn and intake targets, the



user's weight goal and the specified period of time to achieve this goal. Each *Person* record is linked to the *Instance* record within the *Instance* table of the *Principal* database, via the *Instance* identifier.

The *FoodUser* table is where details for each food item consumed by a user is stored, marked by the date and time of consumption. This table includes fields to store the data on a food item's nutritional values including the portion size, calorie intake and whether the item is a fruit or vegetable.

The scenario stores a list of food items in the *FoodList* table. The user is able to select from one of the items stored in this table. Each record holds nutritional data for a default amount of the item and defines the portion type. A portion of a food item could be measured by the weight or the quantity. These two types are referred to as portion weight and portion unit respectively. When the user selects the correct item they then provide the amount consumed. The consumed amount is used with the calorie per default amount to calculate the total calorie intake of the consumed item. The *FoodList* table is also expandable. In situations where the user inputs a new food item manually the details of the food item are then stored in the *FoodList* table for future occurrences.

The same process is used for providing details of activities performed. The *ActivityUser* table stores a record for each activity performed by the user, including data on the calories burned, total time and length of the activity, as well as the date and time it was performed. The *TotalTime* attribute in the *ActivityUser* table is used to calculate the total daily time. In addition to the daily calorie burn target the user can choose to have a daily activity time target. The user can choose from a list of activities contained in the *ActivityList* table. There is an attribute for the activity's *Metabolic Equivalent of Task* value (*MET*). This value is used to measure the strenuous factor of the activity in order to calculate the calories burned [189]. Some activities, such as running, have different calorie burn rates depending on the speed that the person is travelling [190]. Therefore, the user can select between different speed rates for these types of activities.

Food and activity items are both assigned to a category. The *Category* table stores the list of available categories for each type. When searching for their desired item, categories provide a means for the user to filter the results.

Details of the total daily nutritional and activity data on a user are stored in the *DayTotal* table. The current day's record is updated when new food and activity items are supplied by the user, storing the total calorie intake and burn values, the total activity time and the fruit and vegetable count for the day. The user is also able to update their daily weight with the new weight stored in the current day's record. The data within this record are used to assist the user in reaching their daily targets and show the weight movement from the instance's initialisation to the current time.

The tables within the scenario's database have been designed in order that the field data received from a user and the calculations from this data are stored in meaningful locations. The builder defines the location to store the data. The TAU can then utilise the data at a later stage for further calculations and for analysis checks when new field data is received.

### **6.3.2 Template and analysis criteria**

The next stage involves the creation of the scenario's templates together with the analysis criteria for parameter positions within the templates. In each phase of a *Diet Diary* instance the user is required to use an *Incoming* template to supply field data to the server. Therefore, the builder needs to create each *Incoming* template. Figure 6.2 illustrates the *Template* web-form of the web-based user interface being used to create an *Incoming* template. In this situation the template being created would request the user to input basic details for setting up their *Diet Diary* instance.

**Template Form**

Scenario: 7: Diet Diary  
 Show Templates for: Incoming  
☐ Select All  
 Delete Refresh

TemplateID	Name	Type
43	New Diet Diary Instance	0
58	Input Diet Details	0
61	Input Auto Targets	0
63	Input Manual Targets	0
65	Auto Food Search	0

1 2 3

Save New Delete Create/Edit Triggers Parameters

ID: 58

Name: Input Diet Details

Template Structure: Please fill in the following details:  
 Date of Birth: \*dd/mm/yyyy\*  
 Gender: \*male/female\*  
 Height: \*metres\*  
 Current Weight: \*kilograms\*  
 How to determine allowances/goals: \*recommended amounts/input manually\*

Parameter Count: 6

Response Template: None

Instance Creator: ☐ Yes ☒ No

Category: Setup

Allow Attachment: ☐ Yes ☒ No

**Figure 6.2:** Incoming template being created in the Template web-form

Templates are provided with a meaningful name in order for them to be identified. This enables the builder to link templates, within the communication structure, with the appropriate response templates. Users are able to request templates to provide further details once an instance has completed its initial setup stages. Therefore, a meaningful template name helps the user to locate simply the correct template for any data entry procedures. The builder inputs the text of the template in the “Template Structure” textbox, which is sent to users as the main body text. This includes declaring the template’s parameter positions, with temporary data placed between the two asterisks.

The category of the template is selected for *Incoming* templates. *Incoming* templates are split into three sub-types, *InstanceCreator*, *Setup* and *Available* templates. Each scenario may have one *InstanceCreator* template to enable the user to request the creation of a new instance. *Available* templates are for the user to be able to request at

any stage of an instance. The server utilises *Setup* templates to request information from the user, however these types cannot be requested by the user.

Once the template is saved the parameter data of the template is passed to the *Parameter-Input* web-form. This web-form, illustrated in Figure 6.3, takes the builder through each of the declared parameters, where it is possible to alter the temporary data, provide a meaningful name for the parameter and an expected data-type for the returned data from a user. The parameter name is used by the builder to locate the correct parameter when defining analysis criteria and actions in the *Answer* and *DataOut* entities respectively. The procedure for creating *Outgoing* templates is similar; however the template type does not need to be defined. Additionally, for *Outgoing* templates there is an option to provide a response template, which is selected from the list of *Incoming* templates. This would then be sent alongside the *Outgoing* template whenever it is sent to a user, supplying the user with the template structure for sending a reply back to the server. Figure 6.4 illustrates the view of the *Template* web-form for creating an *Outgoing* template.

**Add data to TemplateID58**

Please fill in the following details:

Date of Birth: \*dd/mm/yyyy\*

Gender: \*male/female\*

Height: \*metres\*

Current Weight: \*kilograms\*

How to determine allowances/goals: \*recommended\*

Template Text

Position: 1

Name: Date of Birth

Data: dd/mm/yyyy

Type: Date

Next Previous Return

**Figure 6.3:** *Parameter-Input web-form*

### Template Form

Scenario
7: Diet Diary

Show Templates for:
Outgoing

☐ Select All

Delete
Refresh

		TemplateID	Name	Type
<input type="checkbox"/>	Select	44	New Instance Created	1
<input type="checkbox"/>	Select	46	Diet Details Updated (Manual)	1
<input type="checkbox"/>	Select	48	Diet Details Updated (Auto)	1
<input type="checkbox"/>	Select	62	Confirmed Auto Targets	1
<input type="checkbox"/>	Select	64	Confirmed Manual Targets	1

1 2 3 4 5

Save
New
Delete
Create/Edit Triggers
Parameters

ID
46

Name
Diet Details Updated (Manual)

Template Structure

Thank you, your details have been updated. Your current BMI value is \*BMI\*, calculated based on your weight and height. This value indicates that you are \*Position\*.  
  
You will shortly receive a template to fill details on food intake preferences.

Parameter Count
2

Response Template
63: Input Manual Targets

Allow Attachment
☐ Yes ☒ No

**Figure 6.4:** Template web-form for creating an Outgoing template

Once this process has been completed the builder is able to define analysis criteria and create response actions for *Incoming* templates. Selecting the “Create/Edit Triggers” link takes the builder to the *Trigger* web-form. Figure 6.5 illustrates the *Trigger* web-form for the “Input Diet Details” *Incoming* template. In this situation two *Trigger* entities have been defined. The first trigger is activated if the user manually requests to provide their own targets. The second trigger is activated if the user chooses to have targets automatically generated.

### Trigger Form

Template ID 58

[Back to Template Form](#)

☐ SelectAll

Delete

		TriggerID	Name
<input type="checkbox"/>	Select	10	Manual Targets
<input type="checkbox"/>	Select	15	Auto-Calculate Targets

Answers      DataSave      Actions

New      Save      Reset      Delete

ID      10

Name      Manual Targets

Position      1

**Figure 6.5:** *Trigger web-form*

*Answer* entities are used to determine the correct trigger based on user data. Figure 6.6 illustrates the *Answer* web-form, where these *Answer* entities are created. The builder is able to select the comparison type and provide the expected answer. In this case the *Answer* entity would instruct the TAU to check if the data item contains the word “manual” within its textual structure. Each data item can be converted to a new value for data storage. The builder then defines the location in the scenario’s database to store each data item via the *DataSave* web-form. Figure 6.7 illustrates the *DataSave* web-form, whereby the builder has created a *DataSave* entity for each parameter position of the “Input Diet Details” *Incoming* template to update the *Person* table with a record for the new user.

## Answer Form

**TriggerID 10 for Template 58: Input Diet Details**

[Back to Trigger](#)
[Add Existing Answers](#)

Parameter Name (Position Order) All

☐ SelectAll

[Delete](#)

	AnswerID	Parameter Position	Name
<input type="checkbox"/> <a href="#">Select</a>	1	1	Date of Birth
<input type="checkbox"/> <a href="#">Select</a>	5	5	Goal Determination
<input type="checkbox"/> <a href="#">Select</a>	6	4	Weight
<input type="checkbox"/> <a href="#">Select</a>	7	2	gender
<input type="checkbox"/> <a href="#">Select</a>	8	3	Height

[New](#)
[Save](#)
[Reset](#)
[Delete](#)

ID 5

Parameter (Position) 5 (Goal Determination)

Data Type String

Comparison Type Contain

Obtain Answer From Database ☒ No ☐ Yes

Answer1 manual

Convert To (for database) False

Position 1

**Figure 6.6:** Answer web-form

## DataSave Form

### TriggerID 10 for Template 58: Input Diet Details

[Back to Trigger](#)  
☐ SelectAll  
[Delete](#)

		DataSaveID	Parameter Position
<input type="checkbox"/>	Select	1	1
<input type="checkbox"/>	Select	4	2
<input type="checkbox"/>	Select	5	4
<input type="checkbox"/>	Select	6	3

New

Save

Reset

Delete

ID

6

Parameter

3 (Height) ▼

Table Name

Person ▼

Column Name

Height ▼

Pending Entity

☒ None
 ☐ Create New
 ☐ Use Existing

Allow Multiple Users Update

☒ No
 ☐ Yes

Pending Record

**Figure 6.7:** DataSave web-form

Figure 6.8 illustrates the *Action* web-form where the builder creates *Action* entities. These entities are used by the *ActionHandler* class to enable the TAU to respond to user messages. The list of items displayed by the “Action Type” dropdown-list illustrates the different possible action types available to the builder. Depending on the action type chosen the builder is required to provide extra attribute details for the *Action* entity. The *Action* entity created in Figure 6.9 is a *Reply* action, which would instruct the *ActionHandler* class to send a reply message to the user. The appropriate *Outgoing* template is selected for the *Reply* action. In this example the *Outgoing* template, which was shown in Figure 6.4, is selected.



ID	
Action Type	InstanceNew ▼
Response Group	InstanceNew
Set new instance's status as	StatusChange
Parameter to use for Name	AssignAdd
Proc String	AssignRemove
Check Pending	Reply
	Forward
	MessageSend
	NewsInsert
	Alert
	TemplateList
	Notification
	UserUpdate
	UserAdd

**Figure 6.8:** “Action Type” dropdown-list in Action web-form displaying a list of available Actions

**Action Form**

Trigger ID 10

[Back to Trigger](#)

☐ SelectAll

[Delete](#)

	ActionID	Action Type
<input type="checkbox"/> <a href="#">Select</a>	30	Reply

[Data Out](#)  
[Parameters](#)

[New](#) [Save](#) [Reset](#) [Delete](#)

ID:

Action Type:  ▼

Response Group:  ▼

Outgoing Template:  ▼

Proc String:

**Figure 6.9:** Reply Action entity created in the Action web-form

Finally, the builder assigns the data that is provided in the parameter positions of the *Outgoing* template. Figure 6.10 illustrates the *DataOut* web-form. The column *BMI* of the *Person* table has been selected for the first parameter position.

## DataOut Form

Action ID 30

[Back to Action Form](#)

		DataOutID	ParameterID
<input type="checkbox"/>	Select	67	130
<input type="checkbox"/>	Select	68	131

[Save](#)
[New](#)
[Reset](#)
[Delete](#)

ID

Parameter Position

Data Provider

Table Name

Table Column

**Figure 6.10:** DataSave web-form

The remaining *Incoming* templates and their analysis entities are created using the same steps in the web-based user interface. This process demonstrates that the template creation and the defining of analysis criteria and response actions is achieved declaratively, without any programming effort required by the builder.

### 6.3.3 Application module development

The *Diet Diary* scenario's application module contains the algorithms to calculate various values throughout the operations of an instance. The builder defines and places each algorithm within the methods of the module. The method that is run depends on the *Trigger* entity that has been activated by the TAU on the analysis of an *Incoming* template. The *Trigger* identifier is passed to the module and used as the key parameter of a *switch case* statement to determine the correct method, within the module, to run. The module has access to the scenario's database to make any necessary updates to records.

The builder has developed an algorithm to calculate the *Body Mass Index* (BMI) of the user, which is a measure of their body fat based on the weight and height from

their initial setup details [191]. There are four categories within the BMI index; underweight, normal, overweight and obese. The calculated value is tested against the boundary limits of each of these categories to determine the matching category. The BMI value and the user's category are stored in the *Person* record of the scenario's database. These values can then be inserted into the parameter positions of the *Outgoing* template that is sent as feedback to the user.

There is an algorithm to calculate the daily calorie targets of a user based on the weight goal that the user is aiming to achieve. This algorithm uses the default overall calorie loss amount needed to lose one kilogram of weight weekly [192]. Then together with the *Recommended Daily Allowance (RDA)* [193] of calorie intake for a typical person and the timeframe to reach the new weight the algorithm calculates both the calorie intake and burn targets required daily.

A food item that a user selects from the list of items stored in the *FoodList* table has a default calorie value, which is either measured by a portion unit of one or a portion weight of 100 grams. When the user provides data on the portion size consumed, this value is then used to calculate the total calorie intake of the food item. Similarly for activities, the user is only required to provide the time spent if the activity is selected from the scenario's activity list. Each activity has a unique *MET* value which is then used to calculate the calories burned in the activity. For both food and activities where items have been provided manually by the user, the module has algorithms to calculate the default calorie or MET values respectively. These calculated values are based on the total calorie values provided by the user. For each of these algorithms the calculated values are stored in the *FoodList/ActivityList* tables that are accessible by either the TAU or the application module during a later phase of the active instance.

Daily summary details are calculated each time a new food or activity item is added. The calorie value and other attributes including whether a food item is a fruit or vegetable, as well as the time taken for an activity item are all passed to the *DaySummary* algorithm. The current daily calorie intake and burn values are updated in this algorithm. Furthermore, to assist with providing alerts to users the span between the calorie intake and alert limit (*IntakeToAlert* field) is updated. Using the calculated *IntakeToAlert* value each time a user adds a new food item an *Answer*

entity checks that the calorie intake from the item would not push their daily calorie intake over the alert limit. A similar method is used for activities, with the updated value stored in the *BurnToAlert* field of the *DayTotal* record.

Figure 6.11 illustrates the “Daily Summary” *Outgoing* template, which utilises these values. Using *DataOut* entities, the builder associates each parameter position with the correct database location in the *DayTotal* table. Whenever the user requests the daily summary, these values are sent to them in a readable format.

ID	85
Name	Daily Summary
Template Structure	<pre> Calorie Intake: Current is *calories*cal. Goal is *calories*cal. *above/below* by *calories*cal.  Calorie Burn: Current is *calories*cal. </pre>
Parameter Count	17
Response Template	None

**Figure 6.11:** “Daily Summary” *Outgoing* template

These situations show that the application module is connected with the TAU, via the scenario’s database, to enable calculated data to be utilised by the core components of the platform. The programming effort for the application module is minimal as the builder is only required to develop algorithms for the calculations of a *Diet Diary* instance. Each algorithm is described in further detail in Appendix A1.

#### 6.4 Development time comparison for the *Diet Diary* scenario

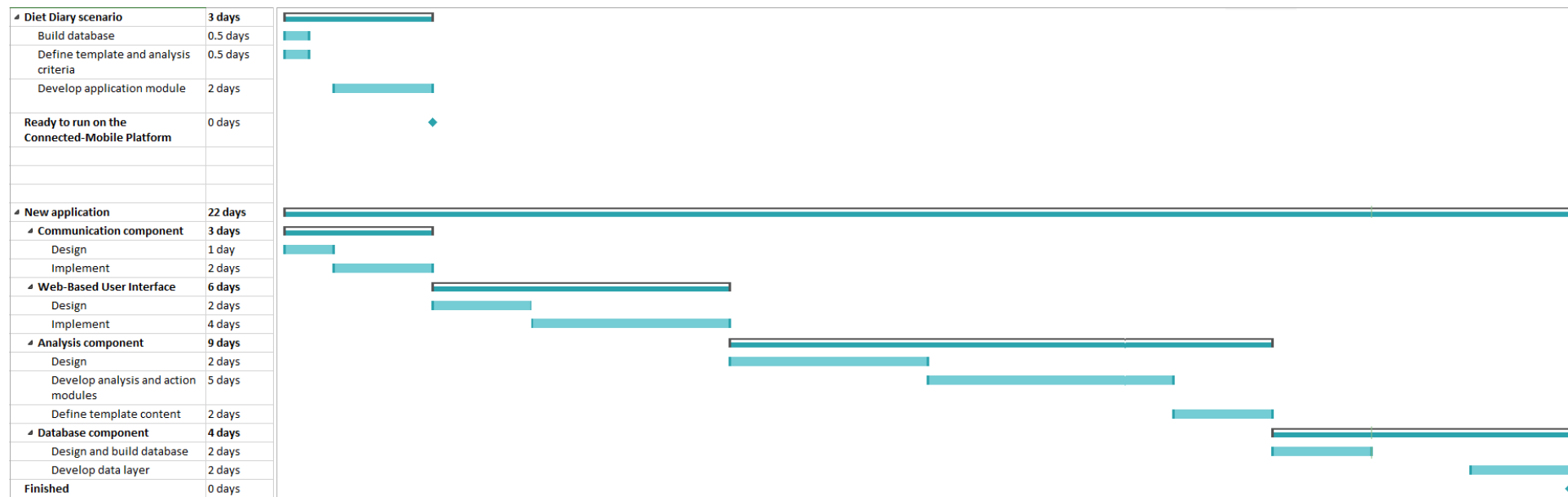
Following the three-stage scenario development process, the *Diet Diary* scenario was designed and implemented in 3 days. Half a day was required for designing and assembling the tables of the scenario’s database in *Microsoft SQL Server Management Studio*. Another half day consisted of defining the template content and analysis criteria for the scenario. In this case study only the development of the application module had a larger timeframe to complete, where 2 days of coding were required. The platform’s TAU does not contain the functionality to support the algorithms that are required to perform the calculations of the *Diet Diary* scenario. The application module consists of a C# class that was developed in the *Microsoft*

*Visual Studio IDE*, which stores the methods for running each algorithm. After completing these three stages the platform was ready to generate and run new instances of the *Diet Diary* scenario.

Creating an application from scratch that would achieve the same functionality as the *Diet Diary* scenario involves a far longer development time. The timeframes summarised below for developing each component of this mock-up application are estimates based on a design specification of the application that outlines the classes and methods required to meet its objectives. The next case study in Chapter 7 (Section 7.4) contains extended details on the comparisons between producing a scenario via the platform and developing a new mock-up application to fulfil the same objectives.

A communication component would need to be designed uniquely for this new application in order to handle the sending and receiving of messages. This component would require 3 days of development. A web-based user interface would need to be included for operator maintenance and user communication via this means, taking approximately 6 days to develop. A message analysis component would take an estimated 7 days to design and implement into the application, with a further 2 days for structuring and defining the template content. Finally, a database would be created to store data regarding all the entities of application, including each user's diet diary, template content and the messages sent between the application and its users. The design of the database and coding of the application's data layer for accessing this database would take 4 days to complete.

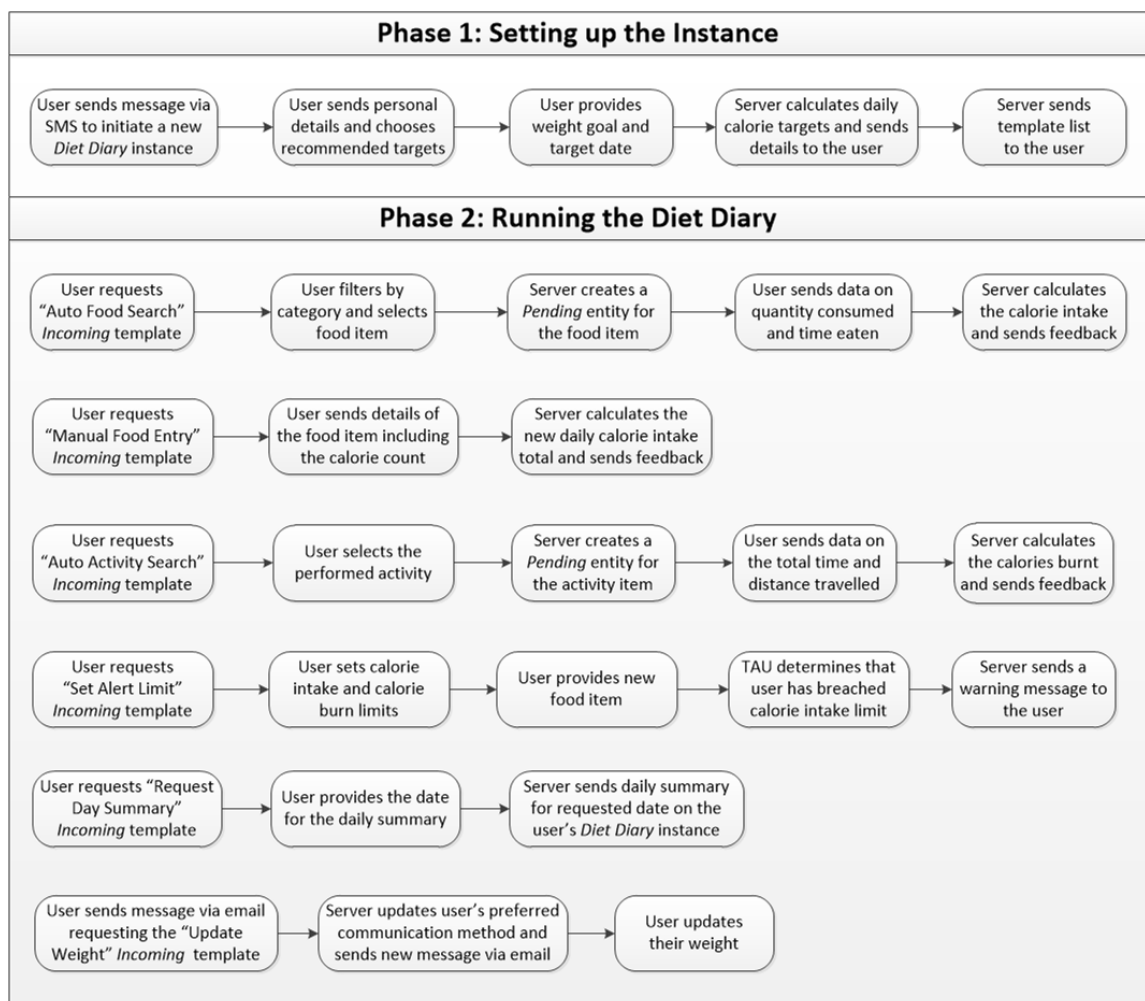
Overall, it would take an estimated 22 days to complete the development of a new application with equivalent functionality to the *Diet Diary* scenario. This emphasises that the platform facilitates a much faster timeframe for producing new types of applications due to the framework of ready to use generic components at its disposal. Only the application module required more than a day to develop, however this was minimal in comparison to the overall completion time for a new application. Figure 6.12 illustrates the comparison between completion times for implementing the *Diet Diary* scenario and developing a new application from scratch.



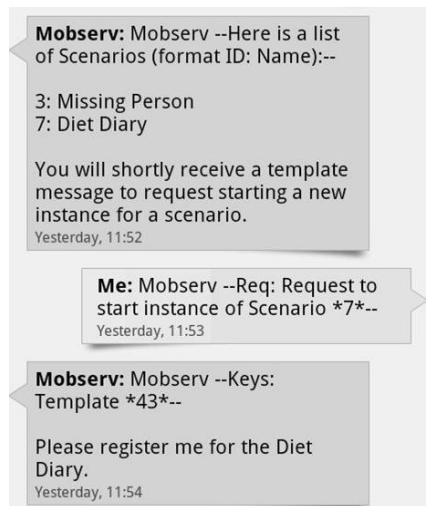
**Figure 6.12:** Comparison of timeframes between developing the Diet Diary scenario using the platform (top) and developing a new application from scratch (bottom)

## 6.5 Running a *Diet Diary* instance

The flow diagram in Figure 6.13 demonstrates an example of the sequence of events performed throughout an active *Diet Diary* instance. A user of the platform can request to initiate an instance of the *Diet Diary* scenario, as illustrated by the second message in Figure 6.14. The third message in Figure 6.14 shows the server response, which contains the scenario's *InstanceCreator* template. This is followed by the user replying with a message based on the *InstanceCreator* template to confirm that they want to start a *Diet Diary* instance. The TAU responds by creating the new instance.

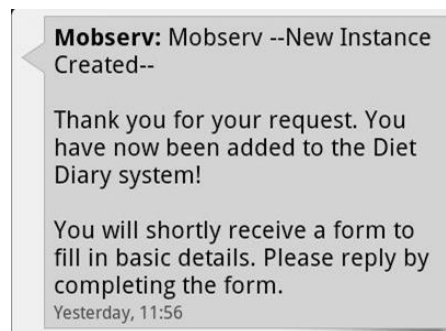


**Figure 6.13:** Sequence of events performed throughout an active *Diet Diary* instance



**Figure 6.14:** SMS message exchange to start a new *Diet Diary* instance (For SMS: Server messages begin with “Mobserv”, User messages begin with “Me”)

A feedback message, illustrated in Figure 6.15, is sent from the server to acknowledge that the new *Diet Diary* instance has been created. This message uses an *Outgoing* template specific to the *Diet Diary* scenario. The TAU can simply switch between sending messages that are based on platform-wide *Request* templates and scenario-specific *Outgoing* templates, depending on the data received from the user.

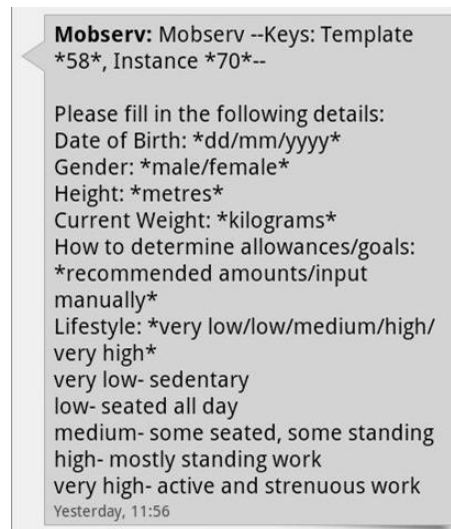


**Figure 6.15:** Feedback message sent from the server to confirm a new instance has been created

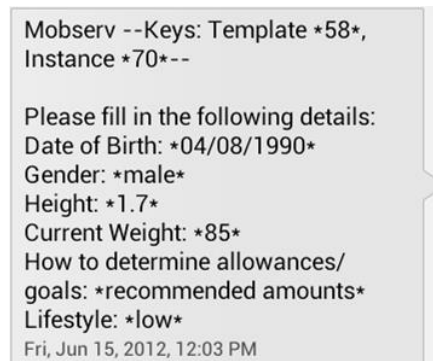
The TAU guides the user through a number of steps to gather the necessary details for their *Diet Diary* instance by using a combination of *Outgoing* and *Incoming* templates. This starts with obtaining personal details about the user. Figure 6.16 shows the *Incoming* template supplied to the user to collect their personal details. The *Template* and *Instance* identifiers are prefilled by the TAU to simplify the input requirements. The user is only required to replace the temporary data between each



set of asterisks of a parameter position with the field data. The temporary data is provided to assist the user in their response. An example of this is where the user can select their lifestyle in the last parameter position of the template. This is the user's normal activity level for the day, which can range from a low level sedentary to a high level active lifestyle [194]. Each level has an associated calorie burn value that acts as the baseline burn value each day. An explanation is provided for each possible choice, which is displayed below the *Lifestyle* parameter position. The user can delete the items that do not match their lifestyle, leaving the valid answer. Figure 6.17 illustrates the input message sent by the user that is based on this *Incoming* template, where each parameter position has been filled in with the user's personal details.

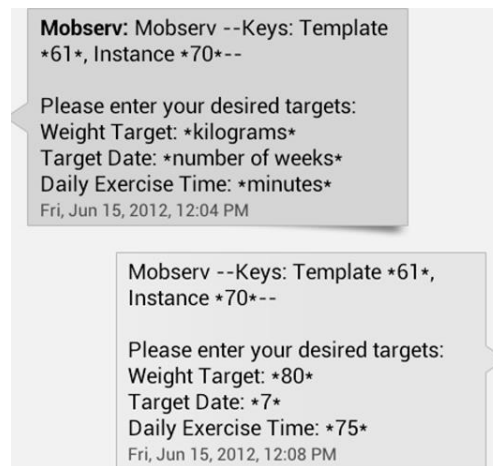


**Figure 6.16:** Incoming template message provided by the server for the user to enter their personal details



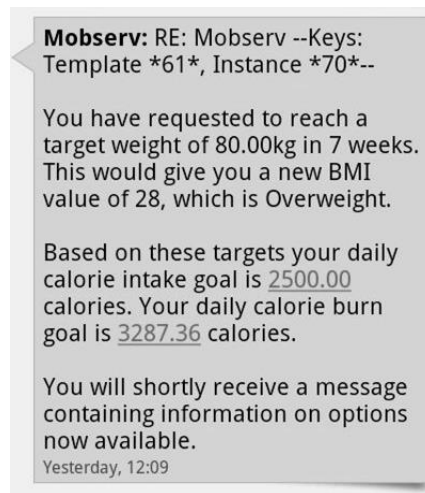
**Figure 6.17:** Input message sent by the user to provide their personal details

Within this template the user is given a choice of the method to determine their daily calorie intake and burn targets. These can either be inputted manually by the user or calculated by the TAU based on the user's desired weight goal and timeframe. Figure 6.18 shows the *Incoming* template message sent to the user if recommended daily calorie targets (calculated by the TAU) have been selected, followed by the user's reply containing their weight goal. Alternatively, if the user chooses to manually input their own daily targets then the TAU responds with the *Incoming* template in which the user can provide these targets.



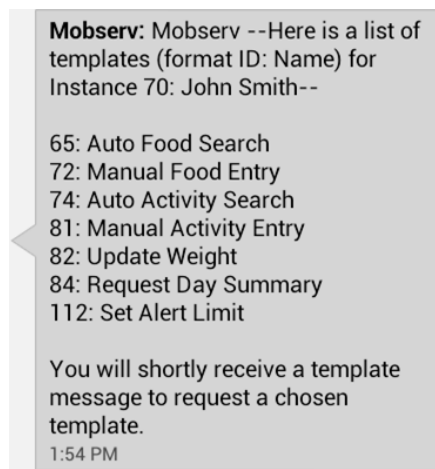
**Figure 6.18:** *Incoming template message to provide a weight goal and timeframe, followed by the user's input message based on this template*

The scenario's application module contains the algorithm to calculate the daily calorie intake and daily calorie burn targets, using the current and target weight data that has been sent by the user over the previous few messages. After the application module has completed its processes the TAU retakes control of the message analysis and sends a reply message to the user as feedback (Figure 6.19). The *DataOutHandler* class of the TAU uses the data provided by the application module for responding to the user, filling in the parameter positions of the message.



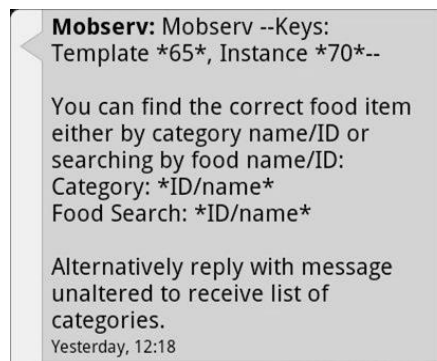
**Figure 6.19:** Feedback message sent to the user to provide details of their daily calorie targets

The TAU updates the *Diet Diary* instance on each occasion that a user adds details of a food item consumed or an activity performed. The message sent from the server, illustrated in Figure 6.20, displays a list of the available templates that would enable the user to provide these details. The user has two choices when they provide details of each food item that they have consumed, manually providing the item's details or selecting from a list of available items in the *FoodList* table. During the scenario's development process the builder inserts multiple records into the *FoodList* table for a range of food items. Each record contains attributes to hold nutritional data on an item, including the calories contained within a standard portion size and the item's relevant food category.



**Figure 6.20:** List of Incoming templates available for a user to request

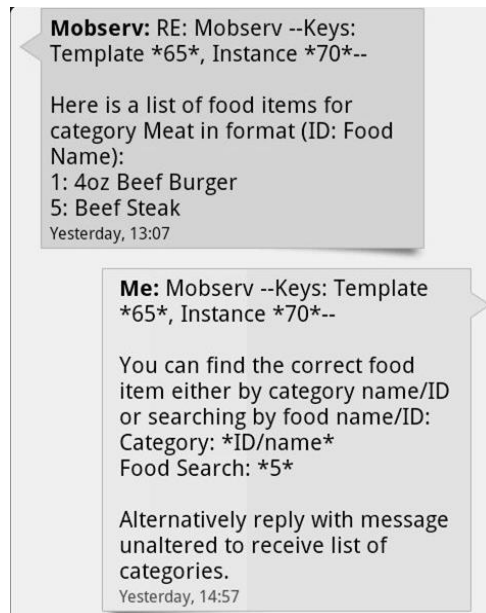
The user can select a food item by requesting the “Auto Food Search” *Incoming* template in Figure 6.21. In this template the user is given the choice of requesting a list of categories, food items within one of the categories or directly searching for a food item by its name or record identifier. This template provides an example of the TAU performing a different response, depending on the data provided by the user. The user is able to maintain a dialogue with the server by drilling down to a particular item, starting from a list of the categories.



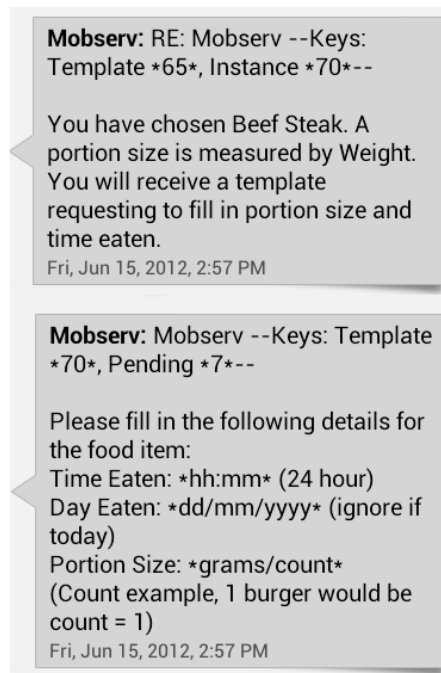
**Figure 6.21:** Incoming template to search for a food item from the FoodList table

Figure 6.22 shows the message returned to the user if items from the “Meat” category have been requested, followed by the user’s reply with their selection. Once the user selects a food item, the server sends a feedback message confirming the chosen item. The server also sends an *Incoming* template to enable the user to send back data on the quantity consumed and the time of occurrence. Both these messages are illustrated in Figure 6.23, where the user has selected the food item “Beef Steak”. Data within this message is related to the food item selected during the previous exchange of messages. To keep track of this relationship a *Pending* entity is created. Once the “Auto Food Search” *Incoming* template in Figure 6.22 has been filled in by the user and sent back to the server, a new record is created in the *FoodUser* table of the scenario’s database to store the details of the food item. The *Pending* entity collects the identifier information of this record to enable future updates. These steps are carried out based on the rules created by the builder when defining the *DataSave* entities of the “Auto Food Search” *Incoming* template (Figure 6.24). The follow-up message sent to the user contains the *Pending* entity’s identifier in the message’s title. Therefore, the user can send multiple messages to the server, regarding different food items, with each message assigned to the correct food item based on the *Pending*

identifier supplied. When all the details of a consumed food item have been provided they are then passed to the application module, which calculates the calorie intake of the item and updates the user's total daily calorie intake. The recently created record in the *FoodUser* table is located using the data within the *Pending* entity. This record is then updated with the calorie intake value.



**Figure 6.22:** Output message from the server providing a list of food items in the chosen category, followed by the user's reply with "Auto Food Search" Incoming template

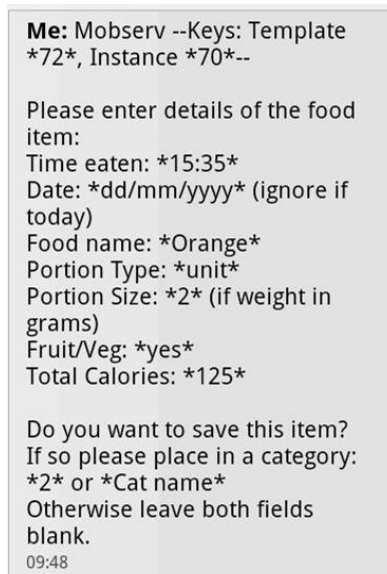


**Figure 6.23:** Output message from the server to confirm the chosen food item, followed by Incoming template for the user to provide the relevant data on the item

New	Save	Reset	Delete
ID		2	
Parameter		2 (food search) ▼	
Table Name		ConsumedItem ▼	
Column Name		FoodID ▼	
Pending Entity		<input type="radio"/> None <input checked="" type="radio"/> Create New <input type="radio"/> Use Existing	
Allow Multiple Users Update		<input checked="" type="radio"/> No <input type="radio"/> Yes	
Pending Record			

**Figure 6.24:** DataSave entity that instructs DatabaseHandler class to create a new Pending entity

A user is required to fill out the “Manual Food Entry” *Incoming* template in Figure 6.25 if they choose to manually supply details of a food item. This template requests all the necessary data concerning the food item, including its nutritional data and the amount consumed. The user is also provided with the option of saving the food item for future use by providing the relevant food category. This demonstrates an example of where the user can directly contribute to the data held on a scenario, with the *FoodItem* table expanding its contents as each new item is added.

The image shows a screenshot of a text-based interface, likely a chat or messaging app, with a light gray background. The text is black and follows a structured template for adding a new food item. It starts with a header line, followed by a request for details, then several fields with placeholder values in asterisks. The final part asks for a category and includes a timestamp at the bottom left.

**Me: Mobserv --Keys: Template**  
**\*72\*, Instance \*70\*--**

Please enter details of the food item:

Time eaten: \*15:35\*

Date: \*dd/mm/yyyy\* (ignore if today)

Food name: \*Orange\*

Portion Type: \*unit\*

Portion Size: \*2\* (if weight in grams)

Fruit/Veg: \*yes\*

Total Calories: \*125\*

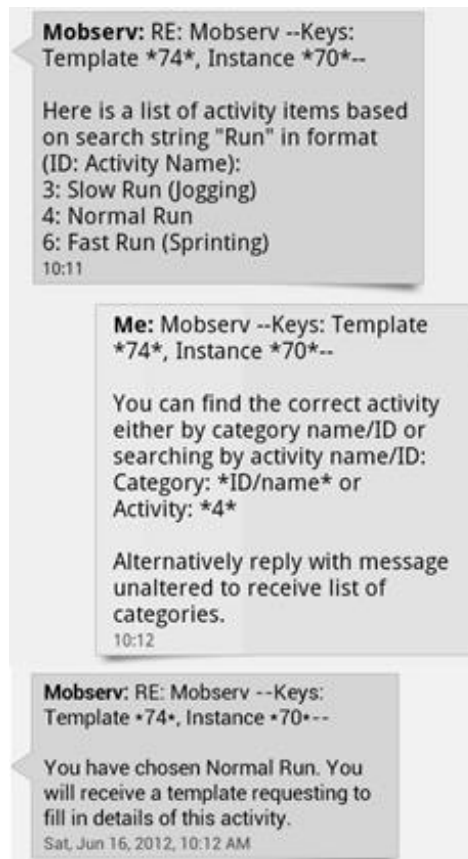
Do you want to save this item?  
If so please place in a category:  
\*2\* or \*Cat name\*

Otherwise leave both fields blank.

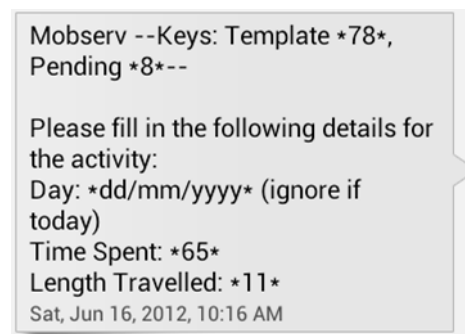
09:48

**Figure 6.25:** *Input message based on Incoming template to manually provide data on a new food item*

The user performs a similar procedure when adding new activity items to their *Diet Diary* instance. They are offered the choice of selecting an activity from the list of items available within the *ActivityList* table or adding a new activity manually. Figure 6.26 illustrates the user selecting one of the activities that has been returned from a search of the *ActivityList* table. This is followed by the details, sent from the user, of the time spent and length travelled in Figure 6.27. This data enables the application module to calculate the calories burned during the activity and then update the user's total daily calorie burn.



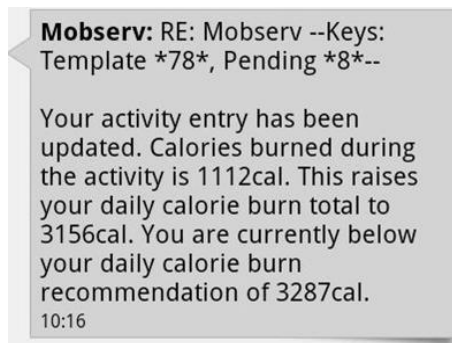
**Figure 6.26:** Message exchange to search for an activity



**Figure 6.27:** Input message based on Incoming template to provide data on the activity performed

On each occasion that users enter details of a new food item consumed or activity performed, feedback is provided on the user's current daily total and their distance to the daily target. Figure 6.28 shows an example feedback message sent from the server after the user has performed an activity.



A screenshot of a text message from Mobserv. The header is "Mobserv: RE: Mobserv --Keys: Template \*78\*, Pending \*8\*--". The body text says: "Your activity entry has been updated. Calories burned during the activity is 1112cal. This raises your daily calorie burn total to 3156cal. You are currently below your daily calorie burn recommendation of 3287cal." The time "10:16" is at the bottom left.

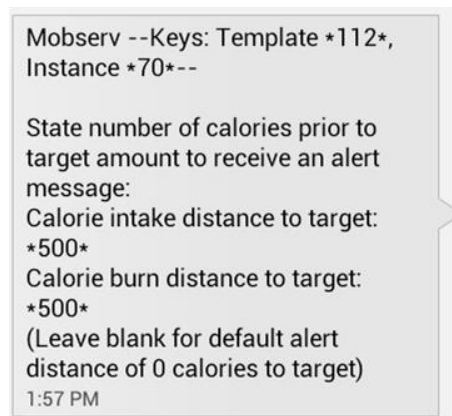
**Mobserv:** RE: Mobserv --Keys: Template \*78\*, Pending \*8\*--

Your activity entry has been updated. Calories burned during the activity is 1112cal. This raises your daily calorie burn total to 3156cal. You are currently below your daily calorie burn recommendation of 3287cal.

10:16

**Figure 6.28:** Feedback message to update the user on their daily calorie burn value.

The user may request the “Set Alert Limit” *Incoming* template to set new alert limits, permitting the server to warn the user before a daily target is breached. The user can set an alert limit for both their calorie intake and calorie burn values, as illustrated in the message in Figure 6.29. In Figure 6.30 the server responds with details of the actual calorie values that need to be breached for a warning message to be generated, which is calculated using the daily target data.

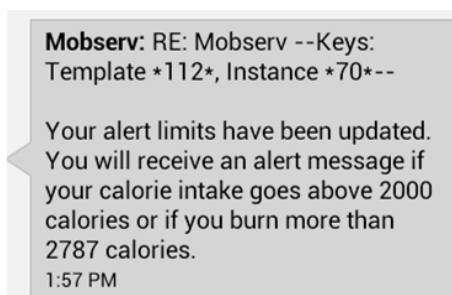
A screenshot of a text message from Mobserv. The header is "Mobserv --Keys: Template \*112\*, Instance \*70\*--". The body text says: "State number of calories prior to target amount to receive an alert message: Calorie intake distance to target: \*500\* Calorie burn distance to target: \*500\* (Leave blank for default alert distance of 0 calories to target)". The time "1:57 PM" is at the bottom left.

Mobserv --Keys: Template \*112\*, Instance \*70\*--

State number of calories prior to target amount to receive an alert message:  
Calorie intake distance to target:  
\*500\*  
Calorie burn distance to target:  
\*500\*  
(Leave blank for default alert distance of 0 calories to target)

1:57 PM

**Figure 6.29:** Input message based on *Incoming* template to provide alert limits

A screenshot of a text message from Mobserv. The header is "Mobserv: RE: Mobserv --Keys: Template \*112\*, Instance \*70\*--". The body text says: "Your alert limits have been updated. You will receive an alert message if your calorie intake goes above 2000 calories or if you burn more than 2787 calories." The time "1:57 PM" is at the bottom left.

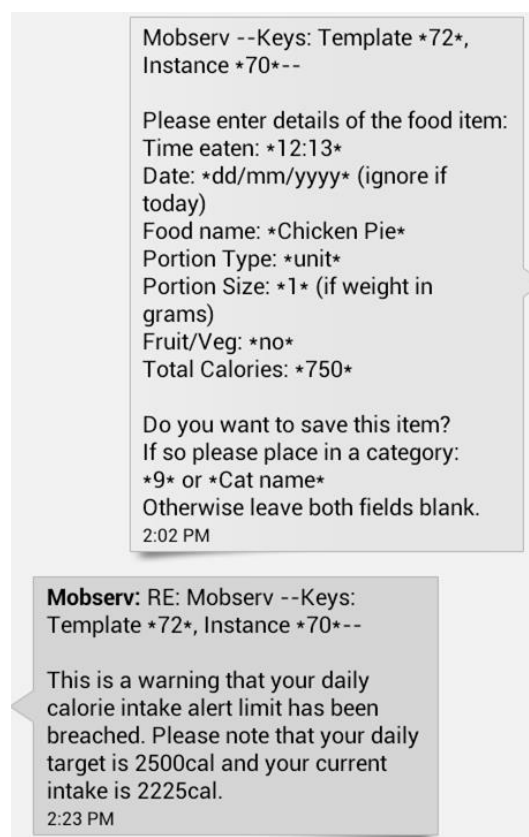
**Mobserv:** RE: Mobserv --Keys: Template \*112\*, Instance \*70\*--

Your alert limits have been updated. You will receive an alert message if your calorie intake goes above 2000 calories or if you burn more than 2787 calories.

1:57 PM

**Figure 6.30:** Feedback message detailing the new alert limit(s)

Figure 6.31 illustrates a process that would result in the user receiving a warning message to inform them that an alert limit has been breached. In this example the user has provided details of a new food item that they have consumed. The calories from this item have pushed the user over the alert limit that was previously set. The warning message instructs the user that this limit has been breached and informs them on how near the current calorie intake is to the daily target. Each time a new food or activity item is added, the *AnswerHandler* class determines whether an alert should be triggered. The builder creates an *Answer* entity (Figure 6.32) to calculate whether consumption of the new food item takes the user's daily calorie intake value over the defined limit. This entity instructs the *AnswerHandler* class to collect the *IntakeToAlert* value and compare it to the new food item's calorie value to determine the result. This process shows how the TAU can be utilised to provide alerts and reminders to users of the platform.



**Figure 6.31:** Message exchange that results in the server sending an alert to the user

	New	Save	Reset	Delete
ID	10			
Parameter (Position)	7 (calories)			
Data Type	Decimal			
Comparison Type	Greater			
Obtain Answer From Database	<input type="radio"/> No <input checked="" type="radio"/> Yes			
Answer1 Table	DayTotal			
Answer1 Column	DistanceToIntakeAlert			
Convert To (for database)	<input checked="" type="radio"/> TextBox <input type="radio"/> True <input type="radio"/> False			
Position	1			

**Figure 6.32:** Answer entity to calculate if a new food item has breached the alert limit

The user can retrieve daily summaries at any time. This is achieved by sending the “Request Day Summary” *Incoming* template in Figure 6.33. The server responds with a message containing the daily summary (Figure 6.34), providing detailed information on the user’s calorie intake and burn amounts thus far. The daily information is updated and stored in the *DayTotal* table with a record for each day, which allows the user to also view a summary of previous days. The fields in this table are associated with parameter positions of the *Outgoing* template that the message is based on, with the use of *DataOut* entities. To generate the message, the *DataOutHandler* of the TAU collects the data from the *DayTotal* record.

```

Me: Mobserv --Keys: Template *84*,
Instance *70*--

Summary Date: *16/06/2012*
11:23

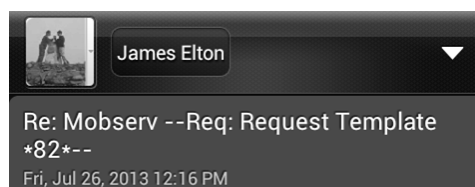
```

**Figure 6.33:** Input message based on *Incoming* template to request a summary of the current day

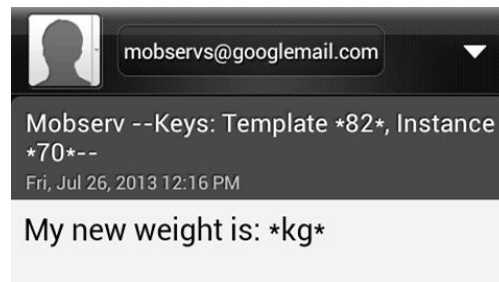


**Figure 6.34:** Output message sent to the user containing the requested day's summary information

The process of switching between communication methods is initiated by the user. In Figure 6.35 the user is requesting an *Incoming* template to provide their updated weight. In this example, the user has switched communication methods, choosing to send the message via email. When the message is processed by the TAU it changes the user's preference to email. This would ensure that any future messages from the server would be sent via this new communication method, which would apply until the user changes to another method. An example of this is illustrated in Figure 6.36 where the server has sent the requested *Incoming* template via email.



**Figure 6.35:** New message sent from the user via email to request the “Update Weight” Incoming template



**Figure 6.36:** *Email response by the server matching the user’s communication method*

The *UserMessageView* web-form of the web-based user interface enables a user to view each message that they have sent or received, regardless of the communication method. This is illustrated in Figure 6.37, where the user is viewing the message that had just been sent from the server (via email in Figure 6.36) containing the “Update Weight” *Incoming* template. The user can move back and forth between messages, using the “Previous” and “Next” links. Clicking on the “Template Response” link takes the user to the *UserMessageSend* web-form, illustrated in Figure 6.38. The “Title” and “Text” textboxes have been prefilled with the relevant data, based on the “Update Weight” *Incoming* template and the current instance. The user is only required to edit the text of the message through inserting their new weight in the parameter position. When the user clicks on the “Send Message” link this results in the TAU processing the message and then performing any necessary actions.

User Message View

User: John Smith (ID: 60)

Message Type

Received

Instance

Instance 70: John Smith (Diet Diary)

Date

07/26/13 12:16:49

Title

Mobserv --Keys: Template \*82\*, Instance \*70\*--

Text

My new weight is: \*kg\*

Message Type

Received

Previous

Next

Latest

Send New

Template Response

**Figure 6.37:** *UserMessageView* web-form displaying a message the user has recently received

User Message Send

User: John Smith (ID: 60)

Instance:

70: John Smith (Diet Diary)

Template:

82: Update Weight

Title

Mobserv --Keys: Template \*82\*, Instance \*70\*--

Text

My new weight is: \*82\*

Reset

Send Message

Message View

**Figure 6.38:** *UserMessageSend* web-form enabling the user to send a new message based on the “Update Weight” Incoming template

The user is returned to the *UserMessageView* web-form, where they can view any new messages that the server has sent, as illustrated in Figure 6.39. Selecting the “Send New” link takes the user to the *UserMessageSend* web-form. In this situation, the user is able to manually select the instance and *Incoming* template for sending a

new message. Figure 6.40 illustrates the *UserMessageSend* web-form, whereby the user is selecting a template from the list of *Available* templates for the *Diet Diary* scenario. Once the template has been selected, the message's title and text are generated and inserted into the "Title" and "Text" textboxes respectively. Messages sent via the *UserMessageSend* web-form result in replies that are only viewable via web-based user interface. It is the user each time who determines the method of communication, whether it is via SMS, email or the web-based user interface.

The screenshot displays a web-form titled "User Message View". Below the title, it identifies the user as "User: John Smith (ID: 60)". The form contains several fields: "Message Type" is a dropdown menu set to "Received"; "Instance" is a dropdown menu set to "Instance 70: John Smith (Diet Diary)"; "Date" is a text field showing "07/26/13 12:19:24"; "Title" is a text field showing "RE: Mobserv --Mobserv --Keys: Template \*82\*, Instance \*70\*----"; "Text" is a large text area containing the message "Your new weight has been updated on the system."; and another "Message Type" dropdown menu at the bottom, also set to "Received". At the bottom of the form, there are four links: "Previous", "Next", "Latest", and "Send New".

**Figure 6.39:** *UserMessageView* web-form displays the new message sent from the server

User Message Send

User: John Smith (ID: 60)

Select Instance:

Instance 70: John Smith (Diet Diary)

Select Template:

None

None

Template 65: Auto Food Search

Template 72: Manual Food Entry

Template 74: Auto Activity Search

Template 81: Manual Activity Entry

Template 82: Update Weight

Template 84: Request Day Summary

Template 112: Set Alert Limit

Title

Text

Reset

Send Message

Message View

**Figure 6.40:** The user is able to select the appropriate Incoming template when sending new messages in the UserMessageSend web-form



Feature	Supported by the Platform	Use in scenario	Generic provision from the Platform	Additions/changes required
User can send field data	Yes	Field data used to add new food/activity items.	<i>TemplateHandler</i> class and <i>Template</i> entities extract field data from user messages.	None, platform provided full functionality
User can send/receive feedback	Yes	Feedback on newly added items and daily summaries. New requests by the user, such as updates on daily summary.	<i>ActionHandler</i> and <i>DataOutHandler</i> classes create feedback messages based on <i>Outgoing</i> templates.	None, platform provided full functionality
Exchange of pictorial data	N/A	Not required by scenario		
Alerts and reminders	Yes- Partial	User can set calorie alert limit. Server warns user when limit is breach.	<i>AnswerHandler</i> class analyses new calorie data to determine if alert limit is breached. <i>ActionHandler</i> class sends warning message to the user.	New application module required to calculate total calorie values.
Database/record keeping	Yes- Partial	Field data sent from users and analysis results are stored for later use. Includes set-up data for weight goals and daily targets, new activity/food items added by user.	<i>Principal</i> database stores records on users, messages, <i>Incoming/Outgoing</i> templates and analysis criteria.	New scenario database required to store field data in meaningful locations. Includes records on the user's targets and for each food/activity item added.
Templates for individualised messages	Yes	User sends field data for set-up details, daily food/activity items and scenario requests. Server instructs user throughout the process of adding new food/activity items, updates user on new calorie totals and responds to user requests.	<i>Incoming</i> templates utilised by user to create input messages for sending field data and making requests. <i>Outgoing</i> templates utilised by server to create output messages for feedback.	None, platform provided full functionality
Autonomous data analysis	Yes- Partial	Processing requests and user decisions, such as opting for manual or automatic targets. Calculating calorie values for food/activity items and daily totals.	<i>TriggerHandler</i> and <i>AnswerHandler</i> classes used to analyse each message.	New application module required for calculations when new food/activity items are added by user.
Anonymous sending	N/A	Not required by scenario		
Collaboration/coordination of users	N/A	Not required by scenario		
Location-based features	N/A	Not required by scenario		

**Table 6.1:** Diet Diary scenario – Features supported by the Platform

## 6.6 Conclusions

The important factors in developing the *Diet Diary* scenario have been to illustrate and describe how the platform operates. This includes demonstrating the platform's functionality in supporting both the rapid development of a typical scenario and the running of an active instance. This instance is run to test the interactions between the platform and a single user, who is able to communicate via multiple communication methods. The *Diet Diary* scenario utilises a large number of the features that were identified and discussed in Chapter 3 and then incorporated into the design and implementation of the proposed platform in Chapter 5. Table 6.1 summarises the list of features, detailing their use within the scenario and the extent to which the generic functionality of the platform supports each feature.

The scenario focuses on one-to-one communication between the server and a user in order to demonstrate the communication structure between both parties. The communication component together with the template mechanism were effective in providing the means for the server to interact with a user. *Incoming* templates guide the user to insert the proper data in the correct locations, making it straightforward for the user to send the required field data at each stage of an instance. This is shown by the steps of adding a food item from the scenario's list, whereby the user is provided with instructions at each step within the *Incoming* templates. The server's feedback messages, created from *Outgoing* templates, offer meaningful information based on the user's actions. This is shown by the output messages from the server once a food or activity item has been added and also on occasions where the user requests the daily summary of their diet diary.

All three communication methods were tested in the example run of this scenario, with the switching between the methods being seamless. The user controls the method of communication, with the server updating the user's preference and matching their method each time a message is received. The functionality within the *MailHandler* and *TemplateHandler* classes is able to efficiently collect messages, with the extraction of the message details and field data, which have been sent from a user. The messages are processed, with their details saved to the correct location within the *Principal* database without any specific scenario functionality required.

The analysis components of the TAU for some aspects of the scenario were successful whilst for others they were of limited use. The user is provided with options in a variety of *Incoming* templates, whereby different actions that the user choose would result in different outcomes within the scenario. An example of which, is opting between providing manual weight goals or automatically generated goals. The TAU would respond appropriately based on the method requested by the user, with the template's *Trigger* and *Answer* entities working together to determine the correct responses. Therefore, the TAU is able to adapt and respond appropriately to user requests and the field data that they have provided.

This scenario showed that providing functionality for mathematical calculations within the TAU's generic structure would be beneficial. The scenario's application module is relied upon for most calculations, with the exception of testing one value against another. For example, methods in the application module computed the calorie intake from consumed food items and the calories burned from an activity performed. If the TAU included a calculation stage, this would reduce the reliance on the application module for many of the basic calculations that were required for this scenario and thus reduce the scenario's development time. Furthermore, the calculated data could then be utilised by the TAU's analysis components instead of the application module being required to perform further analysis on the current message.

The TAU interactions with the data layer, both for the *Principal* database and the scenario's database, worked together successfully. Field data is initially extracted and stored in the *Principal* database. The *DatabaseHandler* class utilised each *DataSave* entity to insert and update records in the scenario's database with the user's field data. The TAU could then use the data again for sending back responses to the user and for analysis of future messages. This enabled the TAU to provide accurate and detailed feedback to the user based on the stored data, as shown by the example where a warning message is sent if the user breaches an alert limit. The user's field data is analysed by using previously received data that is stored in the scenario's database, which determines whether a breach has occurred. The *DataOutHandler* class retrieves the stored data in order to generate the warning message.

Overall, the running of the *Diet Diary* scenario has shown that the platform provides an effective infrastructure and the functionality to support a typical scenario that focuses on one user. The builder can easily implement scenarios, which have similar feature requirements, into the platform by utilising the web-based user interface. The three-stage scenario development process considerably reduces the time required to develop the scenario in comparison to creating an application from scratch. Only the calculations within the application module require any programming effort. The first two stages enable the builder to create content declaratively following a simple procedure. The communication structure is tailored to the scenario's requirements to efficiently collect and process received messages together with providing feedback messages and instructions to the user. These operations are achieved in real-time with no assistance required from the operator.

## 7 Case Study 2: *Missing Persons* Scenario

### 7.1 Introduction

The previous chapter discussed a scenario that only required one user for each instance. In this chapter a more advanced scenario, entitled the *Missing Persons* scenario, is discussed. This scenario focuses on utilising multiple users of mobile communication devices in order to locate people that have been reported missing. Each instance of a scenario represents a *Missing Persons* case, whereby the users involved with helping to locate the person are assigned to the respective instance. Subsequently, there can be numerous users assigned to each instance.

The *Missing Persons* scenario utilises the platform's communication component to interact with each user assigned to an instance. In addition to one-to-one communication the scenario looks at the communication with multiple users, whereby each interaction with one user can result in further interactions with other users. This can be via broadcast communication for disseminating updates to multiple users as an instance progresses. Alternatively, the field data received from one user could be used to provide instructions and tasks for other users.

There are some features, previously identified in Chapter 3, that are not required by the *Diet Diary* scenario. Such features include the coordination of multiple users, location-dependent assignments, multimedia messaging and the ability for users to request anonymity. These features are employed by the *Missing Persons* scenario, with the scenario implementation section of the chapter detailing their inclusion. In the scenario's development process, a scenario database and application module are created to hold tables for the scenario's entities and provide algorithms for unique analysis requirements respectively.

A comparison has been undertaken between using the platform to integrate the *Missing Persons* scenario and developing an application from scratch that achieves the same objectives. This comparison highlights the time and effort savings when applying the platform's scenario development process.

The running of an active instance is investigated, whereby the use of the new features are discussed together with the platform's ability to handle multiple users. This is

followed by a discussion on a scalability test that assesses the performance of the platform when operating with large user groups. This test has been carried out using numerous instances of the *Missing Persons* scenario.

## 7.2 Details of the scenario

The aim of the *Missing Persons* scenario is to track and locate people, who have been reported missing, by managing and coordinating multiple users. In the UK there are numerous reports of missing persons every day [195]. Many cases have a short duration as the missing person reappears soon afterwards unharmed [196]. However, there are some cases where the missing person is the victim of a crime or in possible danger. The longer a person is missing the higher the probability that it is one of these cases. For this reason it is important to determine the risk at an early stage of a case, to make appeals to the public and to carry out searches swiftly in an attempt to find the person before any harmful encounters [197].

Similar to the *Diet Diary* scenario, the *Missing Persons* scenario relies on the platform's communication components and template mechanism to handle its communication structure, together with the procedures of the *Template Analysis Unit* (TAU) to analyse each input message. However, this scenario has been chosen due to its particular requirements and objectives. One of the platform's key goals is the generic framework and functionality for supporting a range of diverse issues. Therefore, it is important to observe how the platform handles different types of scenarios. The *Missing Persons* scenario focuses on applying the methods which are currently in place by the police [197], for reacting to a report of a missing person, into an environment where the general public can actively interact and aid the search process. Each time a missing person has been reported, the current procedure undertaken by the police is to provide a unique case identification number [198]. In the *Missing Persons* scenario each case is represented by an instance and uniquely identified by the *Instance* identifier.

An instance of the scenario consists of three phases that are necessary in order to follow the procedure instructions from police practices. The first phase starts when a user reports a new missing person. The initial response involves gathering details concerning a missing person from the reporter, which includes requesting a physical description and recent photograph to assist in finding the person [197]. Sharing a

photograph of the missing person has been made easier by the number of mobile phones that now have an integrated camera, as described in Chapter 2. The photograph can then be utilised by other users at a later date in the search as it provides an accurate representation of the missing person. This takes advantage of the platform's ability to send, receive and store multimedia files, which are important features identified during the research and development of the platform.

It is important to identify the risk status of the missing person in order to determine the response level required [199]. This is based on whether there is a high possibility of danger from factors such as physical assault and the natural environment [200]. Risk assessment details are requested from the initial reporter to assess the missing person's vulnerability and possible circumstances behind the disappearance.

The second phase involves actively searching for the missing person by involving and coordinating a group of people who are willing to be involved. The platform can appeal for help from people who are associated with the missing person. This is achieved by the initial reporter supplying contact details of these people. People willing to assist are added as new users to an instance, making it possible for multiple users to join any given instance. These users can then supply additional information on the case and help recruit more users. New users are provided with the option of remaining anonymous, whereby a user can choose for only their contact details to be stored by the server or for no details to be stored at all. In this way a user is able to provide information to assist the case in the knowledge that their identities will remain hidden, if they so desire.

A user is able to report details of locations important to the case, for example the missing person's home and workplace. Suspicious sightings and vehicles can also be reported. Each location or sighting reported is to be searched by one of the users participating in the instance, who then updates the server with their findings to further the investigation. Each new search is classified as an assignment. The scenario coordinates the users of an instance by ensuring that only users currently not on assignments are requested to start a new assignment and that the users are allocated to assignments in an efficient manner. The location-dependent assignment functionality of the platform is employed to allocate these searches, with the aim being to designate each assignment to the user who is geographically closest to the location. The server

is able to start a new dialogue with one user based on information received from communication with another user. For example, field data received from the first user could be used to provide assignment instructions to the second user. A user who is placed on an assignment is sent the instructions to be performed and the *Incoming* template for returning field data from their findings.

The third and final phase of an instance occurs once the missing person has been found. In this phase the server uses broadcast communication to relay the update to all the users of the instance. Therefore, in this scenario feedback from the server is not only for the purpose of the initial user but also to inform other users of new information.

## 7.3 Implementation of the scenario

### 7.3.1 Database entities

The scenario builder would work closely with the police to set up each component, template and the analysis criteria of the *Missing Persons* scenario. The components of the *Missing Persons* scenario are represented by the tables of the scenario's database, as illustrated in Figure 7.1. The scenario's database has been created to support each area of the scenario, which ranges from reporting details of a missing person to identifying people and locations for an assignment.



**Figure 7.1:** Entities of the scenario's database



Central to the scenario is the *MissingPerson* table with each record representing the missing person of an instance. A *MissingPerson* record stores details of the case, including the date reported, the risk status, instance phase and person's name. Similar to a *Diet Diary* instance, this central record contains the *Instance* identifier to associate the records within the scenario's database to the appropriate instance in the *Principal* database.

Data relating to the physical appearance of the missing person is stored in the *PersonDescription* table, including the gender, age, physical build and complexion. There is also a *PhotoLocation* field which points to the file location on the server where a photograph of the missing person has been saved. This field is used to locate the file when it needs to be sent to users joining the instance. Logging a photograph and description of the missing person is important to aid the instance's users in searching for the missing person.

The risk assessment data sent back from the user in the initial phase of the instance are recorded into the *RiskAssessment* table, where there is a field for each criteria category. Such criteria includes the missing person's mental health, physical disabilities, drug addictions, possibility of self-harm and history of abuse. In each field the user provides a *yes* or *no* answer to indicate whether there is an issue. Furthermore, the user can provide a more detailed description of the issue. These categories identify possible vulnerabilities, which are used to determine the risk status of the missing person.

The *PeopleInvolved* table stores records on people who are associated with the missing person, along with their reason for association. Where a person may be suspiciously involved a report would be flagged for further investigation by the operator. Alternatively, a person may be recommended by one of the users to assist with the case. Their connection to the missing person would be logged in the record.

Each report of a frequented location or sighting of the missing person is stored in a record in the *Locations* table along with the user who reported the findings. In the case of frequented locations the user can supply information on the location-type, for example whether it's the missing person's home or work location. If an assignment

has been created to investigate the location or sighting then the *Assignment* identifier is logged together with the results reported back from the user.

Suspicious vehicles are logged into the *Vehicles* table which has fields for the owner, make and model, as well as the last date sighted and the location. Users are able to supply a photograph of the car with the file location on the server stored in this record.

G. Newiss [197] describes how a key element of police policies in handling a missing person's case is to ensure that there is comprehensive documentation of all reported information and actions undertaken, with a timestamp and details of the occurrence. The scenario's database enables effective documentation with each discovery by a user recorded in the appropriate table. In addition to supporting a current missing person's case, this data could then be utilised for future cases and repeat disappearances.

### **7.3.2 Template and analysis criteria**

In the *Missing Persons* scenario the template and criteria creation follows the same process as the *Diet Diary* scenario, whereby the builder interacts with the web-based user interface to define each step of the scenario. However, this stage also provides a framework for the use of the new features required by the *Missing Persons* scenario.

The builder is able to specify the messages that a user can attach a multimedia file, by defining the *AttachAllow* attribute of the associated *Incoming* template. The *Template* web-form, illustrated in Figure 7.2, provides the option of choosing this feature and selecting a field in the scenario's database to log the location of received files. Each image file associated with the *Missing Persons* scenario is saved to a location on the server. The file is categorised based on the scenario and contains the *Instance* and *Message* identifiers in the file name to simplify locating the file at a later stage. For each new message, the filename and path is logged to the specified database location.

ID	89
Name	Person Description
Template Structure	<pre>Please enter the details below on the description of the missing person.  If you have a photograph of them please supply as an attachment in your response. Age: *years* Gender: *male/female* Complexion: *complexion*</pre>
Parameter Count	7
Response Template	None
Instance Creator	<input type="radio"/> Yes <input checked="" type="radio"/> No
Category	Setup
Allow Attachment	<input checked="" type="radio"/> Yes <input type="radio"/> No
Attachment Table	PersonDescription
Attachment Column	PhotoLocation

**Figure 7.2:** Incoming template with settings for a message attachment in the *Template web-form*

The *Template* web-form also enables the builder to select the *Outgoing* templates that can include attachments. Each occasion where a new message is to be generated the *AttachAllow* attribute of the *Outgoing* template entity is checked. If this attribute is set to *True* then the specified location in the scenario's database is checked to detect if a multimedia file exists. This multimedia file would then be sent as an attachment. This process enables the builder to define which messages can be sent with an attachment both on the user and server side, as well as a universal method of storage and retrieval for all scenarios.

The *UserAdd* action provides the functionality for new users to be added to the instance, based on field data from an input message that has been sent from a current user. The "People Involved" *Incoming* template allows a user to supply details of a person who is associated or who could be of help with searching for the missing person. Figure 7.3 illustrates the *Action* web-form, where the *UserAdd* action is created. The builder provides pointers to the parameter positions in the template where the user can insert the person's name and contact details. These details are used to create a *Person* record in the *Principal* database in order for the TAU to be able to contact the new user.

### Action Form

Trigger ID 41

[Back to Trigger](#)

☐ SelectAll

[Delete](#)

		ActionID	ActionType
<input type="checkbox"/>	Select	66	MessageSend
<input type="checkbox"/>	Select	67	Reply
<input type="checkbox"/>	Select	89	UserAdd

[Data Out Parameters](#)

[New](#)
[Save](#)
[Reset](#)
[Delete](#)

ID

89

Action Type

UserAdd

Name Parameter

1 (name)

Mobile Number Parameter

5 (number)

Email Address Parameter

6 (email)

**Figure 7.3:** UserAdd action created in the Action web-form

The builder defines an *AssignAdd* action in the *Action* web-form for responding to the “Important Locations” and “Sightings” *Incoming* templates. These two assignment types are defined in the *Assignment* web-form (Figure 7.4) and saved as *Assignment* entities. The builder defines the user selection criteria of an assignment during the creation of an *AssignAdd* action. The “User Selection” dropdown-list in Figure 7.5 illustrates that the user to be selected could either be the message sender, a user near the assignment location or randomly selected from the available users. Selecting the *Location* option as shown ensures the assignment is a location-dependent assignment type.

### Assignment Form

**Scenario ID 3**

Scenario 3: Missing Person

☐ SelectAll

[Delete](#)

		AssignmentID	Name
<input type="checkbox"/>	<input type="button" value="Select"/>	22	Search Sighting
<input type="checkbox"/>	<input type="button" value="Select"/>	23	Search Location

[New](#) [Save](#) [Reset](#) [Delete](#)

ID

Name

Description

*Figure 7.4: Assignment web-form*

### Action Form

**Trigger ID 46**

[Back to Trigger](#)

☐ SelectAll

[Delete](#)

		ActionID	ActionType
<input type="checkbox"/>	<input type="button" value="Select"/>	75	Reply
<input type="checkbox"/>	<input type="button" value="Select"/>	76	MessageSend
<input type="checkbox"/>	<input type="button" value="Select"/>	90	AssignAdd

[Data Out Parameters](#)

[New](#) [Save](#) [Reset](#) [Delete](#)

ID

Action Type AssignAdd

New Assignment 22: Search Sighting

User Selection Location

*Figure 7.5: AssignAdd action created in the Action web-form*

There are specific situations during an instance where the TAU is required to update the operator on a situation in order for there to be further investigation into the matter.

An example of this is where a suspicious person is reported by a user. As discussed in Chapter 5, the implementation of *News* records provide these informative updates. *News* records are generated based on a *News* template structure within the *Template* web-form, as illustrated in Figure 7.6. A *NewsInsert* action would create a *News* record based on the template. Figure 7.7 illustrates the properties provided by the *Action* web-form for defining the *NewsInsert* action. In this example, a *News* record would be created to inform the operator of a suspicious person reported by one of the instance's users.

### Template Form

Scenario ID 3

Show Templates for: News

☐ Select All

[Delete](#) [Refresh](#)

	TemplateID	Name	Type
<input type="checkbox"/> <a href="#">Select</a>	108	Location Person News Update	2
<input type="checkbox"/> <a href="#">Select</a>	110	Sight Person News Update	2
<input type="checkbox"/> <a href="#">Select</a>	115	Suspicious Person	2

[Save](#)
[New](#)
[Delete](#)
[Create/Edit Triggers](#)
[Parameters](#)

ID

115

Name

Suspicious Person

Template Structure

User \*name\* (ID: \*ID\*) has reported a person who may be involved in the case. The name of the suspicious person is \*name\*. Their connection to the missing person is \*connection\*. Details of criminal record \*criminal record\*.

Additional comments left by the user: \*comments\*

Parameter Count

6

**Figure 7.6:** *News* template created in the *Template* web-form

## Action Form

Trigger ID 55

[Back to Trigger](#)

☐ SelectAll

[Delete](#)

	ActionID	ActionType
<input type="checkbox"/> <a href="#">Select</a>	91	NewsInsert

[Data Out Parameters](#)

[New](#) [Save](#) [Reset](#) [Delete](#)

ID

Action Type

Response Group

Template to insert News

**Figure 7.7:** *NewsInsert action created in the Action web-form*

The platform supports broadcast communication for scenarios that incorporate multiple users via the *MessageSend* action. Figure 7.8 illustrates a *MessageSend* action being created in the *Action* web-form. The “Response Group” dropdown-list is where the builder chooses which user(s) a response is sent to. In the *Diet Diary* scenario there was only one user assigned to each instance, therefore this value defaulted to the sender of the originating message. The options available in the *Missing Persons* scenario also include users on and off assignments, as well as all users participating in the instance. The example in Figure 7.8 illustrates that the builder has set the appropriate *Outgoing* template to generate the message and declared that the response group includes all the users of the instance. This action would result in a broadcast response.

## Action Form

Trigger ID 45

[Back to Trigger](#)

☐ SelectAll

[Delete](#)

		ActionID	ActionType
<input type="checkbox"/>	Select	73	Reply
<input type="checkbox"/>	Select	74	MessageSend

[Data Out Parameters](#)

[New](#) [Save](#) [Reset](#) [Delete](#)

ID

Action Type

Response Group

Outgoing Template

Proc String

Check Pending ☐ Yes ☒ No

**Figure 7.8:** *MessageSend* action created, where the response group has been set to all users of the instance

The TAU's *Template* and *Action* entities provide the functionality for the new features detailed in this section. Each *Action* entity is declaratively created by the builder with no additional programming required, highlighting the versatility of the response stage of the platform's framework. The correct actions can be selected in a straightforward process by the builder in order for the TAU to react accordingly to user messages.

### 7.3.3 Development of new modules

Analysis of the risk assessment data, received from a user, is performed by the scenario's application module. The user's response to each attribute within the risk assessment form is analysed to determine the risk status of the missing person. This analysis is based on a police report [199] which states that persons containing certain characteristics, disabilities or historical issues may be more exposed to dangerous elements than others. For example, someone with a physical or mental illness is considered to be in greater danger from the natural environment [200,197]. A person



that has a drug dependency and has intentionally run away could become potentially involved with dangerous strangers, in order to obtain assistance with their addictions.

The *RiskStatus* algorithm uses the *yes* or *no* response to each risk attribute sent back from the user to calculate the risk status. There are three levels to the risk status; low, medium and high [200]. The default risk status is set to the low level due to the research indicating that the majority of cases result in the missing person returning safely [196]. If the missing person is affected by certain attributes, including mental or physical illness and drug dependency then the status is set to the high level. Other issues such as the incident being out of character would set the status to the medium level. The risk status is used by the operator to decide whether it is necessary to request other users of the platform to participate in the instance.

A new module, the *LocationHandler*, has been developed within the TAU to support the efficient distribution of assignments. This module is used if the response action from a message is to create a new assignment. Users are instructed to supply the street name and the name of the town/city of reported locations and sightings. This location data are established as the location of the new assignment and passed to the *LocationHandler* module. The list of available users is scanned by the module to find a user who is close to the location. This is achieved by comparing the most up-to-date location of each user to the assignment's location. If a user is found matching both the street and town/city then their details are passed back to the *ActionHandler* class, which in turn sends a request to the user to perform the assignment. However, in situations where no user matches both parameters, then the method would search for users that only match the town/city of the assignment. The details of the first user found is passed back to the *ActionHandler* class. If no user is found in both searches then the method selects an available user randomly for the assignment. Unlike the scenario's application module, the *LocationHandler* module has not been specifically created for this scenario. The *LocationHandler* module has been designed to handle location-based assignments for any scenario.

#### **7.4 Development time comparison for the *Missing Persons* scenario**

The *Missing Persons* scenario required a timeframe of 2 days to design and implement into the platform. This scenario followed the three-stage scenario development process. Designing the new database, in the first stage, with tables

relevant to the *Missing Persons* scenario required half a day. A further half day consisted of constructing each template and defining the scenario's analysis criteria for the second stage. The *Missing Persons* scenario had a reduced emphasis on mathematical calculations in comparison to the *Diet Diary* scenario, with the majority of analysis depending on a decision-tree structure for determining appropriate responses. Therefore, the generic analysis stages of the TAU could handle most of the analysis for this scenario. The new application module, a C# class developed in the *Microsoft Visual Studio* IDE, required just 140 lines of code for the *RiskStatus* algorithm. This module, developed in the third stage, required 1 day to code and integrate into the platform.

The *LocationHandler* module was developed alongside the *Missing Persons* scenario, taking an additional day to develop. However, this module is not included in the total development time due to being integrated as a generic component of the platform, which is reusable for future multi-user scenarios. The *Missing Persons* scenario was ready to run on the platform once the scenario development process and the new *LocationHandler* module were completed.

In order to determine the amount of time and effort saved by using the platform to implement a scenario, a mock-up application has also been designed to achieve the same objectives as the *Missing Persons* scenario. This application would be developed from scratch, without relying on an integrated communications platform. A design specification of the mock-up application has been constructed, specifying the layout of the C# classes and their methods in order to develop each component. The timeframes have been calculated from estimating the programming time of the classes within each component, which are outlined in the design specification. These timeframes are derived based on the researcher's own experience in software development and in consultation with another professional in computing.

This application would be developed using the same programming environment as the platform, with the code written in C# classes within the *Visual Studio* IDE and using a *SQL Server* database. Therefore, the application is able to utilise the same set of class libraries within the .NET framework in order to assist in the development of each component.

The application would need a user interface for the operator to maintain each missing person's case that is running. Following the same route as the *Connected-Mobile Platform*, developing a website for the user interface would also enable users to both retrieve messages stored by the application and send new messages. The website is similarly developed within the *Visual Studio* IDE, using the graphically tools available from the ASP.NET framework to design the layout of each web-form. This website is not as extensive in functionality as the web-based user interface developed for the platform as it would only be focused on one type of application. The application would not need to include functionality to support new application problems. Therefore, there would be no need for the web-forms and entities that relate to the scenario builder. However, the total time to implement the website is 7 days, where 2 days are required for designing the website's elements and 5 days for development.

Users would need to communicate with the application in order to send information on a missing person's case or to receive updates and instructions. This requires the development of a communication component for the application to be able to exchange messages with the users. This component requires a total of 4 days to implement, consisting of 1 day for the design and 3 days for the coding of the functionality within logic classes. This includes the functionality to send messages via SMS and email over the internet and receive new messages from *Microsoft Outlook*. User messages sent via the website would also trigger an event in this component to handle the processing and storage of the message.

Defining the content of each template message sent to and received from users requires a detailed design and layout to ensure the communication process flows smoothly for each missing person's case run by the application. It takes 2 days to design the template entities and construct each type of *Incoming/Outgoing* template that is to be used in the communication process. The website does not include web-forms for adding new templates, since this is a one-off event for the application. Therefore, it would save time to not include this functionality. The templates are added to the database using *SQL Server Management Studio*. This is a complicated process for template creation as each record is manually inserted into a *Templates* table with the template parameters inserted into a *Parameters* table. In comparison,

the platform already offers a straightforward process for defining template content via template mechanism and web-based user interface.

The message analysis component requires 13 days to implement into the application. Designing each stage of this component takes 3 days, based on following the same analysis stages as the platform. Developing the C# logic classes for data extraction, analysis, calculations and storing the received/calculated data requires 5 days of programming. A further 5 days are required for developing the extensive action modules that this type of application problem demands. This includes providing functionality for the different response types, as well as generating assignments and location-based technology.

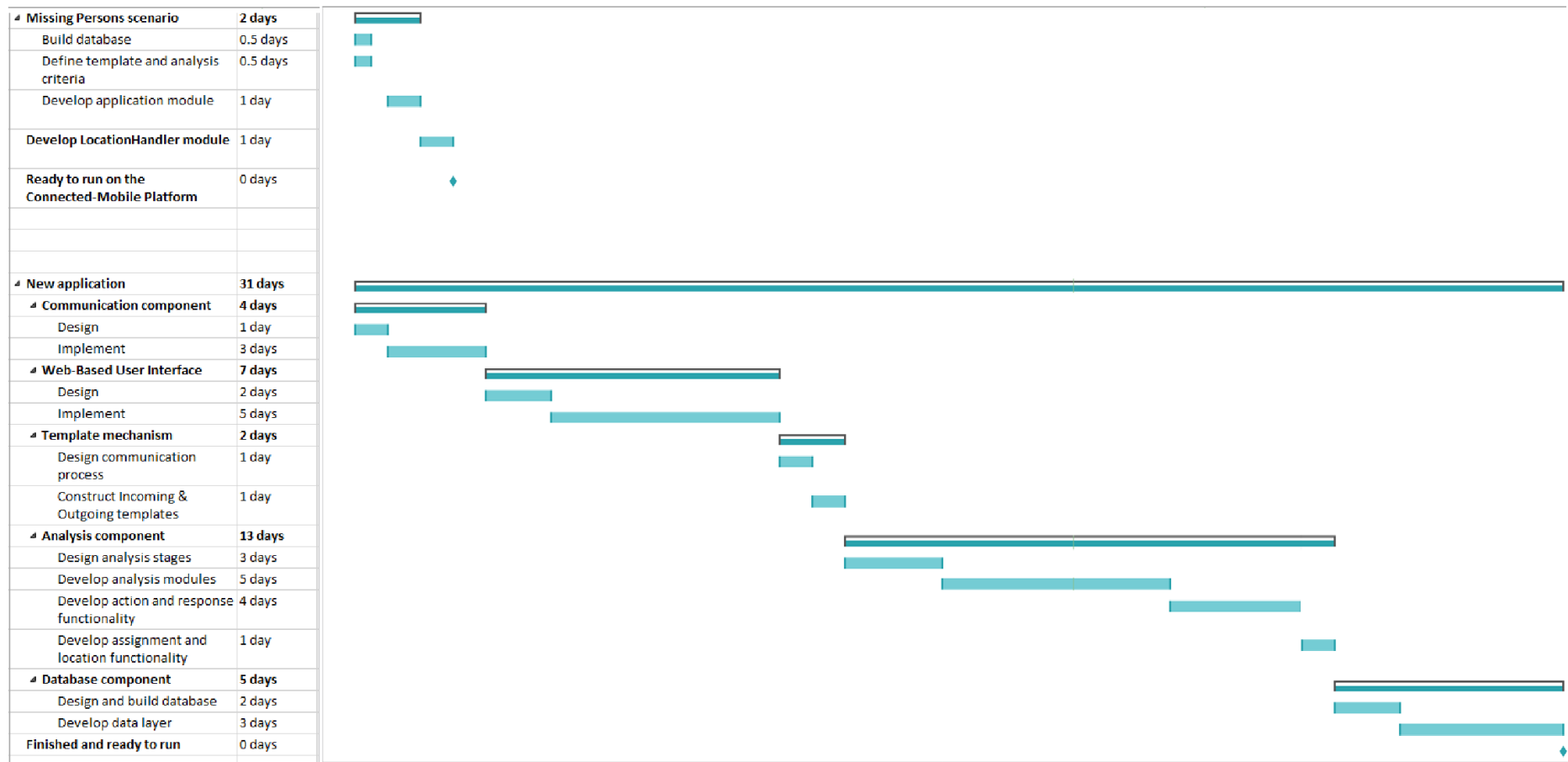
Assuming all the requirements for the application are known at design time, there is no need for defining analysis criteria in the future. Therefore, the analysis criteria is all hard-coded into C# classes as opposed to the platform's approach of using database entities and the web-based user interface to define new analysis criteria. This approach allows for a more efficient means to generate analysis functionality in comparison to adding new scenarios to a generic platform. The methods of each class can be designed specifically for the application's objectives, unlike a generic platform where the methods need to focus on handling a wide range of application types. Therefore, the code can be fine-tuned to ensure the application can perform its tasks effectively. However, the platform enables the inclusion of a plug-in application module which helps to overcome this issue by allowing the scenario builder to code new and precise functionality exclusively for the *Missing Persons* scenario.

A single database is created to store all the data of the mock-up application. This database needs to store data on entities such as the templates, messages and users from the platform's *Principal* database. Additionally, the database stores data on the entities that directly relate to the *Missing Persons* scenario in this case study. The design and construction of the database is achieved in 2 days. A further 3 days are required to develop the data layer of the application to access the database. The data layer includes entity classes in order to handle individual records in each table. Other classes are required to populate the HTML tables on the website and manipulate data across multiple tables. Each of these classes within the data layer utilise the set of ADO.NET classes within the .NET framework in order to access the database.

The total time to develop and implement an application that achieves the same objectives as the *Missing Persons* scenario is 31 days. This is a substantially longer timeframe than the 2 day period for integrating the scenario into the platform. The reason for the longer timeframe in creating the new application is that every aspect of functionality has to be individually designed and developed, as discussed above. In comparison, the scenario builder is only required to design a new scenario's entities, the structure of the communication process and the new functionality for the application module in order for the scenario to be implemented into the platform. The functionality for interacting with the clients, defining message structures, analysing messages and handling application data are already provided by the platform's framework of generic components. Therefore, only a limited amount of development is carried out to implement a new scenario. Figure 7.9 illustrates the large difference in timeframes between implementing the *Missing Persons* scenario using the platform's scenario development process and creating the mock-up application from scratch.

During the development of the separate application it would also have to be rigorously tested to remove software bugs and ensure that each component operates and interacts with other components as expected. This testing process would add additional time to the implementation of the application.

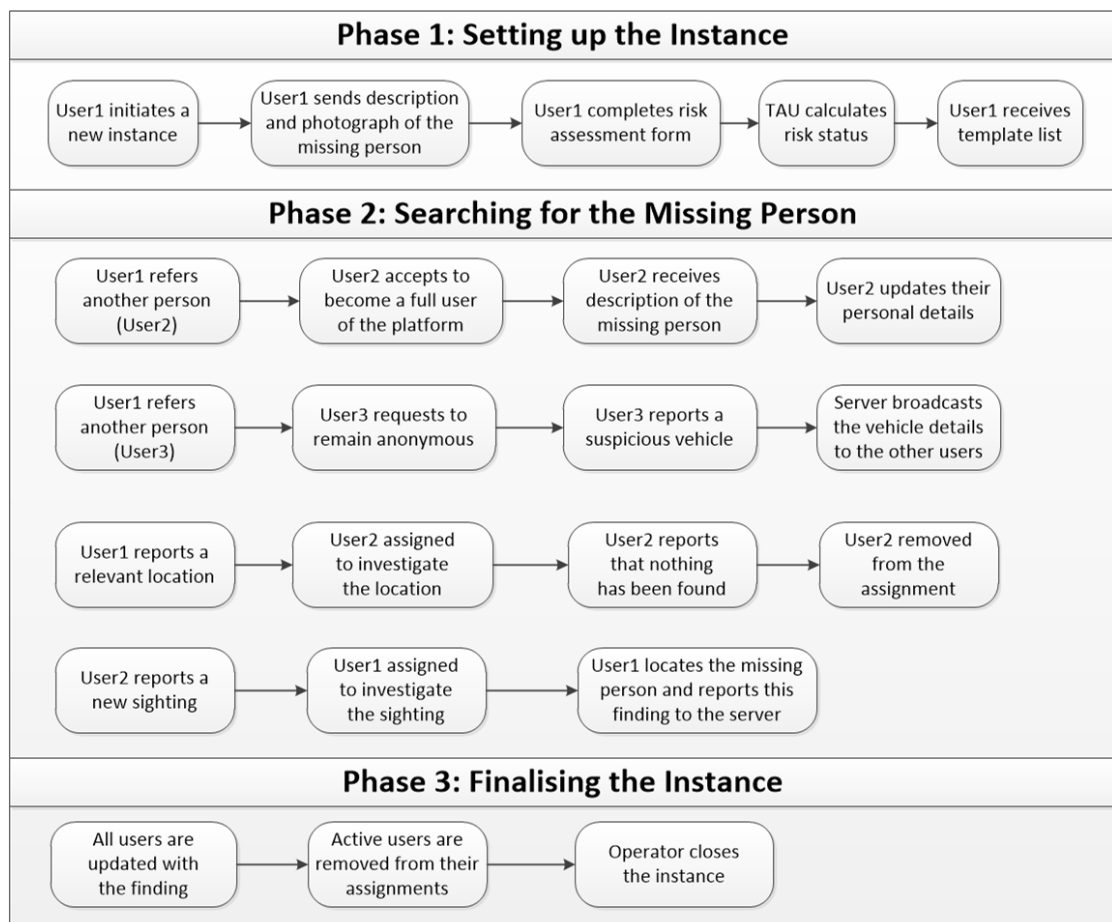
Furthermore, some elements including the creation of template content and analysis criteria require a more complicated process, either through developing new hard-coded modules or manually inputting content into the database. The platform's web-based user interface provides a straightforward means to achieve this by automatically linking associated entities in the database and preventing the risk of input errors with error checking routines at each stage of data input. Therefore, content can be quickly tailored to the requirements of a new scenario. The drawback of not being able to develop code exclusively for one application type is overcome by the platform's inclusion of plug-in modules for new scenarios. The platform's dual approach of offering the dynamic creation of analysis criteria via the web-based user interface and permitting the integration of an application module that has specific hard-coded functionality for the *Missing Persons* scenario enable the objectives of this scenario problem to be covered efficiently.



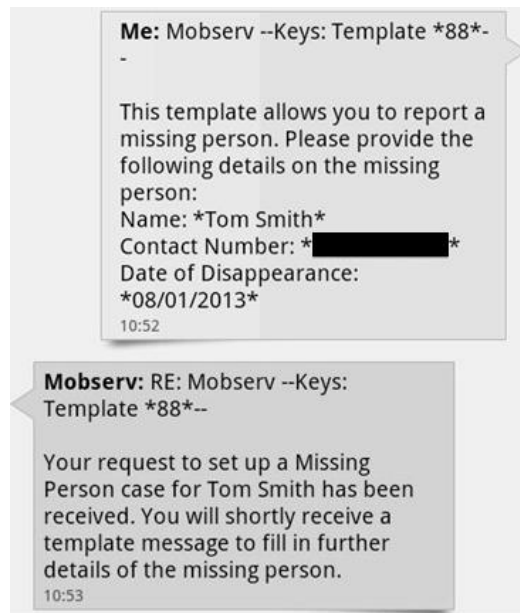
**Figure 7.9:** Comparison of timeframes between developing the Missing Persons scenario using the platform (top) and developing a new application from scratch (bottom)

## 7.5 Running a *Missing Person* instance

The flow diagram in Figure 7.10 demonstrates an example of the sequence of events performed throughout an active *Missing Person* instance, beginning from the initialisation and ending when the missing person is found. A *Missing Person* instance can be initiated either by the operator or a user. A user would request the scenario's *InstanceCreator* template, illustrated in Figure 7.11, to supply the relevant information to start a new instance. The user starting this instance is referred to as *User1*, who could be a family member or close friend of the reported missing person.



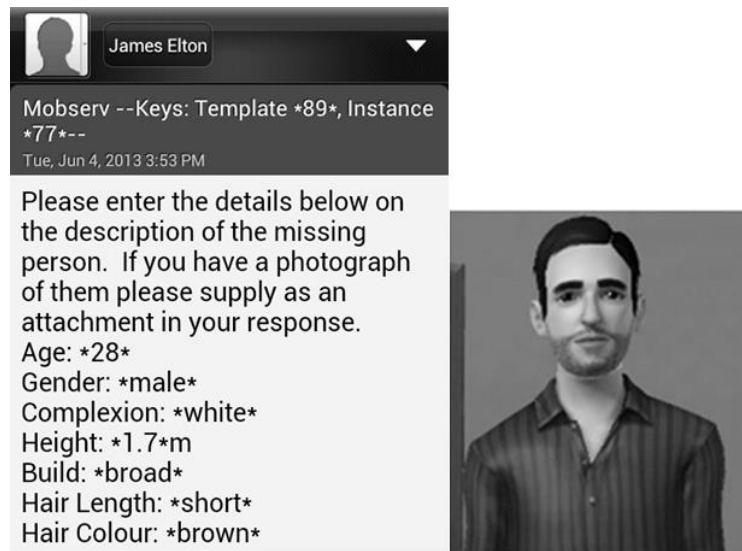
**Figure 7.10:** Sequence of events performed throughout an active *Missing Person* instance



**Figure 7.11:** SMS message exchange to commence a new Missing Person instance  
(For SMS: Server messages begin with “Mobserv”, User messages begin with “Me”)

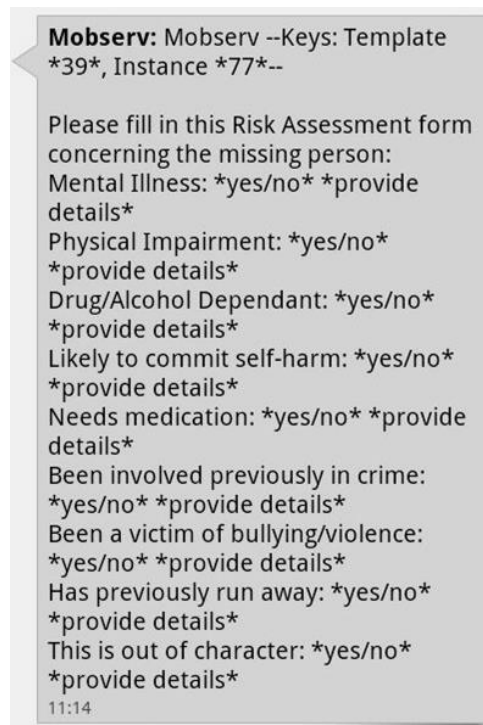
Once the instance has been created *User1* is requested to send a physical description of the missing person within a supplied *Incoming* template. This template also informs *User1* that they should attach an image file containing a photograph, if available, of the missing person. If the user’s preferred communication method is not set to email, then any *Incoming* templates that facilitate the inclusion of an attachment are sent via both the users preferred method and email. The user is then able to respond via either their preferred method without including an attachment or email with the requested image file attached. Figure 7.12 illustrates the message sent by *User1* to provide the physical description with an attached photograph of the missing person. In this example, the user has sent the message via email to include the attached image file. When processed by the TAU, the image file is saved to the server. The field data and the location of the saved image file are stored in a new record in the *PersonDescription* table.



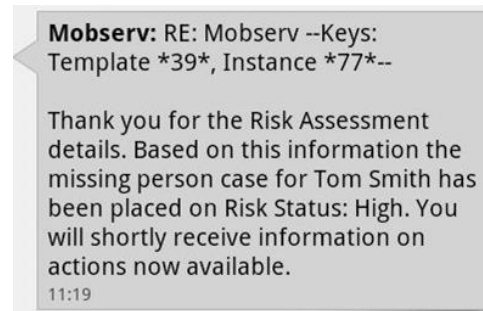


**Figure 7.12:** *User1 sends email message based on an Incoming template to provide a physical description (left) and attached photograph (right) of the missing person*

The final step of setting up the instance involves the risk assessment form. Figure 7.13 illustrates the *Incoming* template sent to *User1* to provide the risk assessment data. The user can give a *yes* or *no* answer for each risk attribute and then provide further details on an attribute if they deem it necessary. The *yes/no* answers are used by the application module to calculate the risk status of the missing person. The extra details are for the attention of the operator. A record is created in the *RiskAssessment* table to store the risk assessment data and the calculated risk status. *User1* is also notified of the risk status, as illustrated by the server's response in Figure 7.14.

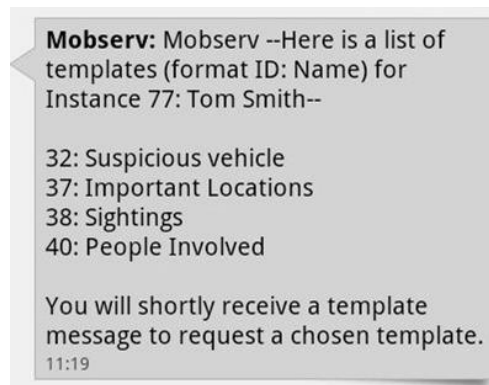


**Figure 7.13:** “Risk Assessment” Incoming template for User1 to complete



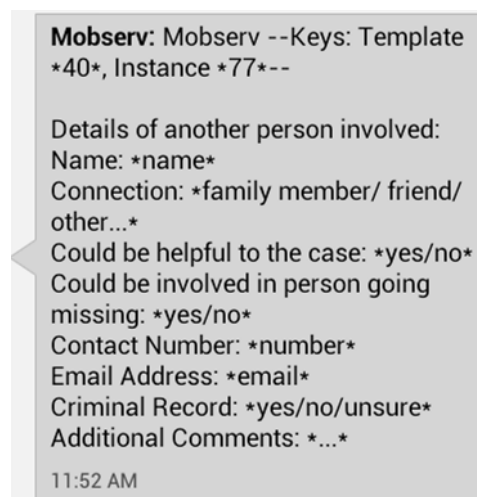
**Figure 7.14:** Feedback message sent to User1 containing the risk status

Once the setup phase of the instance has been completed the user is able to report on the four elements that are involved in the search phase of the instance. These elements include important locations relevant to the missing person, sightings of the missing person, other people involved and any suspicious vehicles that could be associated with the case. The template list sent by the server (Figure 7.15) provides the user with the choice of requesting the *Incoming* template for each of these elements.

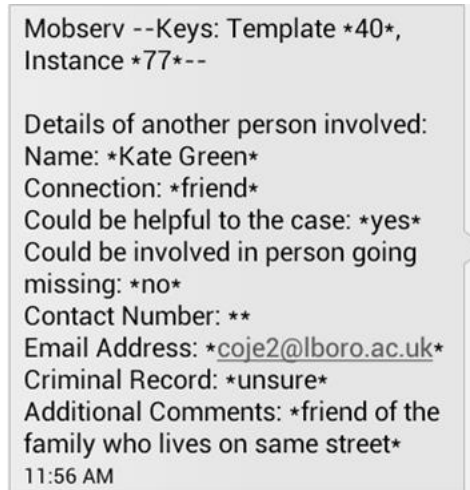


**Figure 7.15:** List of Incoming templates available for a user to request

Users may report on another person associated with the missing person using the “People Involved” *Incoming* template, as illustrated in Figure 7.16. The TAU’s *AnswerHandler* class analyses the field data to determine the correct actions to perform. Initially, a record of the reported person is created in the *PersonInvolved* table of the scenario’s database. If *User1* reports that the person could be helpful then the server would attempt to recruit the reported person to assist with the instance. The *UserAdd* action creates a new *Person* record in the *Principal* database. This record allows the reported person to be a temporary user of the platform, storing the name and contact details of the reported person that were extracted from *User1*’s message (Figure 7.17).



**Figure 7.16:** “People Involved” Incoming template sent to *User1*

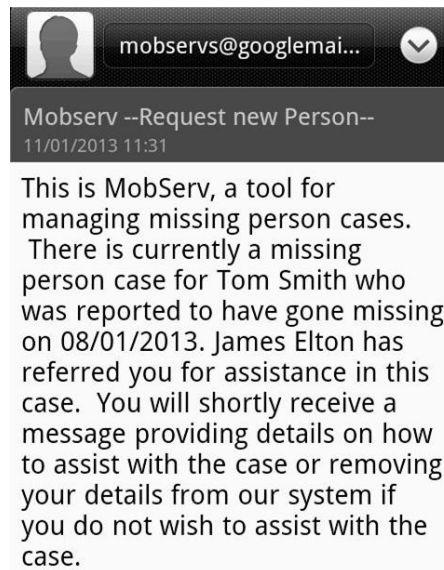


Mobserv --Keys: Template \*40\*,  
Instance \*77\*--

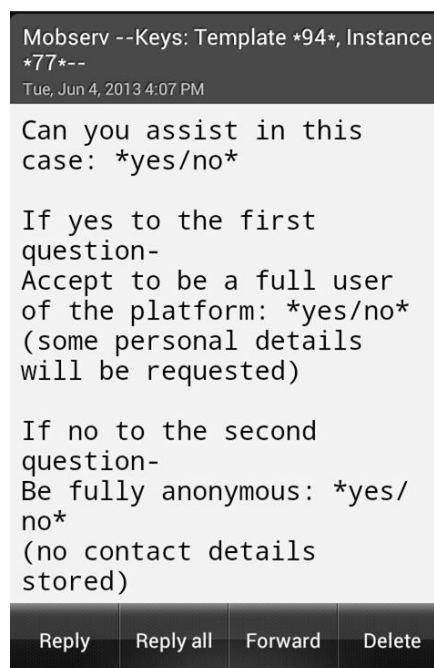
Details of another person involved:  
Name: \*Kate Green\*  
Connection: \*friend\*  
Could be helpful to the case: \*yes\*  
Could be involved in person going  
missing: \*no\*  
Contact Number: \*\*  
Email Address: \*coje2@lboro.ac.uk\*  
Criminal Record: \*unsure\*  
Additional Comments: \*friend of the  
family who lives on same street\*  
11:56 AM

**Figure 7.17:** *Input message sent from User1 reporting a person involved together with their contact details*

The server then attempts to contact the new user, referred to as *User2*, by sending a message describing the instance (Figure 7.18) together with an *Incoming* template to enable *User2* to accept or decline the request for assistance (Figure 7.19). SMS is the default method for sending a message to a new user. However, in this example the messages have been sent via email since the only contact details provided was *User2*'s email address. If *User2* accepts the request then they would become a full member of the platform. *User2* is also provided with the option of remaining anonymous. They can choose to have either no personal details stored or only their contact details stored. In both these cases the *PeopleInvolved* record in the scenario's database is updated to hide personal information. Declining the request for assistance would result in the deletion of the *Person* record relating to *User2* in the *Principal* database, ensuring there is no further contact with *User2*.



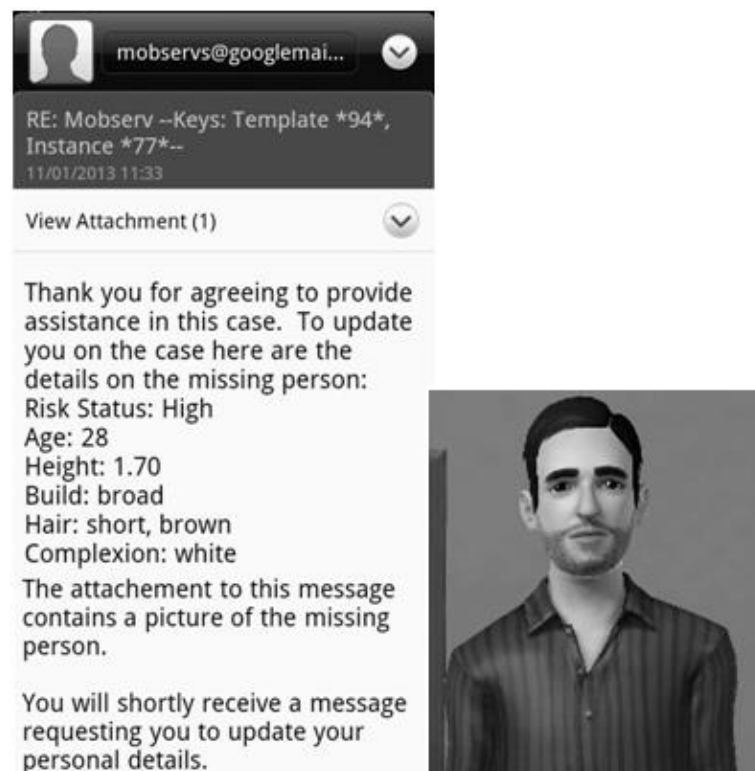
**Figure 7.18:** Email message sent from the server to User2 requesting their assistance



**Figure 7.19:** Incoming template sent to User2 to join the platform

The server sends details of the physical description together with the saved photograph of the missing person to new users joining the instance. This is illustrated in Figure 7.20 where the attachment previously received from *User1* can now be sent to other users, taking advantage of the platform’s support of multimedia files. New users can update their personal details using the “Update Member” *Request* template

in Figure 7.21. This includes the option for the user to provide a mobile number in case they want to switch to SMS messaging at a later date.



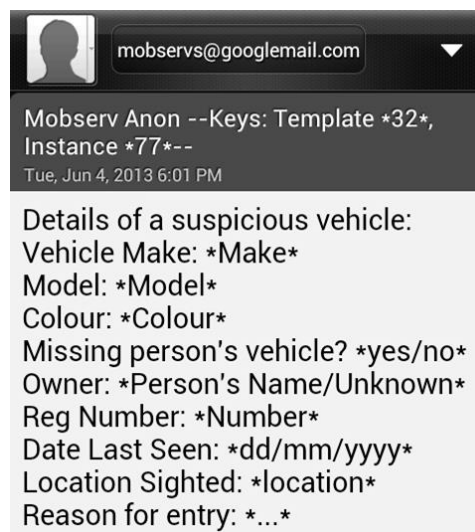
*Figure 7.20: Description of missing person (left) and photograph (right) sent to User2*



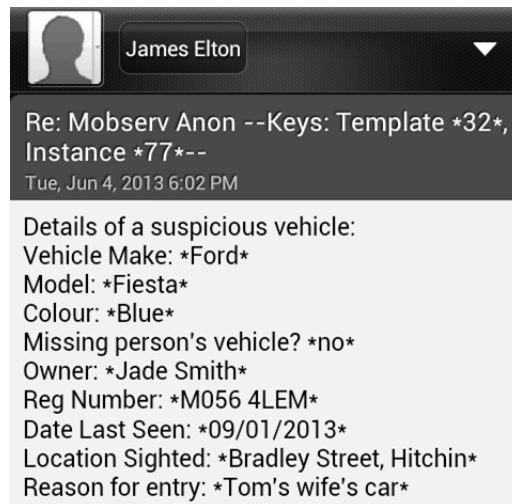
*Figure 7.21: “Member Update” Request template for User2 to provide their personal details*

In this scenario an anonymous user is able to assist an instance by sending reports on the various elements, such as a suspicious vehicle. However, the TAU does not allocate anonymous users to assignments. Fully anonymous users, where the platform does not store any contact details, are required to include the phrase “*Mobserv Anon*” at the start of each input message. This phrase ensures the server is aware that the message sender has requested to remain anonymous, thus enabling the server to maintain a dialogue without recording their contact details.

All new users are provided with the scenario’s template list to send reports on the various elements. Anonymous users are able to request *Incoming* templates from the server, which follows the same procedure as non-anonymous users. Figure 7.22 illustrates the “Suspicious Vehicle” *Incoming* template that would be sent to an anonymous user, referred to as *User3*, for reporting a vehicle involved in the instance. Inserting the appropriate field data and returning the message to the server, as illustrated in Figure 7.23, would result in the analysis of the data and a response by the server.

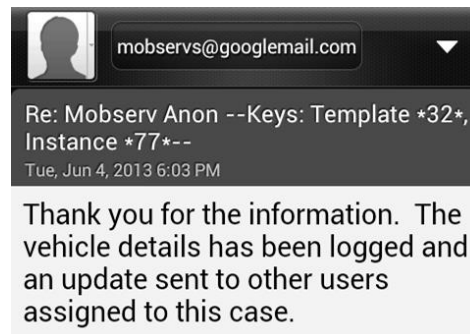


**Figure 7.22:** “Suspicious Vehicle” *Incoming* template sent to *User3* (anonymous user)



**Figure 7.23:** *User3 sends a vehicle report to the server based on the “Suspicious Vehicle” Incoming template*

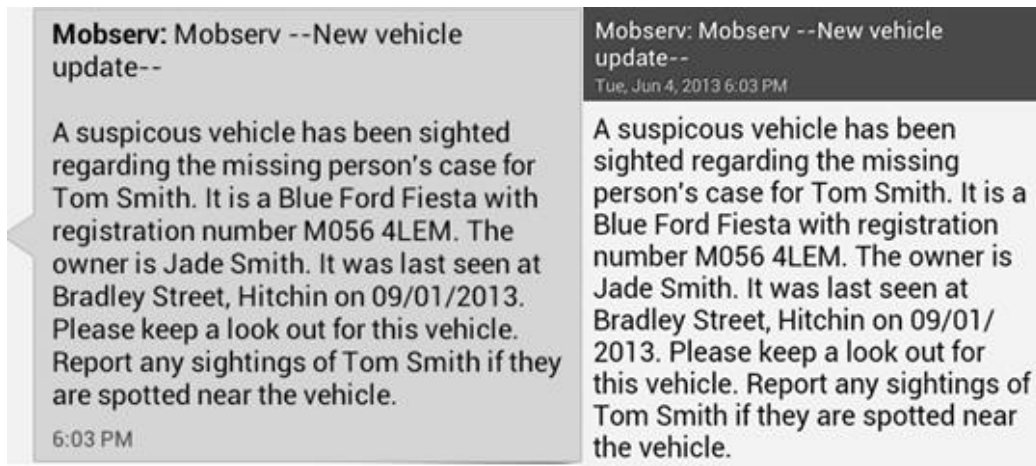
The contact details of *User3* are temporarily stored in the *Full Anonymity* record within the *Person* table of the *Principal* database. Once all the actions have been performed then these contact details are deleted, ensuring *User3* remains anonymous. Figure 7.24 illustrates the response message from the server to *User3*.



**Figure 7.24:** *Feedback message sent to User3 confirming the vehicle has been logged*

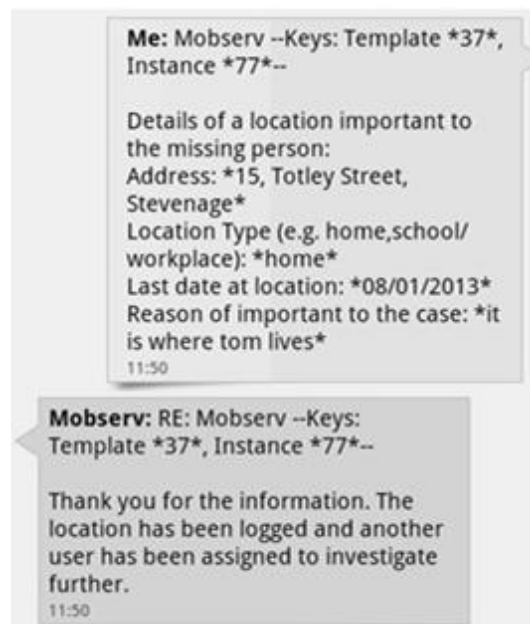
In situations where suspicious vehicles are reported a broadcast message is sent to other users involved in the instance. An example of this message is illustrated in Figure 7.25, where details of the vehicle are described to *User1* and *User2* via their preferred communication method. The details included in this message have been taken from the *User3*’s message reporting the vehicle. This demonstrates how the platform can readily update other users on newly received field data in real-time.





**Figure 7.25:** Broadcast message sent to User1 (left) and User2 (right) regarding the reported vehicle

It is important for the platform to coordinate the provision of assignments for each instance and react appropriately based on the outcome of each assignment. An example of this is where multiple relevant locations or sightings of the missing person have been reported. As previously mentioned, users are allocated to these assignments by seeking the current location of each available user. Figure 7.26 illustrates an example of a relevant location being reported by User1, followed by a notification by the server.

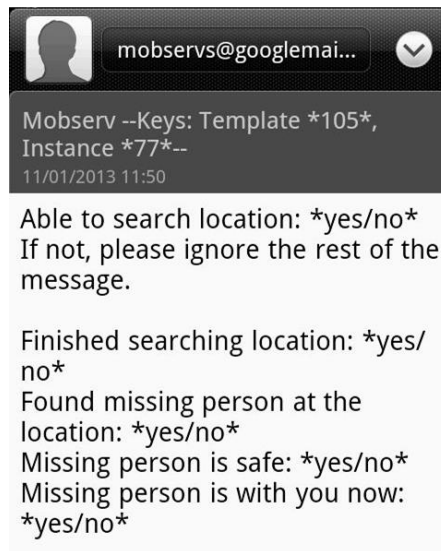


**Figure 7.26:** A location report sent from User1 based on the “Important Locations” Incoming template, followed by a feedback message returned from the server

An assignment is generated for the purposes of searching the location to determine if the missing person is in that location or whether there are any further leads. The *LocationHandler* module processes the address of the location, provided by *User1*, to find the closest user. *User2* is selected by the *LocationHandler* module to perform the assignment since their current location matches the reported location in this example. A message is sent to *User2* via their preferred communication method describing the objectives of the assignment and requesting their cooperation, as illustrated in Figure 7.27. *User2* is also provided with the *Incoming* template (Figure 7.28) to report back and provide data on their observations.

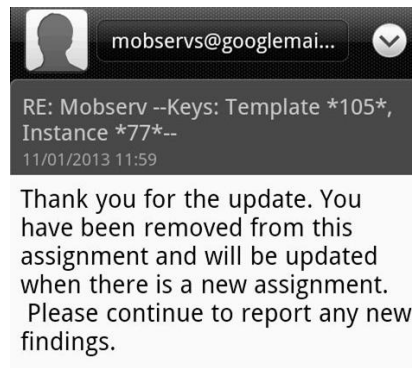


**Figure 7.27:** *User2* receives instructions from the server on the newly generated assignment



**Figure 7.28:** Incoming template sent to User2 regarding the completion of the new assignment

The TAU analyses the returned field data from an assignment to determine whether *User2* has found the missing person. *Trigger* entities are used to determine the correct outcome from the user's message relating to an assignment. The actions associated with a trigger are only carried out if the field data retrieved from the user passes the criteria of the associated *Answer* entities. If the user did not find the missing person at the location then the *NotFound* trigger would be activated. The TAU's responses from this trigger are to remove *User2* from the assignment and to notify them of this action (Figure 7.29). This assignment has demonstrated that a user can be selected based on their current location. The user is taken through the steps of the assignment with the use of *Outgoing* and *Incoming* templates, enabling them to communicate with the server and successfully fulfil the assignment's objectives.

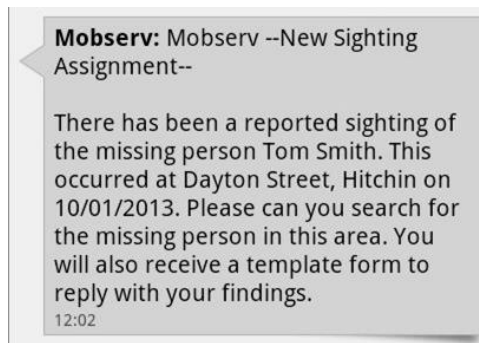


**Figure 7.29:** Feedback message sent to *User2* notifying of their removal from the assignment

Reported sightings of the missing person also generate new assignments. The message in Figure 7.30 illustrates an example whereby *User2* has sent details of a new sighting to the server. As with the previous assignment the *LocationHandler* module selects the user deemed to be closest to the location of the sighting, which in this example is *User1* who is located in the same city. The server sends a message to *User1* describing the sighting and their new assignment, as illustrated in Figure 7.31.

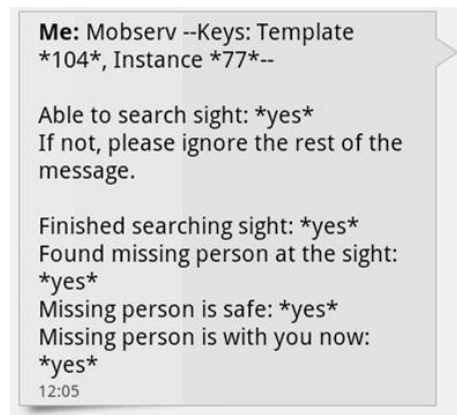


**Figure 7.30:** *User2* sends a sighting report based on the “Sightings” Incoming template

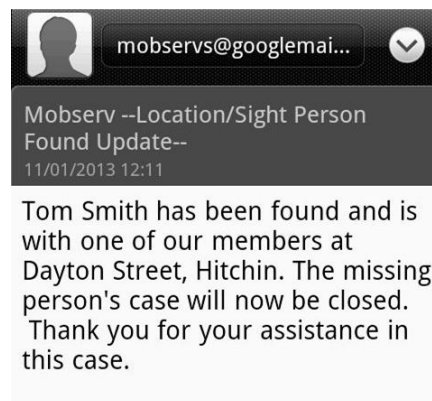


**Figure 7.31:** *User1 receives instructions from the server on the new assignment*

The user also receives an *Incoming* template to respond with their findings. Figure 7.32 illustrates the message that is sent from *User1* after they are successful in finding the missing person in the assignment's location. Similar to the first assignment, the field data is analysed by the *AnswerHandler* class. The *Answer* entities, defined by the builder, enable the TAU to make accurate decisions after a user has reported back from an assignment. The *FoundPerson* trigger is activated with the TAU performing the appropriate responses, based on its associated *Action* entities. These actions include the removal of active users from their assignments and sending a broadcast message to all the users of the instance, notifying them that the missing person has been found. Figure 7.33 illustrates the broadcast message, as received by *User2*. Additionally, the instance's status would be set to its last phase and a *News* record would be generated for the operator to be updated with the progression. The operator can then report the details of the missing person's location to the police and close the instance. These examples illustrate the way users are employed within an instance, with the actions of one user having an effect on others and the ability for the TAU to coordinate multiple users via assignments.



**Figure 7.32:** *Input message sent from User1 informing the server that the missing person has been located*



**Figure 7.33:** *Broadcast message sent from the server updating the other users on the findings of the assignment*

## 7.6 Scalability test to assess the performance of the platform

In the two case studies, thus far, running instances of the scenarios have only been performed with up to three users. The key purposes of these demonstrations were to highlight the feature-set of the platform and how the generic components can be successfully tailored to a specific scenario. However, the platform is designed to support a wide range of scenarios concurrently, where each instance of a scenario may require numerous users to be deployed in a mobile environment. Subsequently, the platform needs to be able to handle a large user group without a sacrifice in performance. In certain scenario types, such as the *Missing Persons* scenario, users would send in time-sensitive data that needs to be processed and acted upon in real-time. Therefore, the platform's central server needs to collect and analyse each input

message in a short timeframe after they have been received, without delaying subsequent messages or affecting the performance in other areas of the platform.

A scalability test has been designed and performed on the platform in order to assess its performance whilst supporting a large user group. The first stage of this test was to observe the platform's ability to handle 20 active users, where the focus was on multiple *Missing Person* instances running concurrently with more than one user in each instance. The users interacted via all three available communication methods, SMS, email and the web-based user interface.

The main occasion when the platform would be put under strain is at the time that the central server receives new messages, due to the extensive procedures carried out by the TAU for analysing and responding to each message. Therefore, to measure how long it takes for the platform to be responsive again for new user interactions, 20 messages were received simultaneously with one from each of the active users. This is a worst-case scenario for the platform to handle, where it is assumed that every user would be sending a message to the central server simultaneously. The platform has only one main thread running, which results in only one message being processed at a time from the batch of retrieved messages. This is based on the order of the date and time that the message is received. Furthermore, all the messages need to be processed before a check can be made for new messages in the *Microsoft Outlook Inbox* folder. A C# *Timer* object would call the *MailHandler* class every 20 seconds to check the *Inbox* folder. If there are messages waiting in the *Inbox* folder then the *Timer* object is paused throughout the duration of the TAU stages. There is an automatic check for new messages once the TAU stages have been completed. The *Timer* object is only reset to check at regular intervals once no messages have been found in the *Inbox* folder.

The complexity of the analysis performed on a message can vary and depends on the volume of *Trigger* and *Answer* entities created for the associated *Incoming* template and whether additional calculations need to be performed by the application module. This complexity would largely determine the length of time between retrieving a message and performing a response. Additionally, the type of actions that would be performed in response to the message is an important factor. The scalability test was performed on a wide range of message types associated with the *Missing Persons*

scenario to ascertain the time taken to process a batch of messages at different levels of analysis complexity.

A message sent by a user requesting for a specific *Incoming* template is a basic type of message for the TAU to process. It would take SMS and email messages on average 12 and 10 seconds respectively to be received by the email server that operates the platform's email account. The time taken to then process 20 of these messages, each sent from a different user and using a mixture of SMS and email, was on average 15.2 seconds. Once all the messages were processed, the *Timer* object was able to check immediately for new messages. The users then received the responses from the central server, with mobile phones taking on average 13 seconds to receive an SMS message, whereas email messages took on average 9 seconds to be received and viewable on a user's email account. These results demonstrated that the platform was able to provide a rapid transaction time for a basic task when handling messages from 20 users.

To resolve a batch of 20 messages with higher analysis complexity the platform took up to 45 seconds for the analysis process. The longest response time was found to be analysing input messages based on the "Risk Assessment" *Incoming* template. There were 20 risk assessment messages received from different users, each being associated with a unique and separate *Missing Person* instance. The TAU was required to extract the risk assessment data, ensure the data was valid according to *Answer* entities and calculate the associated missing person's risk status within the application module. The entire process, for the batch of 20 messages, took 44.6 seconds to resolve. This is approximately three times longer than a basic template request.

The total time to process messages based on the distinct features of the platform was also tested for batches of 20 messages. In comparison to text-only messages processing a batch of email messages containing image files did not have a large effect on the performance of the platform. These multimedia messages took on average 30 seconds to resolve, which included extracting and storing the image files. Sending out image files took slightly longer, taking 37.4 seconds to send out email messages to 20 different users. This is due to the platform's central server being



required to contact the email server when sending out messages, whereas *Outlook* handles this step for the retrieval of messages.

An important aspect of the *Missing Persons* scenario is the ability for one user to request the addition of new users. In order to assess the performance of the platform in relation to adding new users to instances of the *Missing Persons* scenario, two sub-tests were carried out. In the first sub-test 20 users on separate instances sent input messages, based on the “People Involved” *Incoming* template, to recommend a new person to assist in the instance. The total time to resolve the 20 messages and send new messages to the newly added users was 42.4 seconds. The second sub-test involved one user sending 20 requests on a single *Missing Person* instance to add 20 new users. The second sub-test was slightly quicker at 39.3 seconds. Overall, the average time to add new users was 40.9 seconds. This is an acceptable timeframe as new users can be notified within real-time and subsequently the central server would be ready to accept a new batch of messages, before *Outlook*’s next “Send/Receive” request to the email server.

The *Missing Persons* scenario requires all users assigned to an instance to be notified of new updates, such as suspicious vehicles being reported or the missing person being found. After receiving a message from one user reporting a suspicious vehicle, the central server took 4.8 seconds to send out a broadcast message to the 19 other users assigned to the same instance. Processing a batch of 20 messages, in which each message required the central server to broadcast updates to all 19 other users, took 37.6 seconds. This amounted to the central server sending 380 messages out in a short timeframe, where all the messages were then received by the users’ mobile devices in less than 20 seconds. The majority of the work was achieved by either the relevant external email servers that deliver email messages to user email accounts or the SMS web service that forwards text messages to mobile phones. Both of these information services were able to handle and forward the large quantity of data rapidly and with limited performance degradation, as shown by the similar timeframe to the other complex message types that have been tested.

Evaluating the platform’s performance at creating and allocating multiple new assignments was achieved from 20 users, on the same *Missing Person* instance, sending input messages based on either the “Important Locations” or “Sightings”

*Incoming* templates. During the analysis stages of each message the TAU created a new assignment to assign an available user to investigate the respective location or sighting, based on their relative geographical position. The platform was effective at generating and allocating the 20 new assignments, amongst the group of users added to the instance, with an average time of 43.4 seconds to prepare all the assignments and inform the relevant users. Each assignment was created correctly, only being assigned to a user who was not currently on another assignment and who matched the assignment's location. The only exceptions occurred in circumstances where no match was found, whereby a random available user was allocated to the assignment.

The results from this stage of the scalability test illustrate that processing messages from 20 concurrent users took between 15-45 seconds depending on the analysis complexity of each message. Even for the most complex message type, in this case being the risk assessment messages, the central server would still be ready for the next batch of messages that are received by *Outlook*. This was due to the minimum configurable time of once per minute for *Outlook* to make a "Send/Receive" request to check for new messages from the email server. Consequently, the total timeframe for this user group would be under two minutes, which includes the time for users to send the original message, *Outlook* to retrieve each message from the email server and the users to receive the responses. This is an acceptable timeframe, resulting in users being able to receive rapid responses to inform them of instance updates or new assignment instructions.

This stage of the scalability test was performed with users interacting via a combination of SMS and email messages. The time differences between these two communication methods were minimal. Email was on average 2 seconds faster than SMS for the central server to receive input messages and 4 seconds faster for the users to receive output messages. Selected message types for the *Diet Diary* scenario were tested in the same way. The timeframe for processing a batch of these messages from 20 users was between 17-39 seconds depending on the analysis complexity, where calculating the calorie targets took the longest processing time. Table 7.1 shows the average timeframes for each message type tested in both the *Missing Persons* and *Diet Diary* scenarios.

Message Type (Missing Persons)	Initial setup	Users send Image of missing person	Risk Assessment form	Request new users to join	Distribute Image to new users	Broadcast from one user's update	Broadcast from 20 users' update	Assignment creation from reported sightings	Processing results from assignments	Creating news records from user updates	Request list of templates
Time taken (seconds)	31.1	30.0	44.6	40.9	37.4	4.8	37.6	43.4	36.6	28.5	15.2

Message Type (Diet Diary)	Initial setup	Input user details	Calculate calorie targets	Search Food Name	Auto food entry	Manual food entry	Manual activity entry	Input calorie alert	Receive alert after food entry	Request daily summary
Time taken (seconds)	29.0	33.3	38.5	25.6	32.9	23.6	26.4	17.2	36.7	18.1

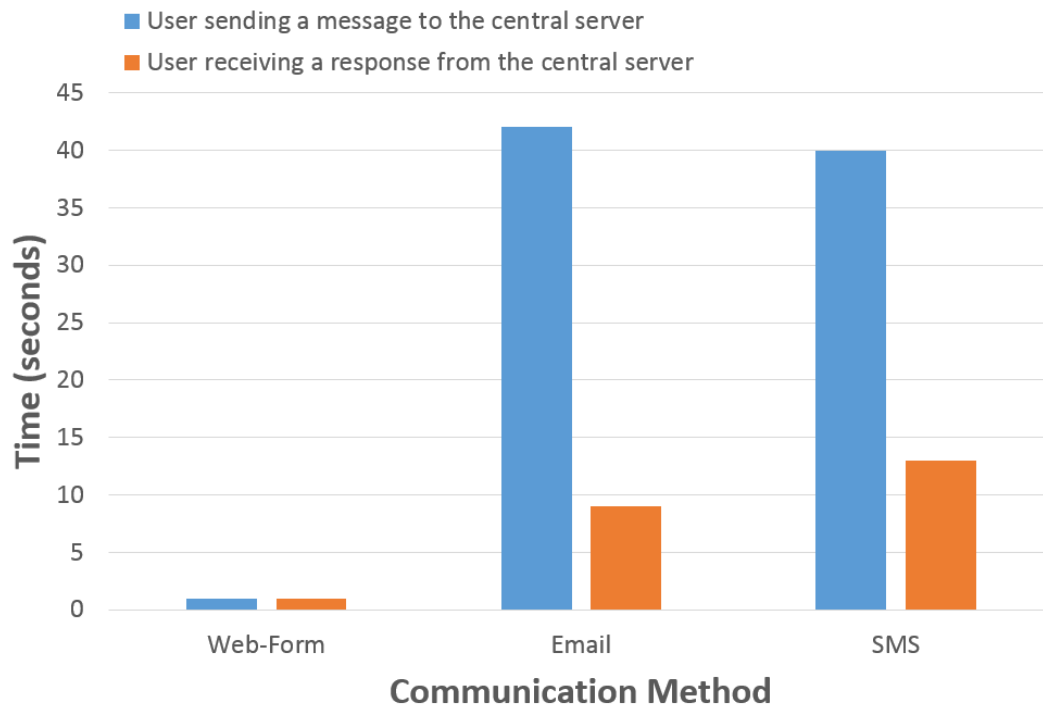
**Table 7.1:** Timeframes for processing a batch of 20 messages on the Missing Persons scenario (top) and Diet Diary scenario (bottom)

The performance of the platform's presentation layer is evaluated in the second stage of the scalability test. The presentation layer runs separately from the logic layer, enabling multiple users to access the web-based user interface concurrently for interactions with the central server. Calculating the processing time for messages sent via the web-based user interface was achieved separately to SMS and email messages. A request would be sent to the TAU to process a web-based message as soon as the user has submitted the data in the *UserMessageSend* web-form. Therefore, in order to calculate the time taken for multiple messages to be processed, a test script was developed to automatically send 20 duplicate messages via the web-based user interface when a user submitted a single message.

Response times for bulk sending web-based messages were considerably faster than via SMS or email, where the time to process all 20 messages ranged from 7-22 seconds depending on the complexity of analysis required. These faster results were due to both removing the need to retrieve messages from the *Inbox* folder and the time required to construct an email message or HTTP request, to the SMS web service, for a response.

Furthermore, the messages were processed immediately since there was no delay for the *Timer* object or *Outlook* to make a "Send/Receive" request to the email server. Once a response had been generated by the TAU it was saved to the *Principal* database. The user could immediately view any replies in the *UserMessageView* web-form. The web-based user interface automatically redirected the user to this web-form, which took a negligible amount of time to load on the web browser. The chart in Figure 7.34 illustrates the average differences in time taken for both a user sending a message and the central server sending a response via all three communication

methods. For both SMS and email, on average, it would take 30 seconds for Outlook to retrieve a new message from the email server via a “Send/Receive” request. This is added to the time taken for a user’s message to be initially received by the email server. Figure 7.34 shows that communication via the web-based user interface is considerable faster for a user, in both directions, than by the other two available methods.



**Figure 7.34:** Comparison time taken between the three communication methods for a user to both send a message to the central server and to receive a response

However, a few limitations were found whilst using the web-based user interface. The tables in the *Principal* database were inaccessible whenever they were being updated with new data, due to the current process placing a write-lock on the table. This denied access to the respective table for other processes. The TAU often required access to the *Message* table, in order to store received and newly created messages. This resulted in occasions where a user was delayed when attempting to view their previous communications with the central server via the *UserMessageView* web-form, with the web request not being resolved until the previous process was completed. Additionally, users would be delayed when submitting a message in the *UserMessageSend* web-form if the TAU was currently running through a batch of messages that had been retrieved via alternative communication methods.

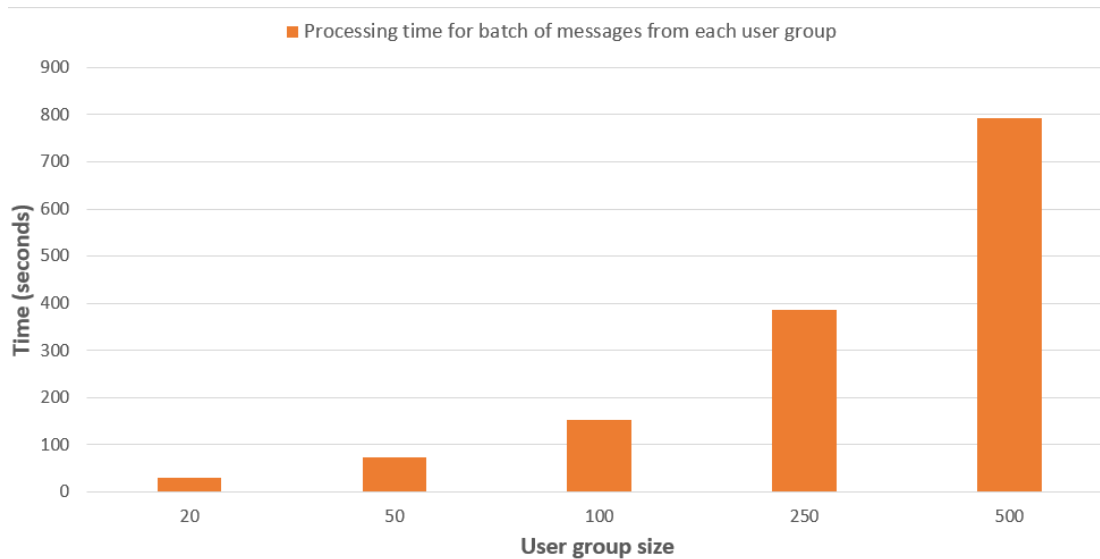
Consequently, these circumstances resulted in reduced performance of the web-based user interface for users.

The third stage of the scalability test involved evaluating the rate of performance degradation of the platform when deployed with a considerably larger user group. In its final deployment the platform would be expected to support an ever increasing number of users as new multi-user scenarios are introduced with concurrently running instances. Therefore, it was necessary to find the current limitations of the platform in relation to the number of users that can be simultaneously handled in order to identify whether functional improvements would be required.

The same 20 users were employed for this stage of the scalability test, with the users being assigned to a variety of newly created instances of both the *Diet Diary* and *Missing Persons* scenarios. Initially, 50 input messages were constructed on the user side based on a range of different *Incoming* templates, which ensured the TAU would be resolving messages with different analysis complexities. The *Timer* object was disabled for this stage to ensure all messages would be in the *Inbox* folder before the TAU started to retrieve messages. This enabled a simulated experience for operating with 50 users. The total processing time for these 50 messages was 1 minute 13 seconds.

This test was then repeated by the platform for the retrieval of 100, 250 and 500 messages in order for the platform to simulate running user groups of these sizes respectively. The processing time for these batches of messages increased to 2 minutes 32 seconds for 100 messages, 6 minutes 27 seconds for 250 messages and 13 minutes 12 seconds for 500 messages. The chart in Figure 7.35 illustrates the rising timeframe for processing messages as the user group grows in size. During these periods the TAU was unable to perform any other operations, including not being able to retrieve subsequent messages from the *Inbox* folder. The total processing time for 100 messages was acceptable since the central server could respond to all the user messages in under 3 minutes, allowing users to receive further instructions or updates fast enough for proceeding with their roles in the active instances. However, depending on the scenario a delay of over 6 minutes for a user group of 250 users may not be acceptable since the users could be looking for a rapid response to receive further instructions. The processing time of over 13 minutes for 500 messages for

time-sensitive communication is excessive since it prevents a real-time response from the platform.



**Figure 7.35:** Timeframes to process a batch of messages from an expanding user group

Ensuring the platform's response time does not become too high is essential. The third stage of the scalability test illustrated that the platform started to lose its ability for real-time responses when supporting user groups in excess of 100 in size. As new scenarios are integrated into the platform the user-base will grow in order to assist new instances of these scenarios. This would then entail that the larger user group sizes, simulated in this test, would eventually need to be supported. Additionally, it is necessary to ensure the waiting time for user requests via the web-based user interface are kept to a minimum. Data requests from other processes to the *Principal* database have shown to result in an impact on the performance of the web-based user interface in supporting user interactions.

The performance of the platform could be improved by introducing functionality for parallel processing into the application code. With several processes running in parallel the analysis of multiple input messages could be handled concurrently, thus reducing the time to respond to each message. Further details of this solution are discussed in the scale-up study in Chapter 8 (Section 8.2.2.1).

Nonetheless, the prototype in this project has been developed to provide a novel integrated communications platform that enables rapid development of new scenarios

and a generic feature-set that can be tailored to each scenario. The prototype can still provide fast responses when supporting a user group of 100 users, whereby users would be sent a response in under 3 minutes, in the worst case, after the central server has retrieved messages from the *Outlook Inbox* folder. Furthermore, operating the platform with 20 users showed that the platform could support users assigned to numerous instances in real-time and perform the distinct features, such as assignment allocation, broadcast and multimedia messaging, with minimal performance degradation at this level.

Feature	Supported by the Platform	Use in scenario	Generic provision from the Platform	Additions/changes required
User can send field data	Yes	Field data used to generate new reports, such as sightings.	<i>TemplateHandler</i> class and <i>Template</i> entities extract field data from user messages.	None, platform provided full functionality
User can send/receive feedback	Yes	Results of processing received messages include notification to the user and broadcast updates to other users in the instance.	<i>ActionHandler</i> and <i>DataOutHandler</i> classes create feedback messages based on <i>Outgoing</i> templates. <i>ResponseGroup</i> attribute of <i>Action</i> entity indicates which users replies are sent to.	None, platform provided full functionality
Exchange of pictorial data	Yes	User can send an image file with messages to share a photograph of the missing person. The image file can be distributed by the server to other users of an instance.	<i>Template</i> entities have attributes for specifying whether a message can include an image file and the storage location for that file.	None, platform provided full functionality
Alerts and reminders	N/A	Not required by scenario		
Database/record keeping	Yes- Partial	Field data sent from users and assignment results are stored for later use. Details of new users added to the platform are stored to enable future communication.	<i>Principal</i> database stores records on templates and messages. New records are created to store personal and contact data for users who are requested to join the platform.	New scenario database required to store field data in meaningful locations. Records stored on missing person, risk assessment data and each reported finding.
Templates for individualised messages	Yes	User sends field data for set-up details, reported findings (e.g. sightings) and assignment results. Server instructs user through set-up of instance, provides assignments to individual users and updates multiple users on significant events.	<i>Incoming</i> templates assist user to create input messages for sending field data on reported findings and assignments. <i>Outgoing</i> templates utilised by server to create output messages for informing a user of a new assignment and broadcasting instance updates to users.	None, platform provided full functionality
Autonomous data analysis	Yes- Partial	Processing reported findings and assignment results. Analysing risk assessment data to determine the risk status of the missing person.	<i>TriggerHandler</i> and <i>AnswerHandler</i> classes analyse each message. <i>ActionHandler</i> class handles response actions, such as adding new users, creating assignments and generating news reports.	New application module required for computing the risk status.
Anonymous sending	Yes	New users can provide field data on reported findings anonymously with no personal data held regarding the user.	"Anon" keyword enables platform to acknowledge that the message sender wishes to remain anonymous. User's contact data deleted after message has been processed.	None, platform provided full functionality
Collaboration/coordination of users	Yes	Assignments may be generated for each reported location and sighting. Different users are allocated to each assignment.	<i>ActionHandler</i> class creates new assignments with <i>AssignAdd</i> action and removes users from assignments with <i>AssignRemove</i> action.	None, platform provided full functionality
Location-based features	Yes- Partial	Each assignment is allocated to a user who is close to the assignment's location.	New module provided functionality	<i>LocationHandler</i> module developed to allocate each user to an assignment based on their location data matching.

*Table 7.2: Missing Persons scenario – Features Supported by the Platform*



## 7.7 Conclusions

The *Missing Persons* scenario has been developed to evaluate the platform's ability to support a scenario with substantially different requirements and objectives to that of the *Diet Diary* scenario. This makes it possible to investigate the platform's effectiveness at handling a range of diverse application problems. The *Missing Persons* scenario focuses on interactions with multiple users to coordinate them in order to search for people who have been reported missing. This scenario also enables the testing of the platform's features that are not required by the *Diet Diary* scenario. Table 7.2 summarises the list of features, detailing their use within the scenario and the extent to which the generic functionality of the platform supports each feature.

The communication component works together with the template mechanism to facilitate communication between the server and each user of an instance. This is achieved through the use of the underlying TAU classes and the declarative creation of template criteria by the builder. The declarative creation of templates utilises the same procedures that were discussed for the *Diet Diary* scenario in Chapter 6, which are achieved via the builder's interactions with the web-based user interface. A major difference with the *Missing Persons* scenario in comparison with the *Diet Diary* scenario is the response possibilities available to the server. Processing an input message could now result in a reply to the same user, a new message sent to another user or a broadcast update to many users participating in the same instance. The stages and outcomes are all handled by the TAU and *Action* entities predefined for each template. Additionally, the *Template* entities include attributes that facilitate the storage and retrieval of multimedia files for both input and output messages that contain attachments. The platform is able to receive an image file containing a photograph of the missing person and then distribute the file to other users who are participating in the same instance. There is no extra programming required for these new components as the bulk of the scenario is created with the necessary functionality provided by the platform, enabling a rapid development and implementation process.

The *ActionHandler* class together with the *Action* entities has a versatile range of attributes that support the new features required for the operations of the *Missing Persons* scenario. There is only a minimal need for more modules to run the scenario.

The risk assessment calculation is the only aspect of this scenario that requires analysis functionality not supported by the TAU or reusable for other scenarios. The remainder of the scenario is handled by the TAU functionality. The *LocationHandler* module has been developed in parallel with the *Missing Persons* scenario. However, this module has been designed to be reusable for future scenarios that feature location-dependent assignments. Moreover, this means that the *LocationHandler* module can receive updates at a later date, benefiting all scenarios that utilise this module. Therefore, the emphasis of each scenario is to utilise current functionality as opposed to new hard-coded modules, thus reducing the complexity of designing a scenario.

Implementing assignment functionality has provided a practical method of coordinating the multiple users participating in an instance. The TAU can rapidly respond to message reports from one user by creating a new assignment and allocating another user to this assignment. Therefore, the interactions from one user influences the actions that are to be performed by another user, whereby the *LocationHandler* module dynamically selects this new user based on their location. This scenario demonstrates that broadcast communication can be used alongside the assignment functionality to update all users on the results of an assignment, making it possible for the other users to keep track as the instance progresses.

Furthermore, the inclusion of the anonymous user feature offers the opportunity for users to report helpful information safe in the knowledge that their personal details would not be stored by the server. This feature could encourage more users to become involved, in a similar way to the *Interactive Learning* application [95] that was investigated in Chapter 3. The information gathered by these anonymous users can be processed by the TAU to generate assignments and notify other users of new events. These new features work together to provide a framework for the platform to instruct and coordinate groups of users, enabling these users to solve issues of an instance as they arise.

The scenario's database is mainly designed for the purpose of record-keeping, with field data received from each new user report being stored in a record in the correct table. The scenario's database operates by the same procedures as the database that was developed for the *Diet Diary* scenario. The extracted field data taken from input

messages is collected by the *DatabaseHandler* class, which uses *DataSave* entities to insert and update database records. The reported data can then be accessed at a later stage of the instance for the sending of output messages to users or for assisting the operator. The expansion of new users demonstrates the way that the scenario's database collaborates effectively with the *Principal* database. The details of each new reported person are stored in a record within the scenario's database. The contact details can then be transferred to the *Person* record in the *Principal* database, creating a new user.

The scalability test demonstrated that the prototype of the platform is able to support a user group of up to 100 users and provide real-time responses to each user participating in instances of the *Missing Persons* scenario. In order to support larger user groups new techniques for processing and handling data need to be applied to the platform, such as parallel processing.

The two scenarios that have been described in Chapters 6 and 7 have both demonstrated the functionality of the platform and the generic framework that enables the support of scenarios with differing objectives. The underlying components of the platform provide the infrastructure to create scenarios in a primarily declarative and rapid process through the use of the three-stage scenario development process. This development process has been shown to reduce the time taken to implement the *Missing Persons* scenario from 31 days to only 3 days, which includes 1 day for the *LocationHandler* module. Furthermore, the platform's web-based user interface provides a simplified approach for defining the analysis criteria of a scenario in comparison to hard-coding classes for a new application.

The *Missing Persons* scenario reveals that as new scenarios are developed the platform can also be expanded to support further modules when new requirements are identified. This is demonstrated by the inclusion of the *LocationHandler* module. Subsequently, as the platform is updated and expanded the complexity of incorporating new scenarios should be reduced and the speed of implementation increased, resulting in an ever more efficient and resourceful infrastructure.

## 8 Conclusions and Future Work

This chapter highlights the contributions that have been brought to the field by the development of the *Connected-Mobile Platform*. This includes discussing the benefits of the platform's novel framework, whereby its generic properties and features are examined to evaluate their effectiveness in facilitating the platform to support and run a multitude of scenario types in a mobile environment. This is achieved by investigating the running of the two scenarios that were developed; the *Diet Diary* and *Missing Persons* scenarios. The development process of both scenarios is discussed to determine if the platform meets its objective of ensuring that new scenarios are incorporated in a structured approach, with minimal time and effort required for implementation. Issues that have been identified with the running of the platform are discussed, with new solutions and ideas brought forward in the future work section.

### 8.1 Conclusions

The *Connected-Mobile Platform* has been developed, which is based on a client-server architecture. The *Remote Experimentation* system [15] and *HELP* system [101] used this type of architecture to ensure the intensive processes and data storage requirements were withdrawn from the user and performed from a central location. These ideas were built on and improved in the *Connected-Mobile Platform*, where there is a focus on the autonomous analysis and response from user messages by the server. Unlike the *UbiquitousSurvey* system [13], the *Connected-Mobile Platform* reduces the role of a human operator. The server has the ability to handle the flow of communication with each user, progress instances of scenarios using the results of field data analysis and coordinate multiple users to resolve objectives.

The users of the platform are able to participate in active instances of a scenario, where their only requirement is to own a basic communication device that supports the sending and receiving of text messages over mobile networks or the internet. Field data extraction and analysis is performed by the server as soon as messages are received, enabling each of these steps to be achieved in real-time. The analysis and storage of field data is managed by the server. The server responds to the user with the results and feedback that have been generated from this data analysis. The location of each user and the capabilities of their devices do not have any bearing for

the purposes of supplying field data to the server or receiving messages in response to the data analysis.

The beneficiaries of the platform changes for each scenario that is developed. For the *Diet Diary* scenario, the beneficiaries would be the individual users who request to start new *Diet Diary* instances. The communicated data for each *Diet Diary* instance would only be relevant and applicable for the associated user. The user would be the anticipated beneficiary for other types of scenarios that focus on single-user instances. In the case of the *Missing Persons* scenario the main beneficiary would be the police. The scenario builder would collaborate with the police in the development of this scenario. The operator would then cooperate with the police during the running of each *Missing Person* instance.

### **8.1.1 Support of multiple communication methods**

An objective of the platform has been to enable interaction with end users by incorporating multiple sources of input and output. The server is able to communicate with as many potential users as possible and remain in contact at all times in order to support each user that is assigned to an active instance. The platform has been created to support communication with users via communication methods that are practical for use in a mobile environment. The majority of people would have a mobile phone or another type of mobile communication device with them at all times in a ready to use mode [44]. Therefore, incorporating mobile communication methods into the platform provides flexibility to the users as they are able to interact with the server at any time, regardless of the user's geographical location [8, 4].

Chapter 2 discussed how the SMS communication method is the standardised method of text messaging in the mobile environment [39]. However, communication via email enables a device to make use of advanced features, which include the ability to send and receive multimedia files [10]. The research undertaken in Chapter 2 illustrated that it is advantageous to utilise multiple communication methods, as each individual mobile communication device could be more suited to one particular method.

The platform has been designed to utilise three different communication methods to cater for as many device types as possible and also to offer flexibility in the way that

the users can interact with the server. The three communication methods that have been incorporated into the platform are SMS, email and web-forms via a web-based user interface. To interact with the server a user's mobile device only requires the capability to communicate via one of the supported communication methods. The user can take advantage of the different benefits each communication method offers by interchanging between the methods, which would then update their preferred choice for the server's method of sending a message. Therefore, it is the user who determines the communication method for both the sending and receiving of messages in the client-server architecture of the platform.

The *Diet Diary* scenario demonstrated the ease with which a user switches between the communication methods, whereby the server would update the user's preference on each occasion the user sends a message via an alternate method. In this scenario the web-based user interface provided the user with a more user-friendly approach to interacting with the server, where the user could navigate between previous messages and directly select *Incoming* templates for the sending of new input messages.

The *Missing Persons* scenario showed that for a multi-user instance the server would respond separately to each user, based on the user's preferred communication method. This scenario also highlighted the platform's ability to handle the exchange and storage of image files. The platform enables users of camera-integrated devices to share visual events through the use of email [10]. In a *Missing Person* instance users are able to send and receive photographs of a missing person to help identify and locate the person. Each of these features is effectively handled by the platform's communication component to provide a diverse range of options for users to communicate with the server.

The users are also able to participate and assist anonymously in active multi-user instances. This feature utilises the principles of the *Interactive Learning* application [95] to encourage people, who may not wish to be identified, to assist in an active instance of a scenario. In an instance of the *Missing Persons* scenario, a user is able to request anonymity and then send in message reports to assist in the finding of the missing person. This was demonstrated in Chapter 7 with an anonymous user reporting a suspicious vehicle. Details of this vehicle were then broadcast to the other users participating in the instance to notify them of the situation.

### 8.1.2 Features of the novel framework

The *Connected-Mobile Platform* has been developed using a novel framework that has generic properties to support the running of a range of different scenario types and to provide a structured development process for the incorporation of each scenario. This has been achieved by identifying the necessary functionality to include in the platform and then designing the platform's components so that this functionality could be tailored to meet the requirements of each new scenario.

In Chapter 3 a range of applications were researched to identify the functionality that would need to be included in the platform to support a range of scenario types. Key to this was the need for the platform to effectively communicate with users via a structured process. The template mechanism enables the exchange of messages between the server and its users. Chapter 3 discussed how the *Attendance Improvement* applications [93, 94] sent personalised messages to their users for appointment reminders, applying a template content structure. The *Connected-Mobile Platform* provides a more comprehensive template mechanism as the template layouts are built for both input and output messages. Users are supplied with a structured message layout, through the use of *Incoming* templates, for supplying field data to the server. The scenario builder is able to design each template to meet the requirements of its associated scenario. This makes it possible for field data that is sent from a user to be extracted and processed by the platform. *Outgoing* templates can then be utilised by the server to autonomously generate and send responses.

The *Incoming* and *Outgoing* templates work together to facilitate the flow of information. In the *Diet Diary* scenario a user employs *Incoming* templates to provide updates on food items consumed, through the input of data relating to a food item. *Outgoing* templates are used to help the user search through the food categories for a specific item. Each time the server receives details of a new food item, feedback regarding the user's daily diet targets can be sent within a personalised message to the user.

In the *Missing Persons* scenario users are able to provide reports on the missing person, including risk assessment details and sightings, through the use of *Incoming* templates. *Outgoing* templates enable the server to notify users of updates to a *Missing Person* instance as new information is received. Both of these scenarios have

demonstrated how the template mechanism provides an effective approach for designing and tailoring a communication structure to different scenario types.

The platform includes a multifaceted and adaptable data layer to meet the need of storing the field data received from user templates. Chapter 3 discussed various applications that utilised a database for the storage and retrieval of data. For example, the *UbiquitousSurvey* system [13] would store the data from user messages in a database for later use by an advisor. The *Connected-Mobile Platform* attempts to offer a multi-layered approach to data storage. The *Principal* database provides a standardised approach for the storage of data relating to users of the platform, exchanged messages, *Incoming/Outgoing* templates and analysis criteria. Each scenario then has its own dedicated database, which is linked to the *Principal* database. A scenario database can be individually designed to focus on the requirements of its associated scenario. The scenario databases, together with the methods of the *DatabaseHandler* class, enable collected field data and analysis results to be stored in records that are unique to the associated scenario.

This is demonstrated in the *Missing Persons* scenario, where there are data-tables for each type of issue that a user can report information on concerning a missing person case. This enables the platform to follow the correct procedures for a missing person case [197], by both keeping records of new information and results from assignments in appropriate locations. Each record in the scenario's database would be linked to both the instance and the corresponding messages via the *Instance* identifier in the *Principal* database. Therefore, the *Template Analysis Unit* (TAU) can readily locate the field data for either further analysis or to send this data to a user of the associated instance at a later time.

### **8.1.3 Autonomous functionality for real-time responses**

The platform set out to incorporate autonomous analysis functionality within its generic framework. Providing a comprehensive analysis component enables an application to interact and respond to data received from users in real-time. The *Telemedicine Monitor* application [99] implemented autonomous functionality on the server-side to analyse blood glucose measurements, which are received from a user's device. However, the platform's TAU is far more extensive as it is equipped with a broad range of analysis and response functionality to cover the processing of user



messages and analysis of field data for each type of scenario. This is achieved through the use of *Answer* and *Trigger* entities, whereby the scenario builder tailormakes analysis criteria for each specific scenario. Received field data is then tested against a range of defined criteria.

The *Diet Diary* scenario demonstrated how the TAU, through the analysis of received field data from a user, can be effectively used for autonomous decision making. For example, the TAU is able to send a warning message to users when they are approaching a calorie target value by observing whether a new food or activity item would breach a pre-defined alert limit. However, this scenario also highlighted the lack of the TAU's ability to perform mathematical calculations. The calculation algorithms for this scenario had to be created from scratch, within the scenario's application module. This increased the development time of the scenario and highlighted the need of a generic calculations module within the TAU.

The TAU's final stage involves performing actions that are based on the analysis results of field data. The *Missing Persons* scenario demonstrated the variety of actions that are at the TAU's disposal. This ranges from sending simple replies to the message sender to the broadcasting of updates to all the users participating in an instance. Other types of actions include the ability to add a new user to the instance and generate *News* records to update the operator on significant events that have occurred. For the *Missing Persons* scenario the TAU's *ActionHandler* class and *Action* entities have been able to offer a versatile range of attributes to support each type of action. The TAU reacts appropriately in real-time to each message received from a user, thereby meeting the platform's objective of performing actions swiftly based on the analysis of received data.

The platform is able to support a user group size of 20 users providing fast responses to received messages, including occasions when all the users send a message simultaneously. Response times for resolving a batch of 20 messages are between 15-45 seconds. This means that a user can expect to receive a reply within one minute of the central server receiving their message. The response time increases as more users join the platform, with users having to wait for up to 3 minutes where the user group size is 100. This should still be an acceptable timeframe for the majority of users. However, a user group of 500 would result in a response time of around 13 minutes.

Scenarios that involve the exchange of time-sensitive data may need a response time faster than 13 minutes. In these circumstance the platform's requirement of providing real-time responses would not be achieved. This would be an important performance issue to resolve prior to the deployment of the platform for real-world use since the user group would be expected to expand as more scenarios are integrated into the platform. Techniques to improve the performance of the platform are discussed in the scale-up study in Section 8.2.2.1.

#### **8.1.4 Assignment functionality for coordination of users**

One key action type that is available to multi-user scenarios is the ability to generate location-dependent assignments and allocate users to these assignments. The decision to implement these features into the platform was based on the research of the *UbiquitousSurvey* and *Road Safety Alert* systems. In the *UbiquitousSurvey* system [13] human advisors are able to assign tasks to the users in the field, enabling these users to work together in order to tackle a common goal. The *Road Safety Alert* application [100] uses location data of the locations travelled by subscribed motorists in order to inform them of arising hazards in those locations.

The *Connected-Mobile Platform* has aimed to combine the two features of location-based technology and assignment functionality to make efficient use of the collective intelligence of the user group. The TAU's *LocationHandler* module handles the allocation of users to assignments. This module determines the most suitable user to be allocated to a location-dependent assignment, based on the location data stored on each user's most recently updated geographical location. Therefore, assignments can be distributed in an efficient manner in active instances of scenarios where the platform can make use of the different locations of each user.

The use of these features has been demonstrated in the *Missing Persons* scenario, where assignments are generated and users allocated to these assignments each time a new location or sighting is reported. Allocating a user to an assignment who is closest to its reported location means that it is more likely for a quicker search and response occurring than through selecting a random user, who could be at a considerable distance to that location. The template mechanism works in conjunction with this assignment functionality to offer a means for the server to provide users with instructions on new assignments and for the user to report back on their findings from

the missing person searches. One of the platform's objectives was to deploy its users to negotiate and solve issues of an active instance of a scenario. During an active *Missing Person* instance the assignment functionality offers a practical method for the coordination of multiple users in order to achieve this objective. In its current stage, the *LocationHandler* module only offers a basic method for selecting users in a close proximity to an assignment's location. This module could be updated to include a more comprehensive method that does not only rely on matching exact location data.

#### **8.1.5 Rapid scenario development process**

The integration of multiple scenarios has been made possible by the three-stage scenario development process that has been defined for the platform. This process was employed for the development of both the *Diet Diary* and *Missing Persons* scenarios, with the aim of reducing the time and effort required for implementing each scenario. To help reduce the amount of additional development requirements for any particular scenario each component of the platform has been designed to be generic in supporting the running of different scenario types.

The *Diet Diary* scenario takes 3 days to complete the development; 1 day is required to build the scenario's database and for the declaration of tailored template criteria, with a further 2 days required for the programming of the application module. The scenario builder's use of the web-based user interface emphasised the ease with which the *Template*, *Analysis* and *Action* entities could be created. There is no requirement for programming new modules in this aspect, with the builder declaratively defining each entity. The builder navigates and submits data in the web-based user interface to tailor new entities to the requirements of the *Diet Diary* scenario.

The creation of the application module has highlighted a drawback of the current platform as all the algorithms for calculations in the scenario had to be developed from scratch. This application module contains 560 lines of code, consisting of the methods that run these algorithms. Nonetheless, this is still a considerable reduction in lines of code required in comparison to developing an entire diet diary application from scratch. Furthermore, the time required to develop a new application that performs the same functionality is estimated at 22 days. This is over 7 times longer than the development time for the *Diet Diary* scenario, highlighting the benefit of the

platform's rapid scenario development process. The reason for the reduced development time is due to the platform's framework of generic components removing the need to develop additional modules. These components have been put in place to provide a communication structure, template mechanism, data analysis and response procedures for the processing of field data and handling of active instances. The demonstration of an active instance has shown that the *Diet Diary* scenario is supported by the functionality offered from this framework of components.

The *Missing Persons* scenario required a timeframe of 2 days to develop and implement into the platform, with an extra day for the *LocationHandler* module. In comparison to the *Diet Diary* scenario, a shorter time was required for the development of the *Missing Persons* scenario's application module. The *Missing Persons* scenario has a reduced emphasis on mathematical calculations, where most of the analysis is capably managed by the generic analysis stages of the TAU. This scenario's application module only required 140 lines of code for the *RiskStatus* algorithm. Instead, an extra day's development was focused on integrating the new *LocationHandler* module. This is more practical than the development of the *Diet Diary*'s application module as the *LocationHandler* module has been designed to be reusable for future scenarios. Consequently, this module has not been included as part of the scenario's total development time.

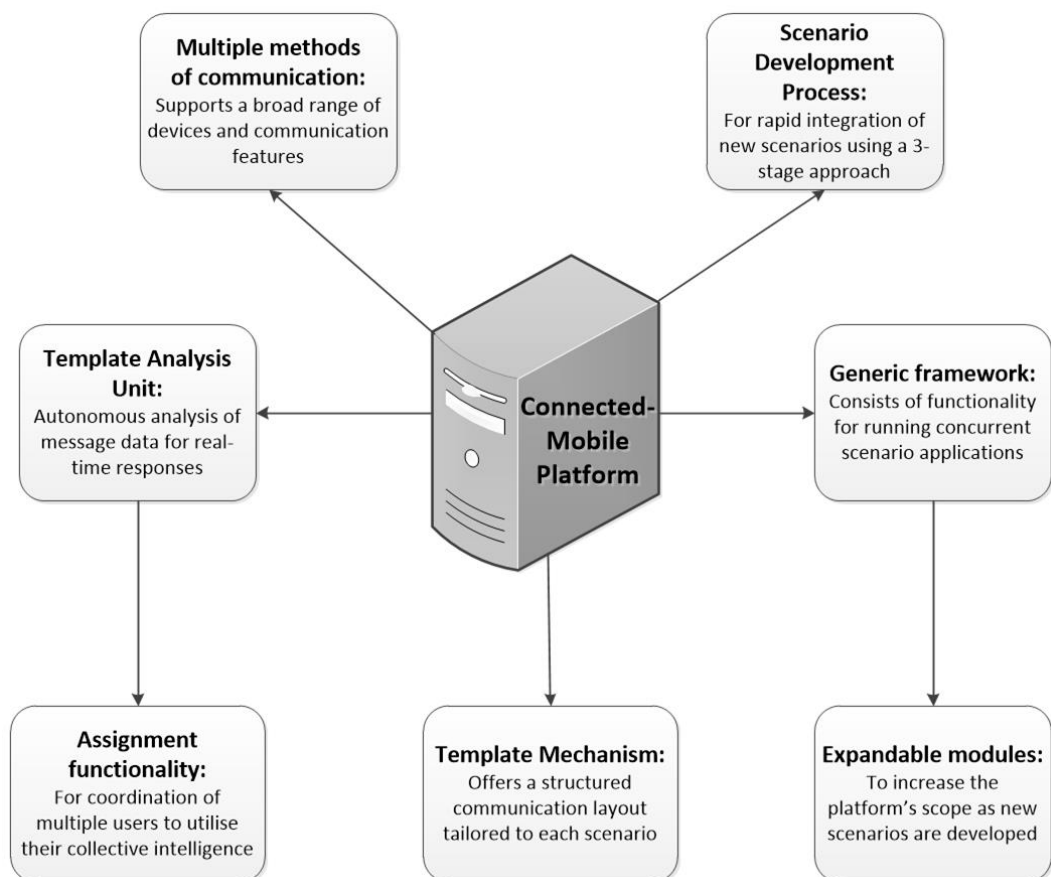
The development of *LocationHandler* module in the *Missing Persons* scenario has also demonstrated how the middleware of the platform is expandable for the inclusion of features that may be identified in the future. The ability to include new modules is an important factor for the platform since practical features that are of benefit to multiple scenarios may be identified during the development of a future scenario. Also, introducing reusable modules would help to reduce the development time for new scenarios through eliminating the duplicate programming of algorithms.

Developing an application from scratch to achieve the same objectives as the *Missing Persons* scenario takes 31 days, which is over 15 times longer than the development time taken by following the platform's scenario development process. Furthermore, the guided process of defining template, analysis and action content via the web-based user interface is not available for the development of this separate application. Subsequently, this increases the complexity of creating the application as the

developer is required to construct new programming modules for each component and manually add message templates to the database. The components of the application need to be rigorously tested to ensure that there are no bugs or faults in the software. In comparison, the testing requirements are minimal for adding new scenarios to the platform since the platform's generic components have already been through testing. These points clarify the rapid and straightforward process offered by the platform to implement new scenarios.

### 8.1.6 Summary of contributions

The aim of this project has been to create an integrated communications platform that supports the development and implementation of a vast array of scenario applications, whereby people in a mobile environment can be utilised to tackle the issues of the scenarios via multiple communication methods. Figure 8.1 illustrates the significant contributions and features of the *Connected-Mobile Platform*.



**Figure 8.1:** Contributions of the *Connected-Mobile Platform*

Prior to the *Connected-Mobile Platform* it was necessary for developers to create applications from scratch when attempting to push information to or collect data from users to solve a particular issue in a mobile environment. The platform makes a significant contribution to knowledge since there is no other type of integrated communications platform currently available that facilitates this rapid development of complex mobile scenario applications. The comparisons between utilising the platform's scenario development process and developing applications without this foundation, detailed in the case studies of Chapters 6 and 7, have demonstrated the considerable savings in time and effort that have been made available by the platform.

The scenario development process is able to be applied to both the *Diet Diary* and *Missing Persons* scenarios, even though they have different requirements and feature usage. Existing works on developing applications for interacting with users in a mobile environment only focus on a specific focal problem, whereas the *Connected-Mobile Platform* is composed of a framework of generic components to fulfil the requirements of different scenario types. These components provide a comprehensive and expandable feature-set that can be tailored to resolve the objectives of each new scenario.

The platform set out to meet the project's objective of providing a foundation to support the running of each new scenario. The case studies of the *Diet Diary* and *Missing Persons* scenarios have established that this objective was achieved. The running instances of both these scenario types have illustrated the way different scenarios could be operated under this single platform. This is a key contribution of the project since the platform is able to support a multitude of different scenario types, whether they require a single user or multiple users.

The design of the template mechanism and TAU are further contributions provided by the development of the *Connected-Mobile Platform*. By providing a complex communication structure the template mechanism improved on previous applications that utilised templates to communicate with their users. The template mechanism allows a scenario builder to construct the framework of each type of message, facilitating two-way communication between the server and its users. Through this feature messages can be individualised and field data extracted by the server in a straightforward process. The *Principal* database operates in conjunction with each

scenario's database for the TAU to effectively analyse the field data and progress a scenario's instance, based on the analysis results. The TAU enables the server to respond autonomously to each user's message, catering for different types of scenarios through a process of defining individual analysis criteria for each scenario. These features bring the advantage of fast responses and an adaptable approach to tailor the communication structure and the analysis procedures.

The platform provides the means for users to interact via multiple communication methods. Therefore, the server is able to communicate with a broad range of mobile communication devices, whilst at the same time supporting the additional features offered by particular communication methods. SMS offers a simple means for users to communicate with the server using any mobile phone. On the other hand, communicating via email and the web-based user interface offers new features, such as multimedia messaging. The user has a high degree of flexibility in being able to choose the most convenient method of communication each time they interact with the server. Through these interactions, each component of the platform is utilised to provide constructive information to users in all types of scenarios.

The platform can autonomously provide users with tasks, through the use of assignment functionality to coordinate multi-user instances from a central location. This enables the platform to take advantage of the collective intelligence of the user group, in a mobile environment, in order to dynamically resolve problems that arise within each instance. The capabilities of the platform's assignment functionality are extended through the implementation of location-based technology. This is achieved by offering the ability to allocate a user to an assignment, based on each user's geographical proximity to an assignment's location.

All of these contributions have demonstrated that the platform meets the aim and objectives that were initially set out in Chapter 1, enabling the platform to effectively support a vast array of different scenarios. By following a three-stage scenario development process, new scenarios can be rapidly developed and implemented into the platform for use in a mobile society. These scenarios can range from basic one-to-one situations, where the platform is only engaged with one user, to complex scenarios that involve the cooperation of numerous users. Furthermore, the expandable nature of the platform, where it can support new modules to extend its

functionalities, would provide the platform with a more comprehensive feature-set as new scenario types are identified, developed and implemented.

## **8.2 Future work**

The *Connected-Mobile Platform* has been proposed as a means to provide a framework for the rapid development and implementation of scenario applications in a mobile environment. A prototype of the platform has been implemented to demonstrate these proposed ideas, with the *Diet Diary* and *Missing Persons* scenarios being developed to assist in evaluating the prototype and to determine its effectiveness in achieving the objectives of the project. Several areas have been identified where the prototype functionality can be expanded and improved. In this section, new features are discussed that could enhance the usability and scope of the platform with ideas for further research also being presented.

### **8.2.1 Improvements to the platform's functionality**

#### **8.2.1.1 Development of utility modules**

A large application module was created for the *Diet Diary* scenario in order to perform the multitude of mathematical calculations required for the running of each instance. Similar to the *LocationHandler* module, a new *Calculations* module could be developed within the TAU to perform these calculations. Research into regular types of calculations, which may be encountered by future scenarios, could be undertaken to determine the algorithms to be included in this module. The key aim of implementing a *Calculations* module would be to move to a more declaratively based creation of content when defining calculations to perform on scenario field data. This would be achieved by the design of new web-forms for the web-based user interface. The scenario builder would use these web-forms to define new *Calculation* entities for each scenario, which would be a similar procedure to the current method of creating templates.

One such scenario could be a sports betting application, whereby users would be able to interact with a bookmaker via the platform in order to make bets on sporting events and track their performance. New algorithms would be required to calculate the return value of each bet made by a user and to update their balance as bets are won or loss. If each type of calculation was available in a generic *Calculations* module then it



would reduce the programming effort by the scenario builder in developing the new scenario.

A *Calculations* module is one type of feature that has been identified for inclusion into the platform. To increase the scope of the platform's generic framework it would be necessary to research new scenarios and their requirements in order to identify other features that could be implemented into the platform. The purpose of this new research is to develop a host of utility modules that would be designed to work with any scenario that requires their functionality. This would lead to an expanding feature-set for the platform, resulting in the platform's functionality being more comprehensive for the coverage of future scenarios.

The *LocationHandler* module currently employs a basic method for determining the closest user to an assignment's location. Chapter 2 investigated the use of *Global Positioning Satellite* (GPS) technology to track mobile communication devices. Many devices, especially high-specification smartphones, have integrated GPS hardware for this purpose [79]. Currently, a user is required to manually inform the platform if their geographical location changes. The *Missing Persons* scenario has demonstrated that if the location details are not updated it could result in the user being given an assignment in a distant location to their current whereabouts. GPS technology could provide accurate and real-time data on a user's geographical location, to within a few metres, with the location data being stored internally on the user's device [78]. An application could be developed for GPS supported devices to continually send updates to the server as the user's location changes.

Furthermore, the server could utilise readily available web services, such as Google Maps [201], whereby the *LocationHandler* module measures the precise distance of each user from the location of an assignment. The *AroundMe* application [85] that is available for smartphones helps users to locate close businesses, based on their GPS calculated location. The platform would utilise a similar feature to the *AroundMe* application. However, calculations would be performed on the server-side to accurately determine the closest mobile device, based on an assignment's location. This functionality would ensure a more efficient method of assignment allocation than the procedure that is currently in place.

The error handling and security elements of the platform would need to be enhanced to support a large user group. In its current iteration the TAU includes limited functionality to check the validity of data from input messages. A new module would need to be developed to focus on rigorous testing of field data. This would help to prevent erroneous data interfering with the progress of an instance. Additionally, users should only have access to the necessary elements of the web-based user interface for the purpose of completing their tasks. Each user can be provided with a password for logging into the web-based user interface, where transmission of this password is encrypted to prevent unauthorised access and data entry from another user. This would ensure that only the authorised user on the receiving device can submit field data and receive messages from the server, which have been sent to their user profile.

#### ***8.2.1.2 New approaches to the user-interface***

The template mechanism has provided a simplified process for the TAU to extract field data supplied within input messages. However, throughout the running of the two scenarios the communication process on the user-side has at times been difficult to operate. This can be a cumbersome process for users since they are required to navigate through a pre-defined *Incoming* template, locate each parameter position, remove the temporary data and finally insert the field data. Devices with small screens and reduced navigational means, such as the old style keypad mobile phones, can make this process even more difficult for a user. The interface scheme for the SMS and email methods could be redesigned by removing the need to insert data between two asterisks. Instead of this, a template could provide a phrase to instruct the user on the information that is required which would then be followed by a separator, such as a colon. The field data would then be inserted after the colon, where a new line would mark the end of the data item.

There is a higher degree of functionality available for facilitating data input via a web-based user interface. Social networking sites, such as *Facebook* [108], have individual textboxes for each data item that is requested in a web-form. This approach simplifies the process of data entry, especially for touchscreen devices where the user only has to touch the screen location of a textbox to then be able to insert field data. Therefore, users of devices that can support this communication method would be

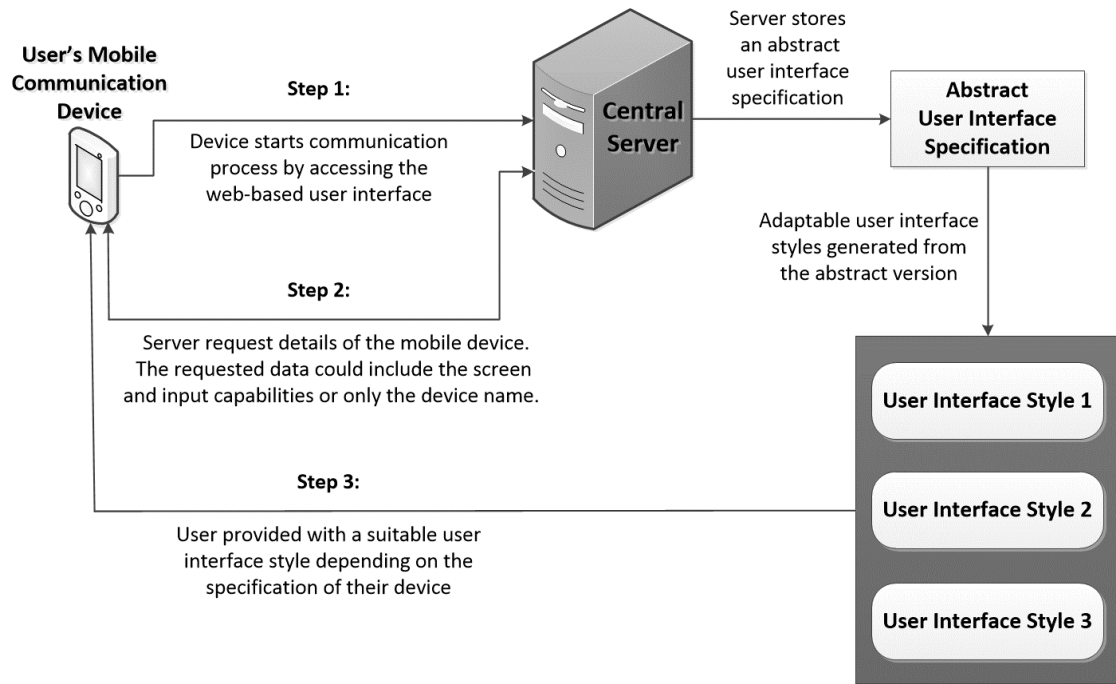
able to take advantage of a more user-friendly approach than is currently available for the platform.

In utilising the web-based user interface it is possible for the platform to limit or remove the need for data entry on the user's side of the communication process. The current situation for parameter positions in *Incoming* templates, where the user chooses one item from a finite list of items, requires the user to manually delete the unwanted items. The template mechanism could include tailored response functionality, whereby for parameter positions in an *Incoming* template a user would only need to select the desired item from the available options. This feature could also be used for other data entry points, such as selecting a date. Employing tailored response functionality should result in increased accuracy of field data that is transmitted from the user.

There is a vast array of mobile communication devices currently available in the market. These devices differ in screen size, input capabilities and the installed operating system [202]. The differing characteristics of these mobile devices have resulted in usability challenges [203] where one static user interface may not be appropriate to ensure a user-friendly interaction process for all types of devices [204, 205]. This issue could be solved by creating an adaptable user interface, whereby aspects of the user interface are altered to meet the needs of the user [206]. The information on display and the way it is presented could be adapted depending on various characteristics of the user's device. For example, a large touchscreen mobile phone with a high resolution display could present a larger amount of information than an old style keypad device. The server could request details of a user's device, each time a new interaction occurs, obtaining data such as the screen size and resolution [204]. Alternatively, the *Principal* database could store details on a list of devices, so only the device name would be requested from the user's device. Based on this data, the user would be presented with an appropriate layout of the information that is suited for interaction and navigation on their device.

An adaptable user interface could also be utilised to examine each device's main input scheme whether it is via a touchscreen, a *QWERTY* keyboard or an old style keypad [204]. The interaction process, in which the user inputs data, could then adapt depending on these capabilities. For example, the tailored response templates that

were previously mentioned would be more suited to a touchscreen interface, whereas the platform's original method of data input could be used for devices with *QWERTY* keyboards. Figure 8.2 illustrates how the server would generate adaptable user interfaces for each type of mobile communication device. Implementing an adaptable user interface would enable the platform to cater for the various types of mobile communication devices and consequently improve the user interaction process.



**Figure 8.2:** *Implementing an adaptable user interface into the platform*

### 8.2.1.3 New methods of communication

Additional communication methods would need to be investigated to include in the platform in order to offer further features for data input and increase the flexibility in the way users can interact with the server. The platform currently only supports the sending and receiving of multimedia files via email. This imposes restrictions on the way that users can interact with the server. If a user's desired communication method is SMS then that user would have to switch back and forth between the alternative methods of communication. Chapter 2 investigated the Multimedia Messaging Service (MMS) as a means to transmit multimedia files over the mobile networks [5]. However, the MMS method has not been implemented into the platform. MMS is not as ubiquitous and popular as the communication methods employed, such as SMS

and email. There has also been a difficulty in finding a web service to facilitate sending and receiving MMS messages at a low cost.

New methods of communication have recently been developed for smartphones to provide a zero cost solution to transmitting messages to other supported devices. One example is the *WhatsApp Messenger* application [207], which is available for smartphones running on popular mobile operating systems, such as *Apple's iOS and Google's Android platforms* [208]. *WhatsApp Messenger* uses the mobile internet to transmit messages from one device to another, where both devices are required to install the application in order to communicate via this method. *WhatsApp* messages can contain text, images and other file types free of charge to either the sender or recipient as long as they have an inclusive internet data allowance [209]. Therefore, the platform could make use of *WhatsApp Messenger* to offer a further means of communication with its users, which would also support the sending and receiving of multimedia files.

Furthermore, *WhatsApp Messenger* includes a feature that allows users to create groups, in which each member can participate in an active conversation and view the sent messages of the other members of the group [208]. The platform could utilise this feature to coordinate a group of users that are participating in an instance. For example, multiple users could cooperate to solve an assignment, with each member of a *WhatsApp* group alerting the other members of the group of their actions and findings. This would enhance the coordination between users by enabling them to communicate with each other, which should lead to a more effective use of the collective intelligence of the user group. This would, in all probability, result in superior solutions being obtained for issues that arise in the scenarios [127].

## **8.2.2 Ideas for future areas of research**

### **8.2.2.1 Scale-up study**

The platform is able to handle a multitude of scenario instances with a small user group. However, the platform encounters performance issues when supporting over 100 users. These issues include long response times when processing multiple messages concurrently and slow speeds on the web-based user interface due to data-tables being used and locked by the TAU. As the platform is designed to run

instances of multiple scenarios concurrently, it would be necessary to manage a large user group with minimal performance degradation.

The current procedure for handling received messages involves processing each message in turn, whereby messages are placed in a queue until they can be processed. If a large quantity of messages are received in a short period of time this could result in a backlog. As a result, the platform may not be meeting its objective of providing updates in real-time if there is a large batch of messages to process. This performance issue could be resolved by improved by incorporating parallel processing functionality into the application code. The platform has been developed based on object-oriented design principles. Each time a class in the TAU is called upon for data analysis procedures it is an object of the class that processes a user's message. Therefore, it would be possible to have multiple objects running concurrently, with each object processing a different received message.

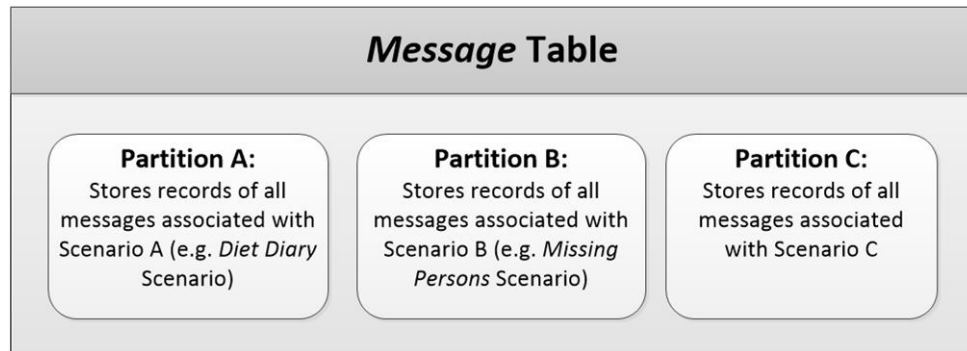
Implementing parallel processing techniques would enable the platform to process multiple messages concurrently, which would reduce the possibility of a backlog, thereby providing faster updates to users. For example, enabling six processes to run concurrently should reduce the waiting time for a group of users to approximately a sixth of the original time. Multiple messages from the same user would also need to be handled orderly, according to the date received, to prevent conflicts in the resulting analysis. A single handler class could be responsible for distributing the input messages to multiple objects that are concurrently running on separate processes. The handler class would manage the communication between each process and the data layer when database access is required to ensure there are no conflicts in analysis between related messages that need to be handled orderly.

The performance of the platform could also be improved by upgrading the hardware capabilities of the server. The server utilised in the experiments of Chapters 6 and 7 had a 3.2 gigahertz (GHz) dual-core central processing unit (CPU) and a 4 gigabyte (GB) unit of random-access memory (RAM). Upgrading the server to a faster six-core CPU means it could run all the processes without slowing down, whereby each thread would be run on a separate core of the CPU. Upgrading the RAM would also be required to ensure there is an adequate amount of memory to hold the data attributes for each object in use.

When storing a large amount of data it is necessary to ensure the databases run optimally. The databases need to be able to handle queries on large tables with swift results in order for the platform to effectively analyse and respond to inbound field data. The current setup of the database has been shown to reduce the performance of the web-based user interface when multiple users are concurrently interacting with the web-forms or the TAU is processing messages at the same time. The user would be delayed, until the required table within the database is free to access, before they can submit data or view previous messages.

A solution to manage a large database would be to divide the tables into smaller fragments in a process known as horizontal partitioning [210]. The rows of a table are segmented into multiple partitioned datasets, resulting in each partition still retaining all the columns but containing a smaller quantity of rows [211]. Partitioning is recommended for tables that are larger than 2 GB in size [212]. The rows are usually segmented based on certain criteria, for example databases that store historical data could have tables split based on the year or month a row was inserted [213]. This process has been shown to reduce query execution time by vast amounts as only the required partitions are examined [212]. R. Schumacher [210] discusses the query execution times regarding a database that has 8 million rows for data spread over 10 years. A query is performed on the database that only requires to read records in one particular year. The query took 38 seconds to execute on a non-partitioned table. However, this time was reduced to 4 seconds when the query was run on a partitioned table where each partition stored a year's worth of records.

Horizontal partitioning could be applied to tables in the *Principal* database in order to increase the speed of database queries. The *Message* and *DataIn* tables would be expected to grow at a faster rate than the other tables due to row insertions being performed at each occurrence of a new message. Partitioning these tables, based on the *Scenario* identifier, would enable faster data analysis queries to be executed. Figure 8.3 illustrates the partitioned *Message* table. Additionally, the other tables could be monitored as new scenarios are implemented and partitioned based on this criteria when they reach the 2 GB size limit.



**Figure 8.3:** Horizontal partitioning of the Message table with each partition holding records for a single scenario (adapted from [213])

Partitioning also provides benefits to the parallel processing operations. Currently, if one of the *Principal* database's tables is being queried then that table would be locked out until the query has been completed. In a partitioned database the partitions not required by the executed query would still be available [212]. Therefore, it would be possible to perform parallel operations on a single partitioned table, preventing a slowdown in the processes. This would boost the performance of the web-based user interface when multiple users are submitting web requests.

The implementation of these features would improve the scalability of the platform to support both the integration of future scenarios and an ever increasing number of users, whilst still providing responses in real-time.

#### 8.2.2.2 Setting up an experimental scheme

A number of the applications that were discussed in Chapter 3 have benefited from the feedback of experiments run on user groups. For example, an experiment was undertaken for the *Pharmaceutical Care* application [12] to determine if patients' interaction with the application increased their usage of prescribed medication. The experiments performed thus far on the platform have only been on a limited number of users in a closed environment. In Chapters 6 and 7 each scenario was tested with 1-3 users to demonstrate their use of the platform's functionality with the scalability test extending this to 20 users in order to assess the performance of the platform. Both these experiment types have only been carried out within a narrow scope to evaluate distinct aspects of the platform.



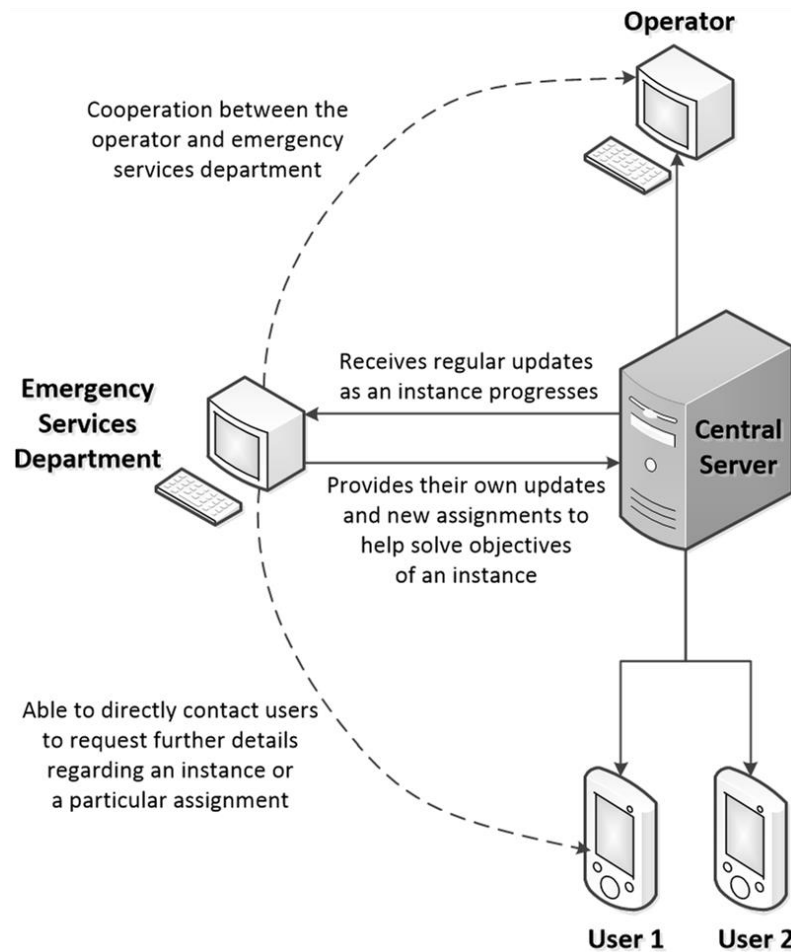
An experimental scheme could be applied to the platform, whereby individual experiments are undertaken to assess the running of each scenario in real-world use. In the case of the *Diet Diary* scenario a group of users could be provided with individual *Diet Diary* instances, where these instances are active for an extensive period of time. The results of the user data over this time period could be examined to determine the effectiveness of the various components of the platform, the acceptability of the concept and the usability of the implementation. This would include observing the TAU's functionality in analysing data and the template mechanism's capability to provide a communication structure for the scenario. The web-based user interface is currently only accessible via the local area network of the platform's central server. The real-world experiment would require the web-based user interface to be published to a website available over the internet in order to be accessible by users from any location. The website could then be tested to assess its performance and operation with users in an active mobile environment.

An experiment could also be run using active instances of the *Missing Persons* scenario. Initially, the experiment would use a test subject simulating a missing person. This experiment would be of importance in testing the platform's ability to coordinate a large user group over an extended period of time. Additionally, experiments using both scenarios concurrently need to be carried out to assess the platform's performance in coping with high user demand after the scale-up upgrades have been implemented.

The users would be required to provide feedback at the end of the experiments. The feedback gathered would be of value in identifying issues that were encountered by the users when they interacted with the server during the experiment. This would include the users' opinions on the accessibility and ease of the interaction process via each communication method, comments on aspects of the experiment where the platform did not perform as expected and suggestions for new features to be implemented. The observation data and feedback could then be utilised to further improve the platform, with both altering the platform's hardware capabilities and increasing the scope of its functionality.

### 8.2.2.3 *Facilitating emergency services*

A new direction for the platform would be to expand the communication infrastructure to contact departments in the emergency services. The *Missing Persons* scenario has highlighted the need to implement this feature, where currently any communication with the emergency services is performed manually by the operator. The *Principal* database could be extended to support this new feature with a new table to store records of emergency service departments and links to associate each scenario with the relevant department. The TAU's action list would be increased to generate and send news reports to the associated department whenever a significant event occurs in an active instance. Therefore, the respective department would be kept informed in real-time with the details of user reports and the results of assignments. For example, in a *Missing Person* instance each occurrence of a suspicious person reported by a user would generate a news report to the police. Figure 8.4 illustrates the role of an emergency services department in the client-server architecture of the platform.



**Figure 8.4:** *The platform's client-server architecture incorporating emergency services during active instances of a scenario*

This feature, of the emergency services being kept informed in real-time, could be used alongside GPS technology to support new scenarios in areas such as disaster management. D. Ashbrook and T. Starner [81] discuss a system that processes the past locations of users throughout the day in order to generate location models of their current and probable future locations. The platform could utilise this technique to coordinate and mobilise groups of users for enhanced cooperation in disaster management scenarios. The platform would need to have a two-way communication structure with emergency service departments. The police could then work in conjunction with the platform to generate assignment data for a disaster management instance. The platform would utilise this data to distribute the assignments amongst individual users, based on their location models. This concept illustrates a new course for the platform, accentuating its future potential to solve issues that continuously arise in a mobile environment.

## References

- [1] S. Hamm, *The Race for Perfect: Inside the Quest to Design the Ultimate Portable Computer*, New York, NY: McGraw-Hill, 2008.
- [2] R. Ling, *The Mobile Connection: The Cell Phone's Impact on Society*, 3<sup>rd</sup> ed., San Francisco, CA: Morgan Kaufmann, 2004.
- [3] International Telecommunication Union (2011). "*The World in 2011: ICT Facts and Figures*," ITU [Online]. Available: <http://www.itu.int/ITU-D/ict/facts/2011/material/ICTFactsFigures2011.pdf> [Accessed: Jul. 23, 2013].
- [4] J. Hanson, *24/7: How Cell Phones and the Internet Change the Way We Live, Work and Play*, Westport, CT: Praeger, 2007.
- [5] R. Dettmer, "Short message gets longer [GSM]," *IEE Review*, vol. 43(3), p.104, 1997.
- [6] G. L. Bodic, *Mobile Messaging Technologies and Services: SMS, EMS and MMS*, 2<sup>nd</sup> ed., Chichester: Wiley-Blackwell, 2005.
- [7] N. S. Baron, *Always On: Language in an Online and Mobile World*, New York, NY: Oxford University Press, 2008.
- [8] Y. How and M.Y. Kan, "Optimizing predictive text entry for short message service on mobile phones," in *Proceedings of the Human Computer Interfaces International*, Las Vegas, NV, USA, 2005.
- [9] BlackBerry. "*BlackBerry Bold 9790*," [uk.blackberry.com](http://uk.blackberry.com) [Online]. Available: <http://uk.blackberry.com/smartphones/blackberry-bold-9790.html> [Accessed: Jul. 23, 2013].
- [10] D. Okabe, "Emergent Social Practices, Situations and Relations through Everyday Camera Phone Use," in *Proceedings of the International Conference on Mobile Communication and Social Change*, Seoul, Korea, 2004, pp.1-19.
- [11] A. Smith, *Text Messaging as a Breaking News Information Source and University Journal Accent Section Portfolio*, PhD dissertation, Communication Dept., Southern Utah University, Cedar City, UT, USA, 2009.
- [12] Y. Mao et al., "Mobile phone text messaging for pharmaceutical care in a hospital in China," *Journal of Telemedicine and Telecare*, vol. 14(8), pp410-414, 2008.

- [13] L. Zhu et al., "UbiquitousSurvey: a framework supporting mobile field survey data collection and analysis," in *Proceedings of the 43<sup>rd</sup> Annual Southeast Regional Conference*, Kennesaw, GA, USA, 2005, pp.70-74.
- [14] Oracle. "Anatomy of the Client/Server Model", docs.oracle.com [Online]. Available: [http://docs.oracle.com/cd/E13203\\_01/tuxedo/tux80/atmi/intbas3.htm](http://docs.oracle.com/cd/E13203_01/tuxedo/tux80/atmi/intbas3.htm) [Accessed: Jul. 23, 2013].
- [15] M. J. Callaghan et al., "Client-Server Architecture for Remote Experimentation for Embedded Systems," *International Journal of online Engineering*, vol. 2(4), pp.1-6, 2006.
- [16] X. Faulkner and F. Culwin, "When fingers do the talking: a study of text messaging," *Interacting with Computers*, vol. 17(2), pp.167-185, 2005.
- [17] The Scottish Government (2005). "Technical Evaluation of Digital Interactive Television Pilot," scotland.gov.uk [Online]. Available: <http://www.scotland.gov.uk/Publications/2006/01/12104731/7> [Accessed: Jul. 23, 2013] (Image Retrieved).
- [18] International Telecommunication Union (2010). "The World in 2010: ICT Facts and Figures," ITU [Online]. Available: <http://www.itu.int/ITU-D/ict/material/FactsFigures2010.pdf> [Accessed: Jul. 23, 2013] (Image Retrieved).
- [19] McTel, "First Delivery Attempt (Direct Messaging)," mctel.net [Online]. Available: <http://www.mctel.net/art.php/en/ar42/first-delivery-attempt.html> [Accessed: Jul. 23, 2013].
- [20] K. Holley, "The GSM short message service," in *IEE Colloquium on GSM and PCN Enhanced Mobile Services*, London, England, 1991.
- [21] L. Leung, "Unwillingness-to-communicate and college students' motives in SMS mobile messaging," *Telematics and Informatics*, vol. 24(2), pp.115-129, 2007.
- [22] M. Divitini et al., "Improving communication through mobile technologies: which possibilities?" in *Proceedings of the IEEE International Workshop on Wireless and Mobile Technologies in Education*, Vaxjo, Sweden, 2002, pp.86-90.
- [23] C. Thurlow and A. Brown, "Generation Txt? The sociolinguistics of young people's text-messaging," *Discourse Analysis Online*, vol. 1(1), 2003. [Online].

Available:

[http://faculty.washington.edu/thurlow/research/papers/Thurlow&Brown\(2003\).htm](http://faculty.washington.edu/thurlow/research/papers/Thurlow&Brown(2003).htm) [Accessed: Jul. 23, 2013].

- [24] M. Rockwell (2007), “*Cold, Hard Facts Straight From the Cellphone*,” Broadcasting & Cable [Online]. Available: [http://www.broadcastingcable.com/article/107837-Cold\\_Hard\\_Facts\\_Straight\\_From\\_the\\_Cellphone.php](http://www.broadcastingcable.com/article/107837-Cold_Hard_Facts_Straight_From_the_Cellphone.php) [Accessed: Jul. 23, 2013].
- [25] Delaware Online, “*Delaware Online Text Alerts*,” delawareonline.com [Online]. Available: <http://www.delawareonline.com/section/MOBILE/DelawareOnline-Text-Alerts> [Accessed: Apr. 05, 2012] (Image Retrieved).
- [26] M. Ghaderi and S. Keshav, “Multimedia messaging service: system description and performance analysis,” in *Proceedings of the First International Conference on Wireless Internet*, 2005, pp. 198-205.
- [27] G. L. Bodic, *Multimedia Messaging Service: An Engineering Approach to MMS*, Chichester: Wiley-Blackwell, 2003.
- [28] J. Moore and J. Blecher, *How to Do Everything with Your Camera Phone*, Emeryville, CA: McGraw-Hill, 2004.
- [29] J. Chen (2009), “*AT&T’s iPhone MM Carrier Update*,” Gizmodo [Online]. Available: <http://gizmodo.com/5367894/atts-iphone-mms-carrier-update-is-live> [Accessed: Jul. 23, 2013] (Image Retrieved).
- [30] J. Delaney, “MMS five years on,” *Journal of Telecommunications Management*, vol. 1(1), pp.69-78, 2008.
- [31] T. Kindberg et al., “The ubiquitous camera: an in-depth study of camera phone use,” *Pervasive Computing*, vol. 4(2), pp.42-50, 2005.
- [32] J. Freeman, *The Tyranny of E-mail: The Four-Thousand-Year Journey to Your Inbox*, New York, NY: Scribner, 2009.
- [33] C. Partridge, “The technical development of internet email,” *Annals of the History of Computing*, vol. 30(2), pp.3-29, 2008.
- [34] M. A. Mazmanian et al., “CrackBerries: the social implications of ubiquitous wireless e-mail devices,” in *Designing Ubiquitous Information Environments: Socio-Technical Issues and Challenges*, C. Sorenson et al., Springer, 2005, pp.337-343.

- [35] Addictive Tips (2009), “*Google Apps Connector for BlackBerry*,” addictivetips.com [Online]. Available: <http://www.addictivetips.com/internet-tips/google-apps-connector-for-blackberry-available-now> [Accessed: Jul. 23, 2013] (Image Retrieved).
- [36] J. Pablo (2007), “*SMS vs Email – The Moot Point*,” Ezine Articles [Online]. Available: <http://ezinearticles.com/?SMS-Vs-Email---The-Moot-Point&id=632879> [Accessed: Jul. 23, 2013].
- [37] A. R. Hickey (2006), “*SMS vs. mobile email: Which is the ‘killer app’?*” Tech Target [Online]. Available: <http://searchmobilecomputing.techtarget.com/news/1193310/SMS-vs-mobile-email-Which-is-the-killer-app> [Accessed: Jul. 23, 2013].
- [38] C. A. Middleton and W. Cukier, “Is mobile email functional or dysfunctional? Two perspectives on mobile email usage,” *European Journal of Information Systems*, vol. 15(3), pp.252-260, 2006.
- [39] I. Faletski (2007), “*SMS vs E-mail: Competitors or Co-workers?*” Mobile Muse [Online]. Available: <http://www.mobilemuse.ca/news/igor-faletski/sms-vs-e-mail-competitors-or-co-workers> [Accessed: Jul. 23, 2013].
- [40] M. Mehra (2008), “*Mobile Email Versus SMS*,” Ezine Articles [Online]. Available: <http://ezinearticles.com/?Mobile-Email-Versus-SMS&id=1116476> [Accessed: Jul. 23, 2013].
- [41] A. Tailor (2009), “*Mobile Email vs. SMS-based Mobile Marketing*,” Mobile Financial [Online]. Available: <http://mobile-financial.com/blogs/mobile-email-vs-sms-based-mobile-marketing> [Accessed: Jul. 23, 2013].
- [42] Text Local [Online] Available: <http://www.textlocal.com> [Accessed: Jul. 23, 2013].
- [43] Bulk SMS [Online] Available: <http://www.bulksms.com> [Accessed: Jul. 23, 2013].
- [44] S. L. Jarvenpaa and K. R. Lang, “Managing the paradoxes of mobile technology,” *Information Systems Management*, vol. 22(4), pp.7-23, 2005.
- [45] T. Cochrane and R. Bateman, “Smartphones give you wings: Pedagogical affordances of mobile Web 2.0,” *Australian Journal of Educational Technology*, vol. 26(1), pp.1-14, 2010.

- [46] A. Pocovnicu, "Biometric security for cell phones," *Informatica Economica*, vol. 13(1), pp.57-63, 2009.
- [47] Digizmo, "How Mobile Phones Influences Mobile Internet Growth," digizmo.com [Online]. Available: <http://digizmo.com/2010/07/14/how-mobile-phones-influence-mobile-internet-growth> [Accessed: Jul. 23, 2013] (Image Retrieved).
- [48] M. Mouly and M. B. Pautet, *The GSM System for Mobile Communications*, Telecom Publishing, 1992.
- [49] W. Shudong and M. Higgins, "Limitations of mobile phone learning," in *Proceedings of the IEEE International Workshop on Wireless and Mobile Technologies in Education*, Washington, DC, USA, 2005, pp.179-181.
- [50] M. N. Boulos et al., "How smartphones are changing the face of mobile and participatory healthcare: an overview, with example from eCAALYX," *Biomedical engineering online*, vol. 10(24), 2011. [Online]. Available: <http://www.biomedical-engineering-online.com/content/10/1/24> [Accessed: Jul. 23, 2013].
- [51] E. Becker, "Using smartphones and Facebook in a major assessment: the student experience," *E-journal of Business Education & Scholarship of Teaching*, vol. 4(1), pp.19-31, 2010.
- [52] M. Kenney and B. Pon, "Structuring the smartphone industry: Is the mobile internet OS platform the key?" *Journal of Industry, Competition and Trade*, vol. 11(3), pp.239-261, 2011.
- [53] H. Verkasalo, "Analysis of smartphone behaviour," in *Proceedings of the 9th International Conference on Mobile Business and the 9th Global Mobility Roundtable*, Athens, Greece, 2010, pp. 258-263.
- [54] Apple iPhone [Online]. Available: <http://www.electricpig.co.uk/wp-content/uploads/2009/03/iphone-sms.jpg> [Accessed: Apr. 05, 2012] (Image Retrieved).
- [55] Sony Ericson k750i [Online]. Available: <http://andacellular.com/?wpsc-product=sony-ericson-k750i> [Accessed: Apr. 05, 2012] (Image Retrieved).
- [56] J. West and M. Mace, "Browsing as the killer app: Explaining the rapid success of Apple's iPhone," *Telecommunications Policy*, vol. 34(5), pp.270-286, 2010.



- [57] J. West et al., "Value creation in the mobile internet: The impact of Apple's iPhone," 2008.
- [58] P. Buckley, *The Rough Guide to the iPhone*, 3<sup>rd</sup> ed., London: Rough Guides, 2010.
- [59] A. Charlesworth, "The ascent of smartphone," *Engineering & Technology*, vol. 4(3), pp.32-33, 2009.
- [60] S. Allen et al., "BlackBerry HTML UI," in *Pro Smartphone Cross-Platform Development*, S. Allen et al., Apress, 2010, pp.235-245.
- [61] BlackBerry smartphone [Online]. Available: <http://www.bccthis.com/blackberry.php> [Accessed: Apr. 05, 2012] (Image Retrieved).
- [62] W. Webb, "Being mobile: smartphone revolution," *Engineering & Technology*, vol. 5(15), pp.64-65, 2010.
- [63] Smartphone internet access [Online]. Available: <http://gallery.techarena.in/showphoto.php/photo/11507> [Accessed: Jul. 23, 2013] (Image Retrieved).
- [64] R. K. Miller et al., "Rethinking reference and instruction with tablets," *Library Technology Reports*, vol. 48(8), pp.4-9, 2012.
- [65] Apple iPad [Online]. Available: <http://www.apple.com/ipad/specs> [Accessed: Jul. 23, 2013].
- [66] N. Singai and N. Rajan (2012), "*Tablets Vs Phones Vs Ultrabooks: Pros & Cons*," Business Today [Online]. Available: <http://businesstoday.intoday.in/story/pros-and-cons-of-buying-a-tablet-smartphone-or-ultrabook/1/188057.html> [Accessed: Jul. 23, 2013].
- [67] A. Mitchell et al., "The tablet revolution and what it means for the future of news," *Pew Research Center*, 2011.
- [68] K. Pratt, "Netbook, eReader, or iPad? – that is the question," *Computers in New Zealand Schools*, vol. 22(2), 2010.
- [69] N. Eichenlaub et al., "Project iPad: Investigating tablet integration in learning and libraries at Ryerson University," *Librarian and Staff Publications: Ryerson University*, 2011.
- [70] J. Ballew, *How to Do Everything iPad 2*, McGraw-Hill, 2011.

- [71] S. Kaur, "The revolution of tablet computers and apps: A look at emerging trends," *Consumer Electronics Magazine*, vol. 2(1), pp.36-41, 2013.
- [72] R. Budiu and J. Nielsen, *iPad App and Website Usability*, 2<sup>nd</sup> ed., Fremont, CA: Nielsen Norman Group, 2011.
- [73] A. I. Shaik, "5 of the Biggest Improvements in Mobile Camera Technology," Gear Burn [Online]. Available: <http://gearburn.com/2013/06/5-of-the-best-improvements-in-mobile-camera-technology> [Accessed: Jul. 23, 2013].
- [74] Mobile phone image capture [Online]. Available: <http://areacellphone.com/2009/05/how-to-take-good-quality-pictures-with-low-quality-cell-phone-camera> [Accessed: Jul. 23, 2013] (Image Retrieved).
- [75] J. D. Murray and W. vanRyper, *Encyclopedia of Graphics File Formats*, 2<sup>nd</sup> ed., O'Reilly Media, 1996.
- [76] G. K. Wallace, "The JPEG still picture compression standard," *Communications of the ACM*, vol. 34(4), pp.30-44, 1991.
- [77] M. Hazas et al., "Location-aware computing comes of age," *Computer (Invisible Computing)*, vol. 37(2), pp.95-97, 2004.
- [78] Y. Zhao, "Mobile phone location determination and its impact on intelligent transportation systems," *Intelligent Transportation Systems*, vol. 1(1), pp.55-64, 2000.
- [79] N. D. Lane et al., "A survey of mobile phone sensing," *IEEE Communications Magazine*, vol. 48(9), pp.140-150, 2010.
- [80] J. McNamara, *GPS For Dummies*, 2<sup>nd</sup> ed., Indianapolis, IN: Wiley, 2008.
- [81] D. Ashbrook and T. Starner, "Using GPS to learn significant locations and predict movement across multiple users," *Personal and Ubiquitous Computing*, vol. 7(5), pp.275-286, 2003.
- [82] R. Bajaj et al., "GPS: Location-tracking technology," *Computer*, vol. 35(4), pp.92-94, 2002 (Image Retrieved).
- [83] Mobile phone navigation software [Online]. Available: <http://www.kokeytechnology.com/gadgets/cell-phonesmobile-phones/t-mobile-garminfone-android-phone-specs-price-release-date-in-the-us> [Accessed: Jul. 23, 2013] (Image Retrieved).

- [84] I. A. Junglas and R. T. Watson, "Location-based services: Evaluating user perceptions of location-tracking and location-awareness services," *Communications of the ACM*, vol. 51(3), pp.65-69, 2008.
- [85] AroundMe smartphone application [Online]. Available: <http://www.aroundmeapp.com> [Accessed: Jul. 23, 2013].
- [86] V. Astarita and M. Florian, "The use of mobile phones in traffic management and control," in *Proceedings of the IEEE Intelligent Transportation Systems*, Oakland, CA, USA, 2001, pp.10-15.
- [87] AroundMe smartphone application [Online]. Available: <http://skytechgeek.com/2012/02/top-10-iphone-apps-you-shouldnt-be-without> [Accessed: Jul. 23, 2013] (Image Retrieved).
- [88] ITV, "*The X-Factor Text Voting FAQs*", itv.com [Online]. Available: <http://www.itv.com/termsandconditions/thexfactortextvotingfaqs/default.html> [Accessed: Apr. 05, 2012].
- [89] The Wright Stuff [Online]. Available: <http://www.channel5.com/shows/the-wright-stuff> [Accessed: Jul. 23, 2013].
- [90] BBC, "*Send us your pictures*," news.bbc.co.uk [Online]. Available: [http://news.bbc.co.uk/mobile/bbc\\_news/weekinpictures/yourpics/index.shtml?context=cps\\_ukfs](http://news.bbc.co.uk/mobile/bbc_news/weekinpictures/yourpics/index.shtml?context=cps_ukfs) [Accessed: Jul. 23, 2013].
- [91] A. Fernando, "If you text it, they may come: talkers give way to texters as technology turns lowly cell phones into multidimensional communication tools," *Communication World*, 2007.
- [92] K. C. Leong et al., "The use of text messaging to improve attendance in primary care: A randomized controlled trial," *Family Practice*, vol. 23(6), pp.699-705, 2006.
- [93] S. R. Downer et al., "Use of SMS text messaging to improve outpatient attendance," *Medical Journal of Australia*, vol. 183(7), pp.366-368, 2005.
- [94] S. R. Downer et al., "SMS text messaging improves outpatient attendance," *Australian Health Review*, vol. 30(3), pp.389-396, 2006.
- [95] C. Markett et al., "Using short message service to encourage interactivity in the classroom," *Computers & Education*, vol. 46(3), pp.280-293, 2006.
- [96] C. Markett et al., " 'PLS Turn UR Mobile on': Short message service (SMS) supporting interactivity in the classroom," in *Proceeding of the International*

- Conference on Cognition and Exploratory Learning in the Digital Age*, Lisbon, Portugal, 2004, pp.491-494.
- [97] A. Rodgers et al., "Do u smoke after txt? Results of a randomised trial of smoking cessation using mobile phone text messaging," *Tobacco Control*, vol. 14(4), pp.255-261, 2005.
  - [98] N. Cavus and D. Ibrahim, "m-Learning: An experiment in using SMS to support learning new English language words," *British Journal of Educational Technology*, vol. 40(1), pp.78-91, 2009.
  - [99] A. Farmer et al., "A real-time, mobile phone-based telemedicine system to support young adults with type 1 diabetes," *Informatics in Primary Care*, vol. 13(3), pp.171-178, 2005.
  - [100] B. Banks et al., "A low-cost wireless system for autonomous generation of road safety alerts," in *Proceedings of the 16<sup>th</sup> International Symposium on Smart Structures and Materials & Nondestructive Evaluation and Health Monitoring*, San Diego, CA, USA, 2009.
  - [101] S. M. Huff et al., "HELP the next generation: a new client-server architecture," in *Proceedings of the Annual Symposium on Computer Application in Medical Care*, Washington, DC, USA, 1994, pp.271-275.
  - [102] Perfect Diet Tracker [Online]. Available: <http://www.perfect-diet-tracker.com> [Accessed: Jul. 23, 2013].
  - [103] A. Hewitt and A. Forte, "Crossing boundaries: Identity management and student/faculty relationships on the Facebook", Poster presented at *Computer Supported Cooperative Work*, Alberta, Canada, 2006.
  - [104] A. Acquisti and R. Gross, "Imagined communities: Awareness, information sharing, and privacy on the Facebook," in *Privacy Enhancing Technologies: 6<sup>th</sup> International Workshop*, Cambridge, England, 2006, pp.36-58.
  - [105] A. N. Joinson, "Looking at, looking up or keeping up with people?: Motives and use of Facebook," in *Proceedings of the SIGCHI conference on Human Factors in Computer Systems*, Florence, Italy, 2008, pp. 1027-1036.
  - [106] C. Abram, *Facebook for Dummies*, 4<sup>th</sup> ed., Hoboken, NJ: Wiley, 2012.
  - [107] N. B. Ellison et al., "The benefits of Facebook "Friends:" Social capital and college students' use of online social network sites," *Journal of Computer-Mediated Communication*, vol. 12(4), pp.1143-1168, 2007.

- [108] Facebook [Online]. Available: <https://www.facebook.com> [Accessed: Jul. 23, 2013].
- [109] M. Gjoka et al., "Walking in Facebook: A case study of unbiased sampling of OSNs," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, San Diego, CA, USA, 2010, pp.1-9.
- [110] S. T. Tong et al., "Too much of a good thing? The relationship between number of friends and interpersonal impressions on Facebook," *Journal of Computer-Mediated Communication*, vol. 13(3), pp.531-549, 2008.
- [111] B. E. Smith, *Sams Teach Yourself Facebook for Business in 10 Minutes*, Sams Publishing, 2011.
- [112] Facebook Check-In Deal [Online]. Available: [http://allfacebook.com/facebook-small-businesses\\_b64117](http://allfacebook.com/facebook-small-businesses_b64117) [Accessed: Jul. 23, 2013] (Image Retrieved).
- [113] N. Park et al., "Being immersed in social networking environment: Facebook groups, uses and gratifications, and social outcomes," *CyberPsychology & Behavior*, vol. 12(6), pp.729-733, 2009.
- [114] C. M. Paris et al., "The role of social media in promoting special events: Acceptance of Facebook 'Events'," in *Proceedings of the International Conference on Information and Communication Technologies in Tourism*, Lugano, Switzerland, 2010, pp.531-541.
- [115] T. Judd, "Facebook versus email," *British Journal of Educational Technology*, vol. 41(5), pp.E101-E103, 2010.
- [116] B. J. Fogg and D. Iizawa, "Online persuasion in Facebook and Mixi: A cross-cultural comparison," in *Proceedings of the 3<sup>rd</sup> International Conference on Persuasive Technology*, Oulu, Finland, 2008, pp. 35-46.
- [117] J. B. Caruso and G. Salaway, "The ECAR Study of Undergraduate Students and Information Technology, 2008." *Educause Center for Applied Research*, 2008.
- [118] Twitter [Online]. Available: <https://twitter.com> [Accessed: Jul. 23, 2013].
- [119] H. Kwak et al., "What is Twitter, a social network or a news media?" in *Proceedings of the 19<sup>th</sup> International Conference on World Wide Web*, Raleigh, NC, USA, 2010, pp.591-600.
- [120] A. Java et al., "Why we twitter: understanding microblogging usage and communities," in *Proceedings of the 9<sup>th</sup> WebKDD and 1<sup>st</sup> SNA-KDD 2007*

- Workshop on WebMining and Social Network Analysis*, San Jose, CA, USA, 2007, pp.56-65.
- [121] Instagram [Online]. Available: <http://instagram.com> [Accessed: Jul. 23, 2013].
- [122] C. Watkins, *Instagram: Why Does My Business Need It?*, Euless, TX: TokaySEO, 2012.
- [123] A. Macarthy, *How to Use Instagram For Business*, 2012.
- [124] G. Blattner and M. Fiori, "Facebook in the language classroom: Promises and possibilities," *International Journal of Instructional Technology and Distance Learning*, vol. 6(1), pp.17-28, 2009.
- [125] G. Grosseck and C. Holotescu, "Can we use Twitter for educational activities," presented at the 4<sup>th</sup> *International Scientific Conference on eLearning and Software for Education*, Bucharest, Romania, 2008.
- [126] F. D. L. Wigand, "Twitter in government: Building relationships one Tweet at a time," in *Proceedings of the 7<sup>th</sup> International Conference on Information Technology: New Generations*, Las Vegas, NV, USA, 2010, pp.563-567.
- [127] S. Georgi and R. Jung, "Collective intelligence model: How to describe collective intelligence," in *Advances in Intelligent and Soft Computing: Advances in Collective Intelligence 2011*, J. Altmann et al., Springer, 2012, pp.53-64.
- [128] Micro Focus, "*Developing Client/Server Applications*," support.micofocus.com [Online]. Available: <http://support.microfocus.com/documentation/books/nx30books/sgdevt.htm> [Accessed: Jul. 23, 2013].
- [129] D. Harkey, *Client/Server Survival Guide*, 3<sup>rd</sup> ed., Wiley, 1999.
- [130] H. Schuster et al., "A client/server architecture for distributed workflow management systems," in *Proceedings of the 3<sup>rd</sup> International Conference on Parallel and Distributed Systems*, Hsinchu, Taiwan, 1994, pp.253-256.
- [131] Oracle. "*Oracle Database Concepts: Application and Networking Architecture*," docs.oracle.com [Online]. Available: [http://docs.oracle.com/cd/E11882\\_01/server.112/e16508/dist\\_pro.htm](http://docs.oracle.com/cd/E11882_01/server.112/e16508/dist_pro.htm) [Accessed: Jul. 23, 2013].

- [132] R. Winkelman, “*What is a Network Operating System?*” Florida Center for Instructional Technology [Online]. Available:  
<http://fcit.usf.edu/network/chap6/chap6.htm> [Accessed: Jul. 23, 2013].
- [133] UNM CIRT: Network Group (1997), “*Computing Architecture*,” University of New Mexico [Online]. Available:  
[http://www.unm.edu/~network/presentations/course/appendix/appendix\\_k/sld001.htm](http://www.unm.edu/~network/presentations/course/appendix/appendix_k/sld001.htm) [Accessed: Jul. 23, 2013].
- [134] Microsoft, “*Client/Server Architecture*,” [technet.microsoft.com](http://technet.microsoft.com) [Online]. Available: <http://technet.microsoft.com/en-us/library/cc917543.aspx> [Accessed: Jul. 23, 2013].
- [135] R. Sripriya, “*Pros and Cons of Client/Server Computing*,” Exforsys [Online]. Available: <http://www.exforsys.com/tutorials/programming-concepts/pros-and-cons-of-client-server-computing.html> [Accessed: Jul. 23, 2013].
- [136] P. Pankaj et al., “P2P business applications: Future and directions,” *Communications and Network*, vol. 4(3), pp.248-260, 2012.
- [137] J. Buford et al., *P2P Networking and Applications*, Burlington, MA: Morgan Kaufmann, 2009.
- [138] Q. H. Vu et al., *Peer-to-Peer Computing: Principles and Applications*, Heidelberg: Springer, 2010.
- [139] J. Dubey and V. Tokekar, “Identification of efficient peers in P2P computing system for real time applications,” preprint arXiv:1212.3074, Cornell University Library, 2012.
- [140] R. Steinmetz and K. Wehrle, *Peer-to-Peer Systems and Applications*, Heidelberg: Springer, 2005.
- [141] D. P. Anderson et al., “*The Science of SETI@home*,” SETI@home [Online]. Available: [http://setiathome.berkeley.edu/sah\\_about.php](http://setiathome.berkeley.edu/sah_about.php) [Accessed: Jul. 23, 2013].
- [142] G. J. Fakas and B. Karakostas, “A peer to peer (P2P) architecture for dynamic workflow management,” *Information and Software Technology*, vol. 46(6), pp.423-431, 2004.
- [143] D. S. Milojicic et al., “*Peer-to-peer computing*,” Technical Report HPL-2002-57R1, HP Laboratories, 2003.

- [144] Freenet: The Free Network [Online]. Available: <https://freenetproject.org> [Accessed: Jul. 23, 2013].
- [145] BitTorrent [Online]. Available: <http://www.bittorrent.com> [Accessed: Jul. 23, 2013].
- [146] Y. Liu and J. Pan, "The impact of NAT on BitTorrent-like P2P systems," in *Proceedings of the IEEE 9<sup>th</sup> International Conference on Peer-to-Peer Computing*, Seattle, WA, USA, 2009, pp.242-251.
- [147] L. Zhong et al., "Topological model and analysis of the P2P BitTorrent protocol," *International Journal of System Control and Information Processing*, vol. 1(1), pp.54-70, 2012.
- [148] A. Mislove et al., "Experiences in building and operating ePOST, a reliable peer-to-peer application," *ACM SIGOPS Operating Systems Review*, vol. 40(4), pp.147-159, 2006.
- [149] R. Olfati-Saber et al., "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95(1), pp.215-233, 2007.
- [150] M. Wooldridge, *An Introduction to MultiAgent Systems*, Chichester: Wiley, 2009.
- [151] L. Busoniu et al., "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 38(2), pp.156-172, 2008.
- [152] M. N. Huhns and L. M. Stephens, *Multiagent Systems and Societies of Agents*, MIT Press, 2008.
- [153] M. Pipattanasomporn et al., "Multi-agent systems in a distributed smart grid: Design and implementation," in *Proceedings IEEE Power Systems Conference and Exposition*, Seattle, WA, USA, 2009, pp.1-8.
- [154] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, 2000.
- [155] L. Padgham and M. Winikoff, *Developing Intelligent Agent Systems: A Practical Guide*, Chichester: Wiley, 2004.
- [156] E. Mangina et al., *Agent-Based Ubiquitous Computing*, Paris: Atlantis Press, 2009.



- [157] CMU Robotics Institute, “*Multi-Agent Systems*,” ri.cmu.edu [Online].  
Available: [http://www.ri.cmu.edu/research\\_guide/multi\\_agent\\_systems.html](http://www.ri.cmu.edu/research_guide/multi_agent_systems.html)  
[Accessed: Jul. 23, 2013].
- [158] F. Zambonelli et al., “Organisational abstractions for the analysis and design of multi-agent systems,” in *Agent-Oriented Software Engineering*, M. Wooldridge et al., Heidelberg: Springer, 2001, pp.235-251.
- [159] A. Moreno (2010), “*Introduction To Agents And Multi-Agent Systems Presentation*,” University Rovira I Virgili [Online]. Available:  
<http://www.slideshare.net/ToniMorenoURV/introduction-to-agents-and-multiagent-systems/#btnNext> [Accessed: Jul. 23, 2013].
- [160] M. P. Papazoglou and D. Georgakopoulos, “Service-oriented computing,” *Communications of the ACM*, vol. 46(10), pp.25-28, 2003.
- [161] M. A. Babar (2010), “*Service-Oriented Architecture*,” IT University of Copenhagen [Online]. Available: [https://blog.itu.dk/MSAR-E2010/files/2010/11/lect\\_w10\\_soa.pdf](https://blog.itu.dk/MSAR-E2010/files/2010/11/lect_w10_soa.pdf) [Accessed: Feb. 03, 2014].
- [162] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*, Boston, MA: Prentice Hall, 2005.
- [163] F. Kart et al., “A distributed e-healthcare system based on the service oriented architecture,” in *Proceedings of the IEEE International Conference on Services Computing*, Salt Lake City, UT, USA, 2007, pp.652-659.
- [164] K. Channabasavaiah et al., “Migrating to a service-oriented architecture,” *IBM DeveloperWorks*, vol. 16, pp.1-22 2004.
- [165] H. He (2003), “*What is Service-Oriented Architecture*,” O’Reilly webservices.xml.com [Online]. Available:  
<http://www.xml.com/pub/a/ws/2003/09/30/soa.html> [Accessed: Feb. 03, 2014].
- [166] D. Krafzig et al., *Enterprise SOA: Service-Oriented Architecture Best Practices*, Upper Saddle River, NJ: Prentice Hall, 2004.
- [167] B. Dong et al., “Web service-oriented manufacturing resource applications for networked product development,” *Advanced Engineering Informatics*, vol. 22(3), pp.282-295, 2008.
- [168] O. Zimmermann et al., “Service-oriented architecture and business process choreography in an order management scenario: Rationale, concepts, lessons learned,” in *Companion to the 20<sup>th</sup> Annual ACM SIGPLAN Conference on*

*Object-Oriented Programming, Systems, Languages, and Applications*, San Diego, CA, USA, 2005, pp.301-312.

- [169] Accenture, “*Service-Oriented Architecture: Business Benefits*,” [accenture.com](http://www.accenture.com/us-en/Pages/service-soa-business-benefits.aspx) [Online]. Available: <http://www.accenture.com/us-en/Pages/service-soa-business-benefits.aspx> [Accessed: Feb. 03, 2014].
- [170] N. Komoda, “Service oriented architecture (SOA) in industrial systems,” in *Proceedings of the IEEE International Conference on Industrial Informatics*, Singapore, 2006, pp.1-5.
- [171] Y. V. Natis (2003), “*Service-Oriented Architecture Scenario*,” Gartner Research, Stamford, CT [Online]. Available: <http://www.gartner.com/resources/114300/114358/114358.pdf> [Accessed: Feb. 03, 2014].
- [172] A. Brown et al., *Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications*, Santa Clara, CA: Rational Software Corporation, 2002.
- [173] EU-Orchestra (2007), “*Introduction to Service Oriented Architectures (SOA)*,” [eu-orchestra.org](http://www.eu-orchestra.org) [Online]. Available: <http://www.eu-orchestra.org/TUs/SOA/en/text/SOA.pdf> [Accessed: Feb. 03, 2014].
- [174] A. Paul (2011), “*Service Oriented Architecture (SOA) and its Advantages and Disadvantages*,” Techyv [Online]. Available: <http://www.techyv.com/article/service-oriented-architecture-soa> [Accessed: Feb. 03, 2014].
- [175] H. Haas and A. Brown (2004), “*Web Services Glossary*,” World Wide Web Consortium [Online]. Available: <http://www.w3.org/TR/ws-gloss> [Accessed: Feb. 03, 2014].
- [176] A. Dubey (2011), “*Service-Oriented Architecture*,” University of Wisconsin-Platteville [Online]. Available: [http://www.uwplatt.edu/csse/courses/prev/csse411-materials/s11/dubeya\\_Service%20Oriented%20Architecture.pptm](http://www.uwplatt.edu/csse/courses/prev/csse411-materials/s11/dubeya_Service%20Oriented%20Architecture.pptm) [Accessed: Feb. 03, 2014].
- [177] S. Gupta (2003), “*How to Use Novell Nsure UDDI Command Beans in Building Web Services-Enabled Applications*,” Novell [Online]. Available:

- <http://support.novell.com/techcenter/articles/dnd20030304.html> [Accessed: Feb. 03, 2014] (Image Retrieved).
- [178] J. Myerson (2005), “*Tame Sabarnes-Oxley Act with IBM Tivoli Storage Manager for Data Retention, Web Services and other Compliance Tools*,” IBM [Online]. Available: <http://www.ibm.com/developerworks/tivoli/library/t-tamesox> [Accessed: Feb. 03, 2014] (Image Retrieved).
- [179] The Open Group, “*Service Oriented Architecture: SOA Features and Benefits*,” opengroup.org [Online]. Available: [http://www.opengroup.org/soa/source-book/soa/soa\\_features.htm](http://www.opengroup.org/soa/source-book/soa/soa_features.htm) [Accessed: Feb. 03, 2014].
- [180] D. Hunt (2011), “*Limitations of SOA*,” IEEE University of Greenwich [Online]. Available: <http://ieeegreen.org/web/images/soa.pdf> [Accessed: Feb. 03, 2014].
- [181] P. V. Cravan (2011), “*The Dark Side of SOA*,” Simpson College, Indianola [Online]. Available: [http://cs.simpson.edu/files/DAMA\\_Presentation.pdf](http://cs.simpson.edu/files/DAMA_Presentation.pdf) [Accessed: Feb. 03, 2014].
- [182] Exforsys Inc. (2007), “*SOA Disadvantages*,” exforsys.com [Online]. Available: <http://www.exforsys.com/tutorials/soa/soa-disadvantages.html> [Accessed: Feb. 03, 2014].
- [183] Microsoft (2013), “*Classes (C# Programming Guide)*,” microsoft.com [Online]. Available: <http://msdn.microsoft.com/en-us/library/x9afc042.aspx> [Accessed: Feb. 03, 2014].
- [184] Microsoft SQL Server Management Studio [Online]. Available: <http://www.microsoft.com/en-gb/download/details.aspx?id=7593> [Accessed: Feb. 03, 2014].
- [185] Microsoft Visual Studio [Online]. Available: <http://www.visualstudio.com> [Accessed: Feb. 03, 2014].
- [186] ASP.NET [Online]. Available: <http://www.asp.net> [Accessed: Feb. 03, 2014].
- [187] Microsoft, “*Microsoft ActiveX Data Objects (ADO)*,” microsoft.com [Online]. Available: [http://msdn.microsoft.com/en-us/library/windows/desktop/ms675532\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms675532(v=vs.85).aspx) [Accessed: Feb. 03, 2014].
- [188] Database Dir, “*What is RDBMS?*” databasedir.com [Online]. Available: <http://www.databasedir.com/what-is-rdbms> [Accessed: Feb. 03, 2014].

- [189] K. Amrein et al., "Sclerostin and its association with physical activity, age, gender, body composition, and bone mineral content in healthy adults," *Journal of Clinical Endocrinology & Metabolism*, vol. 97(1), pp.148-154, 2012.
- [190] M. Jette et al., "Metabolic equivalents (METs) in exercise testing, exercise prescription, and evaluation of functional capacity," *Clinical Cardiology*, vol. 13(8), pp.555-565, 1990.
- [191] National Institute of Health, "Calculate Your Body Mass Index," nhlbi.nih.gov [Online]. Available: <http://www.nhlbi.nih.gov/guidelines/obesity/BMI/bmicalc.htm> [Accessed: Jul. 23, 2013].
- [192] Calories per Hour, "What it Takes to Lose a Pound," caloriesperhour.com [Online]. Available: [http://www.caloriesperhour.com/tutorial\\_pound.php](http://www.caloriesperhour.com/tutorial_pound.php) [Accessed: Jul. 23, 2013].
- [193] Food Counts, "Recommended Daily Allowances & Dietary Reference Intakes," foodcounts.com [Online]. Available: <http://www.foodcounts.com/recommended-daily-allowances/default.htm> [Accessed: Jul. 23, 2013].
- [194] FitDay, "My Fitness Log: Lifestyle Activity Level," fitday.com [Online]. Available: <http://www.fitday.com/fitness/Profile.html> [Accessed: Jul. 23, 2013].
- [195] N. Biehal et al., *Lost From View: Missing Persons in the UK*, Bristol: The Policy Press, 2003.
- [196] G. Newiss, "A study of the characteristics of outstanding missing persons: Implications for the development of police risk assessment," *Policing and Society*, vol. 15(2), pp.212-225, 2005.
- [197] G. Newiss, "Missing presumed...?: The police response to missing persons," *Police Research Series*, Home Office, London, vol. 114, 1999.
- [198] L. Diez, "The use of call grading: How calls to the police are graded and resourced," *Police Research Series*, Home Office, London, vol. 13, 1995.
- [199] C. Hedges, "Missing you already: A guide to the investigation of missing persons," Home Office, London, 2002.
- [200] National Policing Improvement Agency, "Guidance on the Management, Recording and Investigation of Missing Persons," 2<sup>nd</sup> ed., *Practice Improvements*, NPIA, London, 2010.

- [201] Google Maps [Online]. Available: <https://www.google.co.uk/maps> [Accessed: Jul. 23, 2013].
- [202] N. Mitrovic et al., "Adaptive interfaces in mobile environments - An approach based on mobile agents," in *Handbook of Research on User Interface Design and Evaluation for Mobile Technology*, 2007.
- [203] R. Harrison et al., "Usability of mobile applications: Literature review and rationale for a new usability model," *Journal of Interaction Science*, vol. 1(1), pp.1-16, 2013.
- [204] K. Walczak et al., "Adaptable mobile user interfaces for e-learning repositories," in *IADIS International Conference on Mobile Learning*, Avila, Spain, 2011, pp.52-61.
- [205] M. Kenteris et al., "Electronic mobile guides: A survey," *Personal and Ubiquitous Computing*, vol. 15(1), pp.97-111, 2011.
- [206] J. L. Wesson et al., "Can adaptive interfaces improve the usability of mobile applications?" *Human-Computer Interaction, IFIP Advances in Information and Communication Technology*, vol. 332, pp.187-198, 2010.
- [207] WhatsApp Messenger smartphone application [Online]. Available: <http://www.whatsapp.com> [Accessed: Jul. 23, 2013].
- [208] R. Y. M. Li, "Construction safety mobile knowledge sharing among generation y," *Romanian Review of Social Sciences*, vol. 3, pp.3-12, 2012.
- [209] K. Church and R. D. Oliveira, "What's up with WhatsApp? Comparing mobile instant messaging behaviors with traditional SMS," to be presented at *MobileHCI*, Munich, Germany, August 27-30, 2013.
- [210] R. Schumacher, "Improving database performance with partitioning," MySQL [Online]. Available: <http://dev.mysql.com/tech-resources/articles/partitioning.html> [Accessed: Aug. 07, 2013].
- [211] Microsoft, "Partitioning: Horizontal Partitioning," [technet.microsoft.com](http://technet.microsoft.com/en-gb/library/ms178148(v=sql.105).aspx) [Online]. Available: [http://technet.microsoft.com/en-gb/library/ms178148\(v=sql.105\).aspx](http://technet.microsoft.com/en-gb/library/ms178148(v=sql.105).aspx) [Accessed: Aug. 07, 2013].
- [212] Oracle, "Partitioning concepts," [docs.oracle.com](http://docs.oracle.com/cd/B28359_01/server.111/b32024/partition.htm) [Online]. Available: [http://docs.oracle.com/cd/B28359\\_01/server.111/b32024/partition.htm](http://docs.oracle.com/cd/B28359_01/server.111/b32024/partition.htm) [Accessed: Aug. 07, 2013].

- [213] RelationalDBDesign, “*Extended Database Features*,” relationaldbdesign.com [Online]. Available: <http://www.relationaldbdesign.com/extended-database-features/module2/database-partitioning-advantages.php> [Accessed: Aug. 07, 2013].
- [214] A. Smith, “*How to Calculate Calories Burned*,” Live Strong [Online]. Available: <http://www.livestrong.com/article/18303-calculate-calories-burned> [Accessed: Aug. 07, 2013].

## Appendices

### A1 Algorithms for the *Diet Diary* Scenario

Listed here are calculations and algorithms that have been programmed into the application module for the *Diet Diary* scenario.

#### A1.1 Body Mass Index

The *Body Mass Index* (BMI) of a user is calculated using the formula [163]:

$$BMI = \frac{kg}{m^2}$$

The calculated value is used to place the user into one of four BMI categories:

Underweight:	=<18.5
Normal:	18.5 – 24.9
Overweight:	25 – 29.9
Obese:	30+

#### A1.2 Daily calorie targets

Below is the algorithm for calculating the daily calorie intake and burn targets to reach a set weight goal:

*//Calculates total weight change from user's current weight to their goal weight*

Set WeightTotalChange to Person.Get(WeightCurrent) – Person.Get(WeightGoal)

*//1102.3 is the daily amount of calories that need to be burned to lose a kilogram of body fat in a week [164]*

Set CalBase to 1102.3

*//RDACal is the recommended daily allowance of calories to intake [165]*

Set CalEatDaily to Person.Get(RDACal)

*//Losing weight to reach lower weight goal*

IF WeightTotalChange > 0 THEN

Set WeightWeekChange = WeightTotalChange / Person.Get(WeeksToTarget)

Set CalBurnDaily = (CalBase \* WeightWeekChange) + CalEat

*//Gaining weight to reach higher weight goal*

ELSE IF WeightTotalChange < 0 THEN

Set CalBurnDaily to CalEatDaily

Set WeightWeekChange = -(WeightTotalChange) / Person.Get(WeeksToTarget)

Set CalEatDaily = (CalBase \* WeightWeekChange) + CalEat

```
//No Change in weight
ELSE
    CalBurnDaily = CalEatDaily
```

### **A1.3 Food item calorie intake**

Food items can be measured by either portion size or portion weight.

*CalDefault* represents the calorie value for a default quantity of a food item.

*CalDefault* stores value for a portion size of 1 or portion weight of 100.

For a food item selected from the scenario's available list, the total calorie intake of a food item is calculated by the following algorithm:

```
Set FoodQuantity to FoodUser.Get(Quantity)
Set CalDefault to FoodList.Get(CalDefault)

//Calculation for food items measured by weight
IF FoodList.Get(PortionType) = PortionType.Weight
    Set CalIntake to CalDefault * (FoodQuantity / 100)

//Calculation for food items measured by size
ELSE
    Set CalIntake to CalDefault * FoodQuantity

//Updating the daily calorie intake value
Set CalIntakeTotal to DayTotal.Get(CalIntakeTotal) + CalIntake

//Update the IntakeToAlert value to ascertain the distance to the user's preset alert limit
//This value is used by the AnswerHandler class to trigger an alert if necessary
Set IntakeToAlert to DayTotal.Get(IntakeToAlert) – CalIntake
```

For a food item that has been manually supplied by a user the calorie intake value for a default quantity is calculated for storage and future use of the food item:

```
Set FoodQuantity to FoodUser.Get(Quantity)
Set CalIntake to FoodUser.Get(CalIntake)

IF FoodList.Get(PortionType) = PortionType.Weight
    Set CalDefault to (CalIntake / FoodQuantity) * 100
ELSE
    Set CalDefault to CalIntake / FoodQuantity
```



#### A1.4 Activity item calorie burn

Each activity has a unique Metabolic Equivalent of Task (MET) value to measure the strenuous factor of the activity [161]. The MET value is then used in the below formula to calculate the calorie burn value for an activity [214]:

$$CalBurn = \frac{MET \times 3.5 \times Weight(kg)}{200} \times TimeSpent(mins)$$

For an activity item selected from the scenario's available list the calorie burn value is calculated by the following algorithm:

```
Set MET to ActivityList.Get(MET)
Set Weight to Person.Get(WeightCurrent)
Set TimeSpent to ActivityUser.Get(TimeSpent)

//Calculation for calorie burn value of the activity
Set CalBurn to (MET * 3.5 * Weight * TimeSpent) / 200
Set CalBurnTotal to DayTotal.Get(CalBurnTotal) + CalBurn
```

For an activity item that has been manually supplied by a user the MET value is calculated for storage and future use of the activity item:

```
Set CalBurn to ActivityUser.Get(CalBurn)
Set Weight to Person.Get(WeightCurrent)
Set TimeSpent to ActivityUser.Get(TimeSpent)

//Calcualtion for the MET value of the activity item
Set MET to (CalBurn * 200) / (3.5 * Weight * TimeSpent)
```

## **A2 Published Work**

The work carried out in this thesis has resulted in two publications.

### **Conference paper published and presented on 28 September 2012:**

J. Elton and P. W. H. Chung, “A Novel Server System to Support Different Types of Communication Services for Applications,” in *Proceedings of the 15<sup>th</sup> International Conference on Network-Based Information Systems*, Melbourne, Australia, 2012, pp.477-482.

This conference paper discusses the design elements of the platform, describing the client-server architecture and the interaction process for each type of client. The paper investigates the implementation of the *Diet Diary* scenario.

### **Journal article accepted and awaiting publication:**

J. Elton and P. W. H. Chung, “An integrated communications platform incorporating SMS and e-mail to support mobile applications,” *International Journal of High Performance Computing and Networking*.

This journal article goes into further detail regarding the design and implementation of the platform. There is an explanation of the data analysis process performed by the *Template Analysis Unit*. The article investigates the implementation of the *Missing Persons* scenario and concludes by suggesting some areas of future work.