CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Investigation into a best practice model for providing an integrated user experience with mobile cloud applications
Authors	O'Sullivan, Michael J.
Publication date	2016
Original Citation	O'Sullivan, M.J. 2016. Investigation into a best practice model for providing an integrated user experience with mobile cloud applications. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2016, Michael J. O'Sullivan. - http://creativecommons.org/licenses/by-nc-nd/3.0/
Download date	2026-11-02 05:17:02
Item downloaded from	https://hdl.handle.net/10468/2231



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Investigation into a Best Practice Model for Providing an Integrated User Experience with Mobile Cloud Applications

Michael J. O'Sullivan
BSc. (Hons) Computer Science

Thesis submitted for the degree of Doctor of Philosophy



National University of Ireland, Cork

Faculty of Science

Department of Computer Science

January 2016

Head of Department: Professor Cormac J. Sreenan

Supervisor: Dr. Dan Grigoras

Research supported by the EMBARK Initiative of the Irish Research Council

Contents

List of Figures vi
List of Tables ix
Abstract xi
Acknowledgements xiii

1 Introduction **1**
1.1 Introducing the Cloud to the Mobile 2
1.1.1 Mobile Devices 2
1.1.2 Cloud Computing 3
1.1.3 Mobile Cloud Computing 4
1.2 Integrated User Experience of Mobile Cloud Computing 5
1.3 Contributions 7
1.4 Publications 9

2 Literature Review **11**
2.1 Cloud Meets Mobile 11
2.2 Challenges of Mobile Cloud Computing and the User Experience 12
2.3 Approaching Integration of Mobile and Cloud 15
2.3.1 Local Infrastructure 16
2.3.2 Offloading and Partitioning 22
2.3.2.1 Application Partitioning 23
2.3.2.2 Code Offloading 26
2.3.3 Remote Display for Virtualised Desktops 33
2.3.4 Mobile Cloud Middlewares 35
2.3.4.1 Middleware Solutions with Service Oriented Architecture (SOA) 35
2.3.4.2 Other Middleware Solutions 38
2.3.5 Resource Management 40
2.3.6 Discussion 43
2.4 Context Awareness 45
2.4.1 Mobile Devices and Context Awareness 46
2.4.2 Context Representation and Storage 50
2.4.3 Context Awareness with Mobile Web Services 52
2.4.4 Discussion 54
2.5 Web Services 56
2.5.1 Mobile Web Services 56
2.5.1.1 Web Services on Mobile Devices 58
2.5.1.2 Mobile Devices using Server-Based Web Services 61
2.5.2 Service Discovery from Mobile Devices 62
2.5.3 Discussion 67
2.6 Conclusions 68

3 A Middleware for Providing an Integrated User Experience of Mobile Cloud Applications: Design and Architecture **70**

3.1 Introduction . . . 70
3.2 Future Approaches to Mobile Cloud Integration . . . 71
3.2.1 Requirements for an Integrated User Experience . . . 72
3.3 CAMCS: Context Aware Mobile Cloud Services . . . 74
3.3.1 Design . . . 74
3.3.2 The Cloud Personal Assistant (CPA) . . . 76
3.3.3 CPA Components and User Interaction . . . 78
3.4 Evaluation . . . 82
3.5 New Mobile Cloud Usage Scenarios and Enabled Features . . . 86
3.6 Conclusions . . . 88

4 Mobile Cloud Contextual Awareness with the Cloud Personal Assistant 89
4.1 Introduction . . . 89
4.2 Context Representation Requirements . . . 90
4.2.1 Context Representation . . . 91
4.2.2 Context Inference . . . 92
4.2.3 Context Storage . . . 93
4.2.4 Summary . . . 93
4.3 Context Processor Design and Architecture . . . 93
4.3.1 Sending User Context to the Context Processor with the Context Wrapper . . . 95
4.3.2 Consuming Stored Context from the Context Processor with the CPA . . . 96
4.3.3 Discussion . . . 96
4.4 Context Processor Implementation . . . 97
4.4.1 Contexts Utilised . . . 97
4.4.2 Ontology Implementation . . . 98
4.4.3 Collecting Context with the CAMCS Client . . . 99
4.4.4 Context Profiles . . . 99
4.4.5 Context Results . . . 100
4.5 Design, Implementation, and Use of Contexts . . . 101
4.5.1 Context Structure . . . 102
4.5.2 Developer Access for New Context Creation . . . 103
4.5.3 Custom Context Profiles . . . 105
4.5.4 Ontology Mapping . . . 107
4.5.5 Querying Context . . . 107
4.6 Evaluation . . . 109
4.6.1 Tourism Mobile Cloud Service . . . 109
4.6.1.1 Location Services with Foursquare . . . 109
4.6.1.2 Tourism Service Implementation . . . 109
4.6.2 Traffic Mobile Cloud Service . . . 111
4.6.2.1 Traffic Services with Twitter . . . 111
4.6.2.2 Traffic Services Implementation . . . 111
4.6.3 Traffic Mobile Cloud Service . . . 112
4.6.4 Custom Context Creation . . . 113

4.6.5 Discussion . . . 114
4.7 Conclusions . . . 115

5 Mobile Cloud Services - Description, Discovery and Consumption 117
5.1 Introduction . . . 117
5.2 Mobile Cloud Service Registry for User Oriented Service Descriptions . . . 119
5.2.1 The Service Registry . . . 119
5.2.2 Service Description Structure . . . 120
5.2.3 Querying . . . 126
5.2.3.1 Part-of-Speech Tagging of User Queries . . . 126
5.2.3.2 Semantic Similarity Calculation for Context-Aware Discovery . . . 127
5.3 Service Discovery and Consumption with the CPA . . . 130
5.3.1 Discovery . . . 130
5.3.2 Service Consumption . . . 133
5.3.3 Service Sessions . . . 135
5.4 CPA Task Model: Features and Implementation . . . 137
5.4.1 Task Structure . . . 137
5.4.2 Task Implementation . . . 138
5.4.3 Task Results . . . 140
5.4.4 Task Re-Runs . . . 140
5.5 User Privacy and Security Considerations . . . 142
5.5.1 CAMCS Authorisation for User CPAs . . . 142
5.5.2 Privacy of User Data . . . 144
5.6 Conclusions . . . 145

6 Integrating Mobile and Cloud Resources Management 147
6.1 Introduction . . . 147
6.2 Mobile Cloud Energy Management . . . 148
6.2.1 Antenna Energy Modelling . . . 149
6.2.2 Mobile Cloud Considerations for Energy . . . 151
6.3 Mobile Cloud Bandwidth Management . . . 153
6.4 Bandwidth Requirements . . . 154
6.4.1 Bandwidth Management . . . 156
6.5 Cloud Resource Management . . . 158
6.5.1 Cloud Resource Cost Model . . . 158
6.5.2 Mobile Cloud Usage . . . 159
6.6 Thin Client Modelling Experiments for the Mobile Cloud . . . 163
6.6.1 Signal Strength Experiments . . . 163
6.6.2 Mobile Cloud Distance Experiments . . . 166
6.6.3 Power Experiments . . . 169
6.7 Modelling for Thin Client Offloading . . . 174
6.7.1 Analysis . . . 174
6.7.2 Modelling the Offload Decision . . . 175
6.7.3 Model Implementation . . . 178
6.8 Discussion . . . 179

6.9 Conclusions 183

7 Mobile Cloud Application Models Facilitated by the Cloud Personal Assistant 185
7.1 Introduction 185
7.2 Application Models 186
7.2.1 File Storage/Synchronisation 187
7.2.2 Data Processing 188
7.2.3 Group-Based Collaboration 192
7.3 Implementation Challenges and Evaluation 195
7.3.1 File Synchronisation 195
7.3.1.1 OAuth Authentication 195
7.3.1.2 Service Provider APIs 196
7.3.1.3 Synchronisation from Service to Device 198
7.3.2 Data Processing 199
7.3.2.1 Service Extensibility 199
7.3.2.2 Discovery of Data Processing Services 200
7.3.3 Group-Based Collaboration 200
7.3.3.1 Collaboration Manager 201
7.3.3.2 Milestone Structure 201
7.4 Results 203
7.4.1 File Synchronisation 203
7.4.2 Data Processing 204
7.4.3 Group-Based Collaboration 207
7.5 Conclusions 208

8 Experimental Evaluation 210
8.1 Introduction 210
8.2 Evaluating CAMCS on a Public Cloud: Setup and Methodology 211
8.2.1 Amazon AWS Experimental Setup 211
8.2.2 Services Experimental Setup 213
8.2.3 CAMCS Experimental Setup 214
8.3 CAMCS Performance Evaluations on Amazon EC2 216
8.3.1 N-Queens Game Application on EC2 216
8.3.2 Service Registry Application on EC2 217
8.3.3 CAMCS - 500 Users with No Scaling 220
8.3.4 CAMCS - 500 Users with Horizontal Auto-Scaling 223
8.3.4.1 First Evaluation 225
8.3.4.2 Second Evaluation - Change of Scaling Policy . 230
8.3.5 CAMCS - 1000 Users with Horizontal Auto-Scaling . . . 235
8.3.6 CAMCS - 500 Users with Vertical Scaling 239
8.3.7 CAMCS - 10000 Users with Horizontal Auto-Scaling . . 242
8.4 Mobile Device Performance Evaluations with the CAMCS Client 243
8.4.1 Initial Installation Resource Usage 243
8.4.2 Resource Usage For Task Work 243
8.5 User Study 247
8.5.1 Study Setup 250

8.5.2 Study Survey Results . . . 251
8.6 Analysis . . . 253
8.6.1 Mobile Cloud Resource Usage Evaluation . . . 253
8.6.1.1 Energy Usage . . . 253
8.6.1.2 Bandwidth Usage . . . 254
8.6.1.3 Cloud Resource Usage . . . 254
8.6.2 Meeting User Experience Requirements . . . 255
8.7 Conclusions . . . 260

9 Conclusions and Future Work 262
9.1 Summary of Research Problem and Solution . . . 262
9.2 Conclusions . . . 263
9.2.1 Benefits . . . 268
9.2.2 Limitations . . . 269
9.3 Future Work . . . 270

A User Study Survey Responses 291
A.1 Part 1: Mobile Experience . . . 291
A.2 Part 2: CAMCS and Your CPA . . . 292
A.3 Part 3: Contextual Awareness Support with the CPA . . . 295
A.4 Part 4: User Experience Perceptions . . . 297

B OWL Context Ontology Example 298

C User Oriented Mobile Cloud Service Description Examples 301
C.1 Google Calendar Service . . . 301
C.2 Foursquare Tourism/Places of Interest Service . . . 305
C.3 Lufthansa Flight Information Service . . . 307

List of Figures

2.1 Cloudlet Diagram . . . 18
2.2 Application Partitioning Diagram . . . 24
2.3 Code Offloading Diagram . . . 27
2.4 Remote Display Diagram . . . 33
2.5 Middleware Diagram . . . 37
2.6 Context Awareness Diagram . . . 48

3.1 CAMCS System Architecture Diagram . . . 74
3.2 CAMCS Task Flow Diagram . . . 77
3.3 Detailed CAMCS Component Diagram . . . 79
3.4 CAMCS Component Sequence Diagram for Task Execution . . . 83

4.1 Context Processor . . . 95
4.2 Receiving a Location Context Update . . . 96
4.3 Context Profile Selection . . . 102
4.4 Context Structure . . . 105
4.5 Context Definiton Structure . . . 106
4.6 CPA Running a Location Information Task with Foursquare . . . 110
4.7 Foursquare Location Task Result . . . 112
4.8 Twitter Traffic Task Result . . . 114

5.1 Service Description Structure Diagram . . . 121
5.2 Android Thin Client: New Task Creation . . . 126
5.3 Task Execution Process Interaction Diagram . . . 131
5.4 Android Thin Client: Current Tasks List . . . 132
5.5 Android Thin Client: Discovered Service Selection . . . 133
5.6 Android Thin Client: Parameter Data Entry for Service Consumption . . . 134
5.7 Android Thin Client: More Information Required for Task . . . 135
5.8 Android Thin Client: Selection Step for Service Consumption . . . 136
5.9 Android Thin Client: Context Parameter Permissions . . . 137
5.10 Android Thin Client: Parameter Data Entry for Service Consumption . . . 141
5.11 Android Thin Client: Calendar Task Result from Google Calendar Service . . . 141

6.1 Amazon vs UCC 2MB File Offload Time Boxplot . . . 166
6.2 Amazon vs UCC 1MB File Offload Time Boxplot . . . 166
6.3 Amazon vs UCC 500KB File Offload Time Boxplot . . . 167
6.4 Amazon vs UCC 250KB File Offload Time Boxplot . . . 167
6.5 Amazon vs UCC 125KB File Offload Time Boxplot . . . 168
6.6 Amazon vs UCC 500KB File Offload Time Boxplot for Nexus 5 Device . . . 168
6.7 Amazon vs UCC 2MB Offload Time Boxplot for Servers against Constant Signal Range . . . 169

6.8 Amazon vs UCC 250KB Offload Time Boxplot for Servers against Constant Signal Range . . . 170
6.9 (-89, -80) dBm GSM Signal Range File Offload Power Consumption Graph . . . 171
6.10 (-110, -100) dBm GSM Signal Range Power Consumption Graph 172
6.11 Power Consumption Graph for Device Connected to USB Power 172

7.1 CPA File Sync . . . 189
7.2 CPA Data Processing . . . 192
7.3 CPA Group Tasks . . . 194
7.4 OAuth Credentials with the CPA . . . 197
7.5 JAR Files for Service Providers . . . 198
7.6 Graph Model for Group Tasks . . . 202

8.1 N-Queens Game EC2 Response Time for N = 10 . . . 217
8.2 AWS CloudWatch Monitoring CPU Utilisation for N-Queens Game Test with N = 10 . . . 218
8.3 Service Registry EC2 Response Time . . . 219
8.4 AWS CloudWatch Monitoring CPU Utilisation for Service Registry EC2 Instance . . . 220
8.5 AWS CloudWatch Monitoring CPU Utilisation for CAMCS EC2 Instance with No Scaling, Service Discovery and Consumption Disabled . . . 221
8.6 AWS CloudWatch Monitoring CPU Utilisation for Registry EC2 Instance During CAMCS test with Service Discovery Enabled . 223
8.7 AWS CloudWatch Monitoring CPU Utilisation for Services EC2 Instance During CAMCS test with Service Consumption Enabled 224
8.8 AWS CloudWatch Monitoring CPU Utilisation for CAMCS EC2 Instance One, 500 Users, Horizontal Auto-scaling, First Evaluation . . . 226
8.9 AWS CloudWatch Monitoring CPU Utilisation for CAMCS EC2 Instance Two, 500 Users, Horizontal Auto-scaling, First Evaluation . . . 227
8.10 AWS CloudWatch Monitoring CPU Utilisation for CAMCS EC2 Instance Three, 500 Users, Horizontal Auto-scaling, First Evaluation . . . 227
8.11 AWS CloudWatch Monitoring CPU Utilisation for CAMCS EC2 Instance Four, 500 Users, Horizontal Auto-scaling, First Evaluation . . . 228
8.12 t2.micro CPU Credits Balance, Instance Three, Horizontal Scaling, First Evaluation . . . 230
8.13 AWS CloudWatch Monitoring Load Balancer Successful Requests for 500 Users, Horizontal Auto-Scaling, First Evaluation 231
8.14 AWS CloudWatch Monitoring Load Balancer Average Latency for 500 Users, Auto-Scaling, with Service Discovery and Consumption . . . 231

8.15 AWS CloudWatch Monitoring CPU Utilisation CAMCS Instance One, 500 Users, Horizontal Auto-Scaling, Second Evaluation . . 232
8.16 AWS CloudWatch Monitoring CPU Utilisation CAMCS Instance Two, 500 Users, Horizontal Auto-Scaling, Second Evaluation . . 233
8.17 AWS CloudWatch Monitoring Load Balancer Successful Requests for 500 Users, Auto-Scaling, with Service Discovery and Consumption 234
8.18 AWS CloudWatch Monitoring Load Balancer Average Latency for 500 Users, Auto-Scaling, with Service Discovery and Consumption 235
8.19 AWS CloudWatch Monitoring CPU Utilisation CAMCS Instance One, 1000 Users, Horizontal Auto-Scaling 237
8.20 AWS CloudWatch Monitoring CPU Utilisation CAMCS Instance Two, 1000 Users, Horizontal Auto-Scaling 237
8.21 CAMCS EC2 JMeter Response Time Graph, 1000 Users, Horizontal Auto-Scaling 238
8.22 AWS CloudWatch Monitoring Load Balancer Successful Requests for 1000 Users, Horizontal Auto-Scaling 239
8.23 AWS CloudWatch Monitoring Load Balancer Average Latency for 1000 Users, Horizontal Auto-Scaling 240
8.24 AWS CloudWatch Monitoring CPU Utilisation CAMCS c4.xlarge Instance 241
8.25 CAMCS Thin Client Details after Initial Installation 244
8.26 Trepn Profiler Battery Consumption and Data Usage Graph for Calendar Task Interactions, Wi-Fi Connection 245
8.27 Trepn Profiler Android Application CAMCS Resource Consumption Metrics for Calendar Task Interactions, Wi-Fi Connection 246
8.28 Trepn Profiler Android Application OS Resource Consumption Metrics for Calendar Task Interactions, Wi-Fi Connection 246
8.29 Trepn Profiler Battery Consumption and Data Usage Graph for Calendar Task Interactions, 3G Connection 248
8.30 Trepn Profiler Android Application Resource Consumption Metrics for Calendar Task Interactions, 3G Connection 249
8.31 Trepn Profiler Android Application OS Resource Consumption Metrics for Calendar Task Interactions, 3G Connection 249

List of Tables

3.1 Properties Table 82

6.1 Offload Decision Model Variables 175
6.2 Mobile Cloud Approaches - Resource Usage Comparison 182

7.1 Device to CPA Upload Times 205
7.2 Device App Upload Times 205
7.3 XML Processing Cloud Times 206
7.4 XML Processing Mobile Times 206

8.1 Spring Boot and Apache Tomcat Properties 211
8.2 AWS t2.micro Instance Specifications 213
8.3 N-Queens Game EC2 JMeter Metrics 216
8.4 Service Registry EC2 JMeter Metrics 218
8.5 CAMCS EC2 JMeter Metrics with No Scaling 221
8.6 CAMCS Memory Usage with Docker 222
8.7 CAMCS EC2 JMeter Metrics, 500 Users, Auto-Scaling, First Evaluation 226
8.8 CAMCS EC2 JMeter Metrics, 500 Users, Auto-Scaling, Second Evaluation 234
8.9 CAMCS EC2 JMeter Metrics, 1000 Users, Horizontal Auto-Scaling 236
8.10 AWS c4.xlarge Instance Specifications 241
8.11 CAMCS EC2 JMeter Metrics Vertical Scaling using Large Compute-Optimised Instance, Service Discovery and Consumption Disabled 241

I, Michael J. O'Sullivan, certify that this thesis is my own work and has not been submitted for another degree at University College Cork or elsewhere.

_____

Michael J. O'Sullivan

Abstract

Mobile Cloud Computing promises to overcome the physical limitations of mobile devices by executing demanding mobile applications on cloud infrastructure. In practice, implementing this paradigm is difficult; network disconnection often occurs, bandwidth may be limited, and a large power draw is required from the battery, resulting in a poor user experience.

This thesis presents a mobile cloud middleware solution, Context Aware Mobile Cloud Services (CAMCS), which provides cloud-based services to mobile devices, in a disconnected fashion. An integrated user experience is delivered by designing for anticipated network disconnection, and low data transfer requirements. CAMCS achieves this by means of the Cloud Personal Assistant (CPA); each user of CAMCS is assigned their own CPA, which can complete user-assigned tasks, received as descriptions from the mobile device, by using existing cloud services. Service execution is personalised to the user's situation with contextual data, and task execution results are stored with the CPA until the user can connect with his/her mobile device to obtain the results.

Requirements for an integrated user experience are outlined, along with the design and implementation of CAMCS. The operation of CAMCS and CPAs with cloud-based services is presented, specifically in terms of service description, discovery, and task execution. The use of contextual awareness to personalise service discovery and service consumption to the user's situation is also presented. Resource management by CAMCS is also studied, and compared with existing solutions. Additional application models that can be provided by CAMCS are also presented.

Evaluation is performed with CAMCS deployed on the Amazon EC2 cloud. The resource usage of the CAMCS Client, running on Android-based mobile devices, is also evaluated. A user study with volunteers using CAMCS on their own mobile devices is also presented. Results show that CAMCS meets the requirements outlined for an integrated user experience.

This thesis is dedicated to my father, Michael, my mother, Margaret, and my sister, Laura, for all the support and patience shown to me while completing this research, and for all the sacrifices and hardships they have endured to enable me to reach my full potential.

Acknowledgements

My PhD research has been a very personal journey that arguably only a few years ago, I probably would not have been able to undertake. During my time in school, with frequent absence due to illness, one could be forgiven for thinking I would never be able to attend University to undertake an undergraduate degree, let alone a PhD. I can still remember walking into my final Leaving Certificate (Irish school-leaving) exam, at the end of June in 2008. The exam was Higher Level Applied Mathematics. I took all my exams in a separate room, in the event I became unwell. The invigilator knew about my condition, and when I sat down, he first looked at the exam paper, then to me, and said "Just how the h*** did you manage to do this?". Looking back on that time now, I'm still not quite sure how I coped, and ended up at this day, writing the final part of my PhD thesis.

What cannot be denied is the support I received from many people over the years, who had the patience and belief in me to finally reach this point in my life. I would like to start by thanking Professor Cormac Sreenan, Head of Department of Computer Science, and Dr. John Herbert, Senior Lecturer, for sponsoring my original application for my PhD scholarship back in 2012. A second thanks must also go to Professor Sreenan, and additionally to Professor Chunming Rong, University of Stavanger, Norway, in their roles as my internal and external examiner of this thesis respectively. Both showed great knowledge and insight into my work during my Viva Voce examination, with excellent observations and suggestions for the final version of this thesis.

I wish to thank and acknowledge the support of the Irish Research Council for sponsoring this research with the EMBARK scholarship I received. I would also like to thank all the participants of the user study undertaken in this work for their time and cooperation.

I want to thank all my colleagues in the Mobile and Internet Systems Laboratory (MISL) for their friendship, support, and for paying complete attention to my reading group presentations; Ilias Tsompanidis, Dapeng Dong, Dr. Jason Quinlan, Darijo Raca, Rezvan Pakdel, Mohammad Hashemi, Seamus O Buadhachain, Jonathan Sherwin, Dr. Thuy Truong, Dr. Estanislao (Lau) Mercadal, Dr. Ahmed Zahran, Dr. Lanny Sitanayah, Dr. Neil Cafferkey, Dr. Paul Davern, Dr. Xiuchao Wu, Dr. A.K.M. Mahtab Hossain, Saim Ghafoor, Jerome Arokkiam, Samreen Umer, Dr. David Stynes, and Dr. Mustafa Al-Bado. I also want to thank Mary Noonan in MISL who took such great care of myself and everyone in MISL. Thanks also goes to Derbhile Timon, and everyone in the department office, for all their kind support and assistance to me.

A very special thanks also goes to my two mobile-cloud PhD colleagues, Hazzaa Alshareef, and Aseel Alkhelaiwi, for sharing in this adventure with me from the high points to the lows, be it celebrating our papers being accepted for publication, or wondering just why Android cannot do the very thing we

need it to do. I couldn't have completed this work without the support each of you gave to me. Another special thanks also goes to Harshvardhan Pandit for your friendship and support during my studies; our chats were always the most welcome of distractions, and bumping ideas off one another for our research was always a great pleasure.

I also want to extend a special thank you to every student I had the great pleasure of teaching during my studies, at both BSc and MSc levels. It's no secret that being involved with teaching in the labs, and the lectures I got to present, were highlights of the last few years, with one student even asking me "Do you not have a home to go to Michael?" one afternoon when I stayed for an hour after class to answer questions. Seeing some of the great work done by students was inspiring, and also encouraged me to press on during tough days. I'm still also really grateful to the Computer Science BSc class of 2015, some of whom are now my colleagues at work, who I believe nominated me for the Best Demonstrator Award in the College of SEFS in UCC, which I was delighted to win. That was a tremendous honour.

Very special thanks must go to the staff of the Adult Cystic Fibrosis Care Team in Cork University Hospital. To my physios Claire and Pat, dieticians Ciara and Karen, nurses Claire and Mairead, head nurse Cathy Shortt, Professor Barry Plant, and everyone else involved in my care; thank you for all your hard work in keeping me so healthy during this time.

I would also like to thank my colleagues at work in IBM in Cork for their support and patience during my first few months of employment while I was finishing my PhD, especially my manager, Richard Holland, and my squad leader, Sanjay Nayak.

I want to extend a very special thank you to my family; Michael, Margaret, and Laura. I know that putting up with my illness and all the hard times I have brought on our family over the years has been very tough for all of you, and I can never repay all the care and support you have provided to me, all through my life. I hope this work, even though I know you really don't understand any of it, goes someway towards repaying that which I cannot repay. Of course, I also want to thank my extended family for their support as well.

Finally, I want to extend a big thank you to my supervisor, Dr. Dan Grigoras, Senior Lecturer, for also sponsoring my PhD application at the very start, and agreeing to take me on as a student back for my final year BSc project in 2011. Of course, a "big thanks" is really not enough though for everything that I owe you from the last few years. It has been the greatest experience of my life working with you. I have learned so much, and every challenge that you presented to me has been a thrill to undertake. Thank you for teaching me how to "think big" about ideas and concepts, having been so full of doubt and lacking confidence in myself at the start.

Michael O'Sullivan

Cork, Ireland, 16th January 2016.

Chapter 1

Introduction

The use of smart mobile devices has increased significantly over the past several years. Mobiles and tablets are now the most common hardware devices used to access online services, overtaking the traditional desktop PC, and laptops. However, the methods by which mobile users access online services is influenced by service access from these traditional devices. This access takes two forms; a user could install a piece of software locally on their computer to utilise online services, or more commonly, the user accesses software online using a web browser. For the former, the locally installed software and its data was constrained to the computer it was installed on. For example, word processing software such as Microsoft Word. Documents are only stored locally. When the latter method of accessing online software through the web browser came to fruition, the software service was normally provided as a web-application. Microsoft Office 365, the latest offering in the Office suite of products, is now accessed through a web browser. The greatest advantage from this is that the user can now access this service and all their documents, from any device in the world with an internet connection. Microsoft Office 365 is a cloud-based product; with the advent of cloud computing, the use of a web browser has become the most common means by which software is accessed and interacted with.

In the context of mobile computing, similar approaches are in use today. Users can access software by downloading it as "apps" from application markets on the device, such as Google Play for Android-based devices, or the Apple App-Store for iOS-based devices. This software can then access online services. Similarly, the user can also access online software through a web browser on

the mobile device.

As the popularity of mobile devices has climbed, partly due to more connectivity options, and a larger range of software available for the mobile device, users expect to be able to do more with their devices. Due to the limits of the physical resources found within mobile devices, as well as their portability being key to the idea of mobility, the question arises; how can the capabilities of mobile devices be increased? The solution to this problem in current research comes from the idea of utilising cloud-based resources, to enhance the capabilities of mobile devices.

1.1 Introducing the Cloud to the Mobile

Integrating mobile devices with services provided by the cloud is a promising paradigm, and can be seen as a sensible one. The provisioning of cloud-based software and services to clients is known as Software as a Service (SaaS). A report by IDC concludes that revenue generated by the cloud SaaS market will grow to $76.1 billion by 2017 (up from $28 billion in 2012), and "will significantly outpace traditional software product delivery" [59]. Additionally, a white-paper by Cisco has identified the use of cloud services as content providers as one of the five global trends for their 2012-2017 Global Cloud Index [134]. Another of their trends is remote access to such services from mobile devices; especially cloud storage as a "content locker", for both personal and enterprise users.

Combining the resources of mobile devices with cloud infrastructure can be seen as a complex equation. It is worthwhile to understand precisely how the capabilities of mobile devices are limited, and what solutions cloud-computing can offer.

1.1.1 Mobile Devices

As the hardware resources in these devices has improved, so has the capabilities of mobile software. Originally, many of these devices were used as just phones, but now users can access the web, listen to music, and play games among other activities. The perception of the role of these devices is changing. Users expect these devices to be able to do far more than just make phone

calls. Now that the capabilities for these tasks exist, mobiles have become the primary device used to browse the web, taking the place of desktop and laptop computers.

The expectations and use of these devices is clearly changing, yet these devices are still not capable of carrying out more intensive tasks. This is partly due to the limited resources on the device. These include; limited processing power and speed, lack of memory, lack of storage space, intermittent connection and quality of the mobile networks, and finally, the limited battery capacity.

Some of these limitations are to be expected of a mobile device. Mobility and portability are the primary physical attributes that distinguish these devices from their desktop counterparts. Adding better resources onto these devices would make them bigger and heavier, defeating their purpose. Their form factor dictates that they will not feature abundant physical resources. Moreover, even if larger resources could be built into these devices, they would almost certainly drain the energy supply from the battery at a higher rate. To demonstrate this problem, the authors in [21] carried out an experiment to see how long it would take to drain a full smartphone battery, while keeping the display turned on, and invoking a task with continuous network I/O, that also consumed the entire CPU. The network I/O and energy consumption is an important consideration for applications that require a continuous stream of data. They found the battery to be drained after only one hour and twenty minutes.

1.1.2 Cloud Computing

Cloud computing is a distributed computing paradigm with the purpose of delivering computing resources on a pay-per-use basis to subscribers [15]. These resources can be physical, such as CPU power/time, memory, storage, and network I/O. Such resources are known as *Infrastructure as a Service (IaaS)*. Application platform resources are also provided to software developers for developing and deploying applications. These resources are known as *Platform as a Service (PaaS)*. End user software can also be deployed in the cloud, such as Microsoft Office 365. This kind of cloud resource is known as *Software as a Service (SaaS)*. Large companies provide these resources in several locations around the world in large data centres featuring an ever growing number of servers. The big names in this space are Amazon, Microsoft, and

Google. Amazon's cloud offering is Amazon Web Services (AWS). Microsoft's product is known as Microsoft Azure, and Google's cloud platform is known as Google AppEngine.

The greatest benefit of cloud computing is its ability to provide elastic resources. Resources can be dynamically scaled up and down as and when required. If an application running on a virtual machine (VM) deployment in the cloud reaches a certain user load, a dynamic load balancer can automatically create and deploy another virtual machine instance running the software, and balance traffic between these instances. When the user traffic reduces, under-loaded instances can be shut down automatically. None of these tasks require any administrative user intervention, which is another advantage of cloud computing.

1.1.3 Mobile Cloud Computing

To address the resource limitations present in mobile devices, research has tried to combine the infinitely growing resources of cloud infrastructure with mobile devices. This research area has become known as *Mobile Cloud Computing* (MCC). Commonly, the research and enterprise pattern of building and provisioning clouds that offer services and applications specifically to mobile devices, has become informally known as simply *mobile cloud*.

The common use case of mobile cloud computing, is that a mobile application will offload its resource intensive work to cloud infrastructure for execution. Once the work is complete, the results of the execution are returned to the mobile device, where execution continues. Several approaches to this have been taken in the research literature, and these are reviewed in Chapter 2. Utilising this approach, applications that would normally be considered too demanding for the limited resources of a mobile device (a common use-case being augmented reality applications), can now be executed in the cloud, with output delivered to the mobile device.

However, there are limitations to the effectiveness of mobile cloud computing. Implementing the mobile cloud computing paradigm is a difficult challenge. The main argument for the use of mobile cloud computing, is that utilising cloud infrastructure for execution of mobile applications will spare the limited resources of the mobile device, as intensive computation will not be performed on the device. The reality of the situation is that implementing the paradigm

commonly results in its own costs to the mobile device.

The difficulties in implementing the mobile cloud computing paradigm result from several challenges that must be addressed. For example, many approaches require a continuous, uninterrupted, and high-quality connection to the cloud infrastructure. In a mobile world, this requirement is simply not practical or realistic. The primary attribute of mobile computing is mobility itself; users will constantly be on-the-move, changing from Wi-Fi to cellular networks (and vice-versa), experience handover, and even become disconnected in areas with poor signal. Another common challenge faced is the requirement for continuous, high-volume data transfer. Aside from the fact that lack of uninterrupted connection will prevent this requirement already, there is a high energy cost incurred by mobile devices for using their network interfaces to transfer data. This can result in the limited battery power supply draining quickly. On cellular networks, mobile network operators (MNOs) can charge high rates for data-usage. A final example of a problem, is the commonly low bandwidth available on cellular 3G/4G networks. This results in high latency connections, which results in slow network performance, and naturally, a slow-down in the performance of any applications waiting on network I/O to complete. As a result of longer waiting times, energy usage on the mobile device is commonly higher on cellular networks, compared to Wi-Fi networks.

1.2 Integrated User Experience of Mobile Cloud Computing

Several papers in the research literature ultimately are faced with a trade off when it comes to benefiting from the promise of mobile cloud computing. Consider a mobile user out and about, connected to a cellular network, while using a demanding application. This application can benefit from offloaded execution to the cloud. Unfortunately, the cellular network has low bandwidth available, and also suffers from high latency. In such a situation, some of the mobile cloud solutions developed recognise that it may be more costly to offload computation to the cloud, compared with local execution. High power usage may be required, as well as time and patience from the user. In this example, the application will examine the state of the network, by sending out packets and judging low long it takes for responses (pings). Based on the results, the appli-

cation estimates a cost for local execution, which consists of utilising the CPU. The application also estimates a cost for offloading the computation. This consists of network transfer for the offload, along with keeping the connection open and waiting for a response. The size of the data to be offloaded, must also be considered.

Only if the cost for remote, offloaded execution is cheaper than the cost for local execution, should the cloud infrastructure be used. As shown in the experiments of one related work examined in the literature review chapter [21], the mobile application, not once in all experiments, opted for remote cloud execution, instead always choosing local execution. When the authors modified the test application to force offloading on the cellular network, performance was far worse than local execution. The choice of using the cloud for the mobile user in this example would have resulted in a *poor user experience*.

At first glance, it might appear that mobile cloud computing simply exacerbates the problem it claims to solve, but this need not be the case. The aim of the research presented in this thesis, is to deliver a mobile cloud computing solution that provides an *integrated user experience* of mobile cloud applications. This is an essential requirement for any solution that implements the mobile cloud computing paradigm. If the approach taken to implementing the paradigm delivers an experience that proves detrimental to the user's perception of an application's performance, or drains the limited resources of the device, then that approach will not be adopted by mobile users. The example of the mobile user given above, should outline why consideration of the user experience is of such fundamental importance to all solutions, yet, for the most part, is completely neglected in the research literature for mobile cloud computing solutions.

An integrated user experience of mobile cloud computing is more formally defined in Chapter 3. For the purposes of this introduction, one can think of an integrated user experience of mobile cloud computing, as *an experience that provides seamless interaction between the mobile device and the cloud, with no detrimental impact or costs incurred by the device*. A mobile cloud computing solution that adopts this experience as a primary implementation requirement, will not encounter the problems outlined in the previous section. The solution will adapt to mobility constraints by continuing to perform in disconnected scenarios, i.e. it does not require a continuous, uninterrupted, high quality connection to the cloud. The solution does not require continuous, high-volume

data transfer, instead relying on infrequent, low data-volume transfers. Also, the solution does not require a high power drain from the battery, either from network activity, or computation-intensive work. It should be noted that the integrated user experience perspective taken in this work is that of the more favourable user experience delivered on the mobile device by respecting the limited physical resources available, rather than a study of user Quality of Service (QoS) perception, or direct human-computer interaction (HCI) experience with mobile cloud applications and services. However, a user-study gathering feedback from volunteers was conducted, and is presented as part of the evaluation.

The solution implemented in this research, and presented in this thesis, was designed to deliver this integrated user experience of the mobile cloud. The research shows that a disconnected, asynchronous approach to mobile cloud computing, which delivers an integrated user experience of applications and services is both feasible and practical, and can be delivered by the solution presented. This has not been achieved in existing research.

1.3 Contributions

The research presented in this thesis provides the following contributions to mobile cloud computing research:

1. An analysis of related work implementing the mobile cloud computing paradigm, with a focus on the user experience they provide. This is the subject of Chapter 2. From this, requirements are derived that future approaches to mobile cloud computing should adopt to provide an integrated user experience of mobile cloud applications. This is the subject of Chapter 3.

2. A mobile cloud middleware solution, Context Aware Mobile Cloud Services (CAMCS), which delivers an integrated user experience of mobile cloud applications, by delivering services to smart mobile devices using an asynchronous, disconnected approach. A thin client application, the CAMCS Client, is also provided for the mobile user to interact with CAMCS. The main component behind the CAMCS middleware is the Cloud Personal Assistant (CPA). A CPA runs within CAMCS, completing user-assigned tasks by using mobile cloud services. Each user of CAMCS

is assigned their own CPA. CAMCS can be compared favourably with related work, which do not respect the user experience as a specific requirement, resulting in detrimental impacts on resources such as energy and bandwidth usage. This is presented in Chapter 3.

3. Support for utilising user context-data gathered from the CAMCS Client on the mobile device. Compared with related work which typically just provides a middleware for supporting collection, processing, and storage of context data, this research shows, additionally, how CAMCS can use this data for mobile cloud service discovery, and how the data can be provided to mobile cloud services to personalise their execution to the user's situation. This is the subject of Chapter 4.

4. A novel mobile cloud service description format, which provides user-oriented service descriptions to the mobile user, allowing participation in the service discovery process. This is used in place of traditional XML-based solutions, which are not suitable as a solution to mobile cloud service discovery, as such formats and the technical meta-data they contain cannot be comprehended by ordinary end-users. This is the subject of Chapter 5.

5. A mobile cloud service discovery and consumption solution for use with CAMCS, and the CAMCS Client. The aim of this solution is to be user friendly, and accessible to users with no technical background. This can be compared favourably with existing solutions, which are only capable of providing support for a limited number of services, and require the end user to know technical details of the service before invocation (e.g. WSDL file location, SOAP or RESTful service type). This is also presented in Chapter 5.

6. Guidelines and models for resource management in mobile cloud computing. A novel analysis compares existing mobile cloud computing solutions and CAMCS with these models, to understand how these works make use of the limited resources available. Experiments are presented which evaluate the impact of signal strength and cloud-infrastructure location on mobile-to-cloud network communication, as well as experiments which determine the resulting power usage of this communication on a cellular network. Based on these experimental results, a zero-overhead algorithm is developed for queuing communications between the mobile device and the cloud, when network and device conditions

are not optimal. This is in comparison to other profiling algorithms, which work by performing costly network probes. This is the subject of Chapter 6.

7. Additional application models provided by CAMCS; file synchronisation across cloud service providers, XML data processing, and group-based CPAs for collaborative tasks. Guidelines based on implementation challenges for future mobile cloud application development are presented, along with experimental results showing the benefits of a CPA approach to these application models. This is the subject of Chapter 7.

8. A thorough evaluation of the performance of CAMCS, deployed on the Amazon EC2 cloud infrastructure. A performance evaluation of the CAMCS Client, with a view towards resource usage is also presented. A user study was also performed, where participants used CAMCS via the CAMCS Client installed on their own Android mobile devices for several days. A survey of their experience and perceptions was taken at the conclusion of the study, and these results are also presented. The results of these evaluations and the user survey responses show that CAMCS is an effective mobile cloud computing solution, delivering an integrated user experience of the mobile cloud computing paradigm, by meeting the requirements that were derived for an integrated user experience in Chapter 3. These evaluations and results are presented in Chapter 8.

1.4 Publications

Chapters of this thesis are made up of the following publications:

[98] M. J. O'Sullivan and D. Grigoras. User Experience of Mobile Cloud Applications - Current State and Future Directions. In Proceedings of 12th International Symposium on Parallel and Distributed Computing (ISPDC), pages 85-92. 2013, IEEE.

[97] M. J. O'Sullivan and D. Grigoras. Mobile Cloud Application Models Facilitated by the CPA. In Proceedings of 2013 International Conference on MOBILe Wireless MiddleWARE, Operating Systems and Applications (Mobilware), pages 120-128. 2013, IEEE.

[99] M. J. O'Sullivan and D. Grigoras. Mobile Cloud Contextual Awareness

with the Cloud Personal Assistant. In Proceedings of 2014 International Conference on Future Internet of Things and Cloud (FiCloud), pages 82-89. 2014, IEEE.

[103] M. J. O'Sullivan and D. Grigoras. Integrating Mobile and Cloud Resources Management using the Cloud Personal Assistant. Simulation Modelling Practice and Theory, 50:20-41. 2015, Elsevier.

[102] M. J. O'Sullivan and D. Grigoras. Mobile Cloud Application Models Facilitated by the CPA. EAI Endorsed Transactions on Scalable Information Systems, 15(4). 2015, ICST.

[100] M. J. O'Sullivan and D. Grigoras. Delivering Mobile Cloud Services to the User: Description, Discovery, and Consumption. In Proceedings of 4th International Conference on Mobile Services, pages 49-56. 2015, IEEE.

[101] M. J. O'Sullivan and D. Grigoras. Context Aware Mobile Cloud Services: A User Experience Oriented Middleware for Mobile Cloud Computing. In Proceedings of 4th International Conference on Mobile Cloud Computing, Services, and Engineering (Mobile Cloud). 2016, IEEE. To Appear.

The end of each chapter will specify which of the above papers they were based on, where appropriate.

Chapter 2

Literature Review

The focus of this review is on the approaches taken to provide integration between mobile devices and cloud resources in the literature, with a view to the user experience of using applications and resources on the device. This integration can greatly enhance the potential of the mobile device, but it is a challenging problem for many reasons which are examined. Some approaches in the literature are more prominent than others, but even these approaches may not be the ideal solution with their own shortcomings, which will be discussed. Other areas that are of relevance to the research presented in this thesis are also reviewed; namely, mobile contextual awareness, and mobile web services.

2.1 Cloud Meets Mobile

In the context of mobile cloud computing, it is expected that the infinite resources of these cloud data centres can be used to address the physical resource limitations of mobile devices. These devices can utilise the plentiful cloud resources by sending tasks and intensive applications to the cloud, with the completed results returned to the mobile device upon completion.

Satyanarayanan demonstrated in [119] several scenarios where mobile cloud computing can prove extremely useful, if not essential. Most of these scenarios have some real time requirement, such as a patient with Alzheimer's disease, who has trouble remembering people or objects in his/her environment. By wearing a special pair of glasses which can connect to the mobile cloud, visual cues could appear in the lens, or be spoken into the ear with an attached ear-

phone. For example, when the glasses focus on a person, his/her name could be displayed on the lens or played into the ear, as a result of facial recognition. For this to be feasible, this must happen in real time. The facial recognition work required here would be too demanding for the mobile device alone, and the database of faces would come from cloud infrastructure. Satyanarayanan et al [117] identifies that the big obstacle in the way of real time requirements is the latency between the mobile device and the cloud infrastructure. This issue and a proposed solution is explored shortly, when solutions based on computing infrastructure near the user's location, are discussed.

With the context of mobile cloud computing now outlined, some of the difficulties in the way of realising this paradigm are examined.

2.2 Challenges of Mobile Cloud Computing and the User Experience

While several different paths have been taken to approach cloud utilisation from mobile devices, it can be seen that several challenges across all paths are common. Some of these are highlighted in the work by Simoens et al in [124]. These challenges highlighted by the authors are as follows:

Device Battery Lifetime - Any solution should not result in a large drain on the limited power provided by the battery.

Wireless Bandwidth Availability - Transferring large amounts of data between the device and the cloud is not ideal, as the cost incurred by the mobile user in cellular 3G/4G LTE networks can be high, and take far too long with the limited bandwidth available in these networks.

Interaction Latency - Some applications have real-time or near-real-time requirements, such as multimedia or gaming applications. Where the latency between the mobile device and the cloud is very large, unacceptable performance may result for these applications, or they could become unusable [136].

Other challenges include:

User Mobility - The mobile user is constantly on the move with changing context, and handoff takes place between mobile networks.

Device Disconnection - The user may move out of range of a mobile network and lose connectivity. This can be especially troublesome for applications that require a continuous connection to cloud services, which most applications do.

Mobile Device Heterogeneity - The multitude of different mobile device types and the platforms they are powered by can make developing a "one-fits-all" solution very challenging.

Cloud Heterogeneity - The current space of cloud applications and services is one dominated by heterogeneity, as different cloud providers offer completely different services which are not compatible with each other. No cloud standards or interfaces are currently in place.

In work by Sankaranarayanan et al [116], the authors have proposed a future vision of mobile cloud computing, based on seamless mobile services, that can share data with each other. These services exist on cloud infrastructure, and should be able to share data easily. However, the process of how to share is a question of solving many hard problems which are outlined in the paper. The authors identify some key features of the future of mobile cloud services that will enable their vision, which in turn provide a greater user experience:

Rich User Experience with Mobile Context Awareness - Services and applications should be able to utilise the context of the mobile user to provide more relevant facilities. Contexts such as location, time and activity, can create a richer, more personalised experience for the user.

User Focused Applications - Applications should be designed and delivered to cater to the mobility needs of the user, not to cater to the device.

Sharing of Information - Applications and Services should share their data with each other where possible and appropriate. This would allow a seamless user experience between applications as they converge for personalisation for the user.

As described by Sankaranarayanan et al, the design of software and services is not simply a mobile version of a desktop application or service, the design should focus on the differences between these two platforms, and what the mobile context can offer that the desktop context cannot. The paper proposed a system called COSMOS (Clouddb for Seamless Mobile Services) to realise the authors vision which enables some of the described features. It is powered

by a middleware called SMILE (Sharing Middleware). The paper contains no implementation and no results. However, the scenarios demonstrated from the above features, and discussion of the problems in realising this vision are discussed in detail. The ability of applications and services to make use of context awareness is widely recognised in the literature, and the idea of personal cloud services which may be able to learn from one another by the sharing of data, is a promising research path. In terms of sharing data, enabled by contextual awareness, works by Kazi et al [64] [65] defined a "personal cloud", where users cloud based data and services are delivered consistently and seamlessly across all of the users mobile devices, rather than one device. The authors refer to the one-device mobile cloud computing solution as "device-centric mobile cloud". The seamlessness and sharing across all devices is the key aspect to a personalised mobile cloud experience. In this thesis, when discussing personalisation, the meaning should be taken as personalised service execution in the cloud and personalised results from that execution, both made relevant to the user's situation based on contextual data.

The approaches to confronting some of these problems in the mobile cloud computing domain is now examined in the literature. In addition, in line with the research presented in this thesis, this review also examines:

Mobile Contextual Awareness - Contextual awareness refers to software or hardware that knows about its situation or environment, and will adapt its operation accordingly to be relevant to those circumstances. For mobile devices, contextual information can provide meaningful data about the user, their situation, their preferences, and their activities. This review investigates existing work with mobile contextual awareness, and how this can be used to provide personalisation for mobile cloud applications.

Mobile Web Services/SOA - Mobile web services, or mobile SOA (discussed later) refers to how mobile devices can consume web-based services, or how they can act as service producers, providing services. This review investigates existing work from both these perspectives, and how they can be used to with mobile cloud computing, specifically in terms of service description, discovery, and consumption (execution).

First, the existing common works and approaches aimed at realising the mobile cloud computing paradigm of integrating mobile devices with cloud resources are investigated and evaluated.

2.3 Approaching Integration of Mobile and Cloud

Support for mobile cloud computing is not something that is supported "out-of-the-box" with existing cloud platforms. For example, OpenStack [95] a complete open source cloud, provides components for all features that are required for day-to-day operation and management of VMs deployed as part of a cloud. It does not provide any features or functionality that uniquely support the requirements of mobile devices integrating cloud-based infrastructure or services. Other approaches to cloud, such as Cloud Foundry [42], provide a cloud platform using a PaaS model. With this model, the developer/operator does not interact or directly provision the underlying supporting infrastructure. Cloud Foundry, and other cloud platforms built upon it, such as IBM Bluemix [12], provide a *Mobile Cloud Boilerplate*, which is a cloud-based application that can provide cloud-based storage, Push notification support, and application security to mobile devices, but nothing more in terms of integration. Amazon AWS Mobile Hub [55] provides a similar service to mobile devices as the Mobile Cloud Boilerplate, by provisioning AWS cloud resources to enable data storage, user authentication, analytics, and content delivery to mobile devices, but once again, these are high-level application features, rather than full resource integration.

By formulating full and seamless integration between mobile and cloud as a research problem, and examining the research literature for mobile cloud computing, it is clear that there are four main approaches to integrating mobile devices with cloud resources:

Local Infrastructure - The use of computing infrastructure located near the mobile user to complete resource intensive work. Can include small servers or even other mobile devices; usually one Wi-Hi hop away.

Offloading and Partitioning - Mobile applications have come of their codebase offloaded to a cloud server for execution with results returned to the mobile device, or the application is partitioned up into components which are distributed to devices in the vicinity for execution.

Remote Viewing - Essentially, existing remote display technologies found in desktop computers, applied to mobile devices. An operating system runs on a cloud server, and the user interacts with the user interface remotely from their mobile device.

Middlewares - Mobile cloud middleware solutions are developed to provide cloud-based services to mobile devices. The middleware is often deployed in a cloud, with a corresponding thin client application installed on the mobile device, to communicate with the middleware. Services provided by the middleware often utilise existing web service technologies in line with Service Oriented Architecture (SOA). The research presented later in this thesis takes this approach, with the aim of overcoming the existing issues in these related works, discussed shortly.

Existing research literature from each of the four areas outlined are now reviewed in detail. They are evaluated from the user experience perspective, against the challenges/difficulties outlined in the previous section. Finally, this section briefly looks at some of the research literature in the area of resource management considerations for mobile cloud computing. The use of the scarce resources of energy and bandwidth are a crucial investigation point for the mobile cloud computing solutions presented in the following subsections, as well as how these solutions make use of the server-side computing resources available to them on the cloud infrastructure.

2.3.1 Local Infrastructure

By far the most common approach to using local infrastructure as a solution to mobile cloud computing comes in the form of *Cloudlets*, proposed by Satyanarayanan et al [117]. This solution involves placing self-managing cloud infrastructure in locations near where the user is likely to access cloud services from the mobile device, for example, near the Wi-Fi access point in a coffee shop. The Cloudlet could be integrated with a wireless access point. Not intended to replace large cloud infrastructure in large data centres, the local infrastructure would be minimal. The hardware runs virtualisation software intended to run virtual machines owned by a small number of users. The Cloudlet possesses a "base-VM", which can be modified into a full VM, using what is known as an overlay-VM, by a process known as dynamic VM synthesis. The base VM stores the base operating system such as Microsoft Windows, and the overlay contains the users applications, profile, and settings. The overlay is stored on the mobile device, and amounts to roughly 100MB in size. When the mobile user enters the vicinity of the cloudlet, the overlay is sent to the base VM on the cloudlet, to create the full VM. The Cloudlet per-

forms some work for the user, while more complex work (presumably with no real-time requirement), is forwarded to a remote cloud data center.

Cloudlets aim to solve the latency issue in mobile cloud computing. As the Cloudlet should typically be located only one Wi-Fi hop away from the mobile device, the latency would be negligible. Real time applications could then run on the Cloudlet with results received at the mobile device quickly. They also bring in a level of personalisation, allowing the users to have their own custom applications and data stored in the VM-overlay. Drawbacks to this approach include Cloudlets not being available near the vicinity of the mobile device at all times. Additionally, tests show that the time to fully synthesise the VM on the Cloudlet takes roughly between fifty and sixty seconds. However this performance is expected to improve in the future with advances in WLAN bandwidth, and further optimisations to the dynamic VM synthesis process. Bandwidth may also pose an issue when multiple users are using the same network to access the Cloudlet, especially, as described in the paper, if the Cloudlet is co-located with an ordinary Wi-Fi access point. This approach also faces other significant mobility issues; graceful disconnection, and saving of session state if a user moves away from the Cloudlet, are not discussed.

An OpenStack based implementation of Cloudlets, known as OpenStack++ (OpenStack plus extensions for mobile cloud), has been proposed by Satyanarayanan et al [118], and implemented by Ha and Satyanarayanan [48]. The OpenStack++ cloud deployed in the remote cloud data center, receiving the complex work offloaded from the Cloudlets, would work with services already deployed in the cloud, but the same mobility concerns remain in terms of mobile device access to the local Cloudlets.

Results have shown that optimisations of the Cloudlet approach have been somewhat successful. Verbelen et al [141] modified the definition of the Cloudlet, and developed a minimal dynamic Cloudlet architecture that makes use of any device (such as smartphone or laptop) on a WLAN with available resources, by breaking an application up into several components and distributing these components to the devices. Here several devices in close proximity form the Cloudlet. They communicate via a Node Agent running on each device. The node agent manages Execution Environments running on the devices, which manages the executions of the application components. A Cloudlet Agent, which typically runs on the device with the most available resources in the Cloudlet, manages the Cloudlet and communicates with the

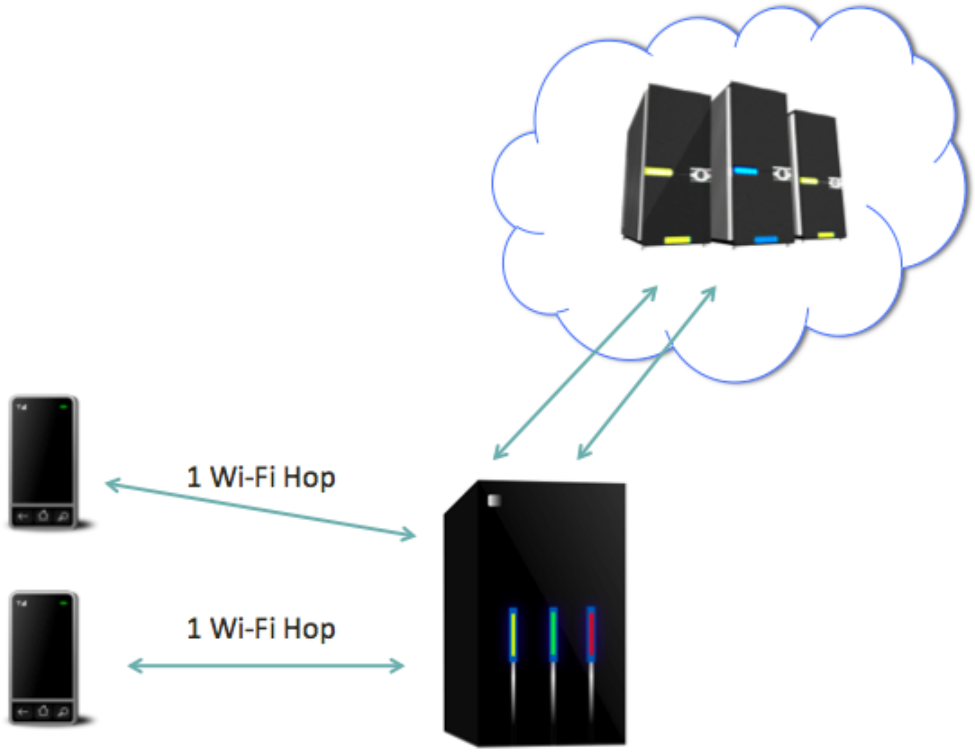


Figure 2.1: Cloudlet Model. Mobile devices are located one Wi-Fi hop away from the Cloudlet, which may be co-located with an access point or router in a public area such as an airport or cafe. A user's custom VM-overlay on his/her mobile device is combined with a base VM on the Cloudlet to form a full VM. Complex computation is forwarded to remote cloud infrastructure.

Node Agent running in each device to distribute components. With this approach, timing performance has been shown in some cases to be better than running an entire application on one VM, in certain configurations. The components can even have specified constraints such as time for completion of processing in individual components. The Cloudlet Agent can then deploy components to devices in the Cloudlet depending on how likely a device with its available resources is able to complete the components work within the constraint.

The use-case carried out in this work for testing was an augmented reality application. While this approach generated some improvements in performance for some applications where the constraints were met, other aspects of the use-case saw performance degradation where the constraint was not met, and performance difference was negligible in other measurements. The augmented reality application was developed specifically for this Cloudlet component based architecture, and so other existing applications may also have to be modified for optimal performance on this Cloudlet based architecture as well. R-OSGI

was used to develop the application as a collection of components. The approach is very similar to that of Giurgiu et al [45], which is discussed in section 2.3.2. An issue exists here in the case of the user allowing their device on the WLAN to be part of the Cloudlet. The user of the smartphone with limited battery power, who may not be making the same use of the Cloudlet for tasks as other users, may not want their device to use up battery power for tasks from other people, if it is barely enough for the device's own resources and tasks. Users need to be incentivised to participate.

A common feature exists in the Cloudlet approach with those highlighted in [124] and [24]. This is the use of virtual machines. The benefits of a local Cloudlet infrastructure here are consistent with [124] in recognising the need for cloud infrastructure to be near the mobile device to minimise latency from RTT. It can be concluded that Cloudlets can be a very viable solution to the network latency problem, becoming an enabler of real time applications. However, the overhead of a VM based solution is a large price to pay for simple applications. For example, as shown by experiments conducted by Clinch et al [20], the distance, and resulting RTT and latency between cloud infrastructure and many mobile applications, has little impact on the performance of these applications. As such, the complexity of a Cloudlet approach may not be necessary in many cases. In more recent work by Flores et al [37], it was concluded that that few mobile cloud applications exist that are as resource heavy as augmented reality applications, again drawing the conclusion that this approach will result in large overhead for most mobile cloud applications. Virtual machines running in the cloud are also not suitable to utilise context information from the mobile device.

Cloudlets, or similar server infrastructures, are said to be *network edge* solutions. This is because the infrastructure is located at the edge of the network closer to host devices such as mobiles, rather than existing in the *network core* closer to data-centres. The European Telecommunications Standards Institute (ETSI) have proposed the Mobile Edge Computing (MEC) initiative [34], a standard in which mobile edge computing infrastructure will host and provide services that can be utilised by mobile devices. The motivations behind the standard are the same as the goals of the local infrastructure approaches - low latency, high-bandwidth connectivity between the mobile device and the mobile edge infrastructure. The work is considered to be a driver of 5G network connectivity, and if implemented and widely deployed, could avoid the overheads and shortcomings identified with existing local infrastructure

works.

One work which also benefits from local infrastructure, which directly tackles the disconnection problems unlike Cloudlets, is the M2C2 framework by Mitra et al [89]. M2C2 is described as a mobility management system. As the mobile user moves through different networks, such as Wi-Fi and 3G, and additionally, between different basestations while on a cellular network, the user naturally experiences small periods of disconnection. The M2C2 system determines two different choices; one, the choice of best network access interface, based on network conditions, and secondly, the best cloud to use, based also on network conditions, and further cloud VM metrics such as CPU utilisation (ideally, looking for a VM with low CPU load, for example). As the user moves through heterogeneous access networks (HANs), the system uses a probing technique, making calls to RESTful APIs, to determine the quality of each network interface. After the interface has been decided, it probes different clouds running the user application (the evaluation experiments were performed with an augmented reality application, which receives location and activity context information from the mobile device), and determines, based on the VM metrics, which cloud is most likely to meet specified Quality of Service (QoS) guarantees, and chooses that cloud. The work supports local clouds on the same access network, and public clouds (such as Amazon EC2). Their results point to the same guiding principle of the Cloudlet approach, that clouds near the user are essential for applications with low-latency requirements. Finally, the solution also tackles the disconnection problem by supporting multi-homing, where a device can be connected to two access networks at once; this is for a short time only, while the mobile is moving to a new network, and has cloud connectivity restored to the new access network. The authors measure of success in this regard is minimal packet loss, which does not have any bearing on the performance of the augmented reality application, as it runs during a handover (and the mobile is connected to two networks at once). However, this application works by making hundreds (or thousands as in one experiment) of sequential RESTful API calls, which would be considered a very costly operation in mobile cloud computing. No analysis is performed to determine if the benefits of low latency, and choice of low utilisation clouds, has any benefit over the cost of running this algorithm.

Nishio et al [92] take the approach of defining mobile devices in the vicinity, as part of a cloud, similar to the work by Verbelen et al. Here, the authors refer to a cloud of mobile devices as a "local cloud", managed by one elected mobile

node, known as the local resource coordinator (LRC). Mobile devices wishing to complete work (which is defined here as a task requiring resources, which are communications and computation services), request these resources from the local resource coordinator, who then decides which mobile devices in the local cloud can provide these resources as services. These mobile devices then communicate directly. Devices can request resources from others (through the LRC), and share their own as services. This brings again the question of incentives for mobile devices to share their constrained resources. This work mathematically models the costs for communication and computation sharing, as well as optimisation models for task executions. This work is one of few which aims to settle the question of incentives, by means of a fairness measure. Resource usage is modelled as service latency, a time based measure, from when a task starts, to when it is finished. The more a device contributes its resources (depending on if it has resources to spare, which is also considered by the model), the task latency is reduced, and the greater a device is rewarded under the model. How fairness and rewards are defined is not made explicitly clear, except to assume that a device that shares resources can also share execution of its own tasks with other devices.

Some approaches have attempted to relieve some devices from having to use WLAN or remote cloud infrastructure, and instead use information from other devices in close proximity, one of which may have an active Internet connection to access cloud resources. This formation would be similar to a multi-hop ad-hoc network (MANET). These devices can create an ad-hoc mobile cloud. One such approach is by Huerta-Canepa and Lee [56]. They call this a "Virtual Cloud Provider". The idea behind this approach is that users who share the same environment may be interested in performing the same tasks. The example given was that of users attempting to translate a Korean text description of a museum piece. Due to roaming charges, the user does not want to access cloud services, but he/she can obtain the same information from another mobile device in close proximity, who already has this information. An ad-hoc network is created between the devices to obtain the information. The implementation involves intercepting application calls to cloud infrastructure, and modifying them to use the virtual cloud provider. It also determines if devices nearby are stable (not moving away out of the vicinity), and resource availability to determine if a task needs to be offloaded to another device. The current implementation yielded results that were slightly slower for tasks than when carrying out the entire task locally without any offloading, although the

majority of the taken time was in the process of processing the tasks to be sent to the virtual cloud providers.

For this approach to work, a device needs to be close to other devices doing the same kind of work. In many cases this is feasible such as in the example given, but in other areas it may not be so. This approach again also brings in the uncertainty as to a user willing to sacrifice some of their own battery power for the tasks of another user in the vicinity. Not only does another device nearby have to be available at the same time, but for the approach to be mutually fair, devices have to be working on the same task with the same information, at the same time.

To summarise, the use of local infrastructure is an approach to mobile cloud computing, where a mobile device uses computing devices nearby to complete work. This can include the Cloudlet approach, with a nearby server, or the modified Cloudlet, which is made up of several devices in the WLAN. This is ideal for applications with real-time, low latency requirements, which will be required for the user experience in terms of performance perception, but the infrastructure is costly for applications not as resource intensive as the common use-case of augmented reality. Bandwidth and mobility concerns remain, as do incentives for user participation when utilising the devices of other users. As such, Cloudlets at the present time, cannot be considered as a solution for an integrated user experience.

2.3.2 Offloading and Partitioning

The partition and offloading of application code to be executed on cloud infrastructure is addressed heavily in the literature as an approach to integrate mobile applications with the cloud. Not only is it intended to save the battery power in the device by offloading execution, but code may execute quicker in the cloud, given the greater CPU and memory resources there compared to the mobile device. Application code that is suitable to be offloaded is moved from the mobile device onto the infrastructure, executed, and then the results are returned to the device, where execution continues.

2.3.2.1 Application Partitioning

The model of offloading to multiple devices is similar to the version of a Cloudlet given by Verbelen et al [141], and that of the local cloud by Nishio et al [92].

Alfredo by Giurgiu et al [45] is an application partitioner. The aim of the project is to move computation intensive components of an application onto cloud infrastructure. They developed this component system on top of R-OSGI. By designing applications into movable (or possibly non-movable) components, they could be moved between the mobile device and the infrastructure with the aim of maximising or minimising an objective function, such as minimising interaction latency. Alfredo profiles an application into a consumption graph, where the vertices represent the application components known as bundles, and the edges between bundles represent a one-to-one dependency. Various annotations are used on the graph to describe attributes such as data size, energy or time constraints. Various configurations of bundle deployments are calculated from this graph based on the objective function. They also demonstrate two novel algorithms used to decide the deployment configuration from the graph. They tested the implementation with a new application designed for the Alfredo architecture, and an existing application they modified for the architecture. The results showed the algorithms were optimal in deciding configurations for minimising latency. The results of application performance varied based on the configuration chosen. The authors concluded by suggesting a new approach to mobile applications; rather than a single application running on the mobile device or accessing software through a web browser, that a third approach could be to use the mobile device as an application controller. It will be responsible for providing the interface, and the other components are delegated to the cloud. This is similar to the remote display/thin client solution work by Simoens et al [124], which is discussed in subsection 2.3.3.

The benefits of partitioning the applications in such a manner are clear; ultimately it enabled the applications that were too resource intensive to run locally on the mobile device to be used. The example shown to minimise interaction latency again is desirable from the user experience point of view, as is it's capability to calculate deployments to minimise energy costs. It may be worth highlighting that minimising latency and maximising energy savings may not result in the same deployment. Both are important for the user experience. In this case, a deployment that saves energy, may increase latency,

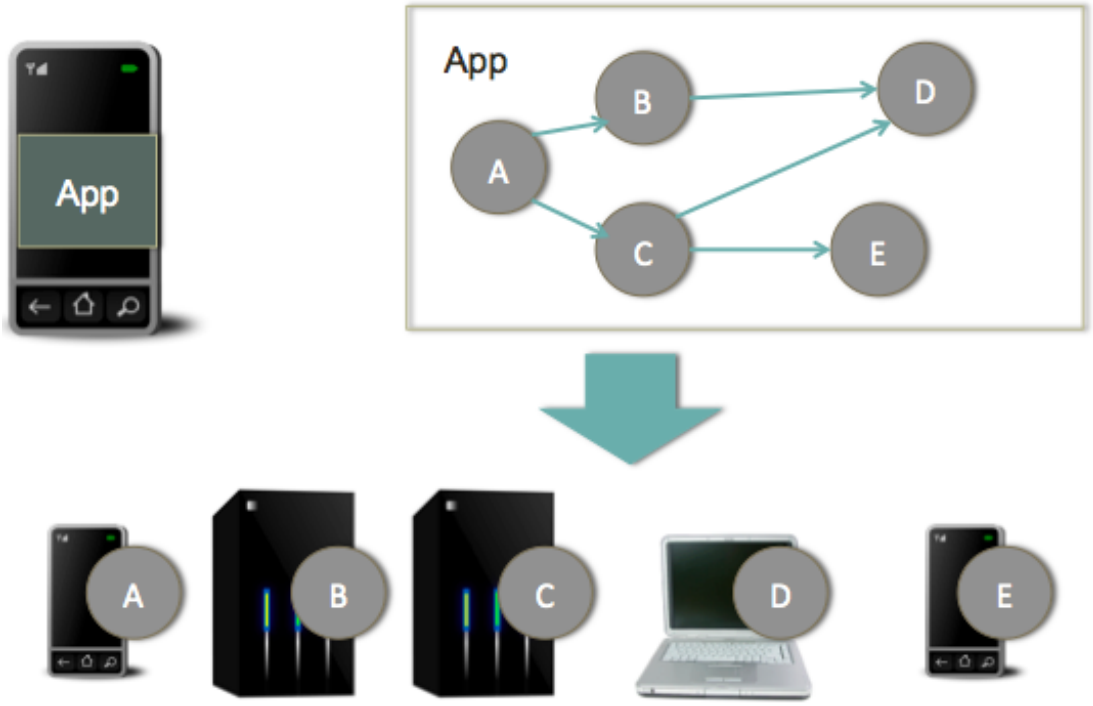


Figure 2.2: Application Partitioning Model. Mobile applications are developed as separate components forming a graph. A cut is then made in the graph to form a distribution of components for execution on other devices located in the vicinity. The cut specifically is made to meet an objective function, such as minimise energy consumption.

and vice versa. No tests were carried out in the work on that note, nor would it be decisive as the results would vary for different applications. The experimentation used was between a tablet device and a laptop computer; no cloud infrastructure is utilised, which would be useful, as deployments could also take place between different cloud nodes. The approach as a whole is very similar to the dynamic Cloudlet solution by Verbelen et al, such as it requires applications to be built as distributable components, and existing applications would need to be modified for this.

One issue not taken into account by the original Alfredo partitioner is the fact that the network environment is consistently changing. Therefore, at one time, a partitioning may be the most optimal, but a later time, the partitioning may no longer be optimal. In this case, an element of "dynamic-ness" must be introduced to partitioning. An extension to the Alfredo framework in later work by Giurgiu et al [44] builds on the previous work by the authors to introduce dynamic partitioning. This kind of partitioning results in the deployment of application components to local nodes changing at run-time, based on online

calculations for optimal re-deployments, as network and device conditions change. The authors describe the first version of Alfredo, as a 'static partitioning" model, in that after the initial calculation of the application component graph and deployment cut, the deployment configuration never changes. Once again the work is based on OSGI bundles, and so applications must be adapted to the OSGI framework. The results showed the value of dynamic partitioning over the original work, with performance increase up to 75%, and power consumption improved by up to 46% (on Wi-Fi), depending on the specific application.

Similar to the second version of Alfredo, the MoSeC framework by Wu and Huang [146] considers the extension of the cloud to an ever-changing network edge consisting of many mobile devices and sensors; a dynamic configuration as Giurgiu et al would describe it. Instead of the traditional mobile cloud offloading model of making one decision to offload to one server, this paper presents a model for the topology reconfiguration problem. In this problem, a mobile device, at different decision points, may offload several application components, to various different sites/nodes. This process repeats several times for the changing environment. The paper proposes a system for monitoring network state, and then applies one of three different algorithms to reconfigure the offloaded application components, to match an updated network topology, using resources such as energy saving as a reward for reconfiguration. MapReduce algorithms are used to calculate the reconfiguration at these decision points. The work formulates the reconfiguration problem for use with the algorithms. No implementation or evaluation has taken place; the model is purely theoretical.

One example of work, which still takes an OSGI-based approach, yet tries to remove the burden of partitioning considerations from the mobile application developers, is the AIOLOS middleware by Bohez et al [13]. Specifically, based on OSGI, it takes mobile applications developed as a set of components, and sends these components to AIOLOS nodes, which consist of various mobile, cloud edge (Cloudlet) or cloud data centre VMs, for execution (where each runs AIOLOS). The focus of this work was the scalability of the AIOLOS nodes to support multi-user applications. AIOLOS can monitor the resource usage, and based on policies, re-configure the deployment of components to match a goal, such as minimise execution time or maximise energy savings, again, as seen with Alfredo. This reconfiguration of the deployments is similar to the second version of Alfredo, and the topology reconfiguration problem defined

with the MoSeC framework. The other aim of this work was to separate the developer burden of having to develop the application to handle execution of these different kinds of nodes, by making the aspects of remote or local execution, transparent to the developer. This is important, as other code offload approaches such as MAUI and CloneCloud, as discussed in the next subsection, suffer greatly from developers having to modify the applications for remote mobile cloud execution.

To summarise application partitioning, this technique has demonstrated the benefits of performance improvements, and the realisation of applications that are too resource intensive for mobile devices. The major difficulty here is the requirements shared by most of these frameworks, that the applications must be developed as components for the OSGI framework (or OSGI bundles to be precise). The argument of the authors of these works, is that it is good programming practice to develop applications in this way, coding to interfaces, and loosely coupled, which is an agreement shared by software developers in general. However, more often than not, this is unlikely to be the reality, requiring extensive application modification to support OSGI bundles. Aside, there is also the issue of devices being nearby and willing to participate, similar to the concerns with the Cloudlets in the previous subsection. Finally, the cost to the mobile device of profiling the network repeatedly, calculating graph cuts, and component deployment, is non-negligible. Adding this cost on is not beneficial to the user experience because of the resulting power cost. Arguably if this profiling was cheap, this would be an excellent approach from the user experience perspective, but troublesome for the developer experience.

2.3.2.2 Code Offloading

Code offloading solutions offload specific marked parts of application code, rather than application components. A first example of this kind of solution is the MAUI platform by Cuervo et al [21]. MAUI will offload application code at the method level, marked by developers as "Remoteable". This works at the VM level as part of the Microsoft .Net Common Language Runtime (CLR). Not all methods are suitable for offload, such as code that accesses the resources on the device such as the sensors, the camera, or native code. It is up to the developer to mark suitable methods themselves according to the given constraints. All object state within the class containing the Remoteable method on the heap is serialised and offloaded with the method code. Code is executed on

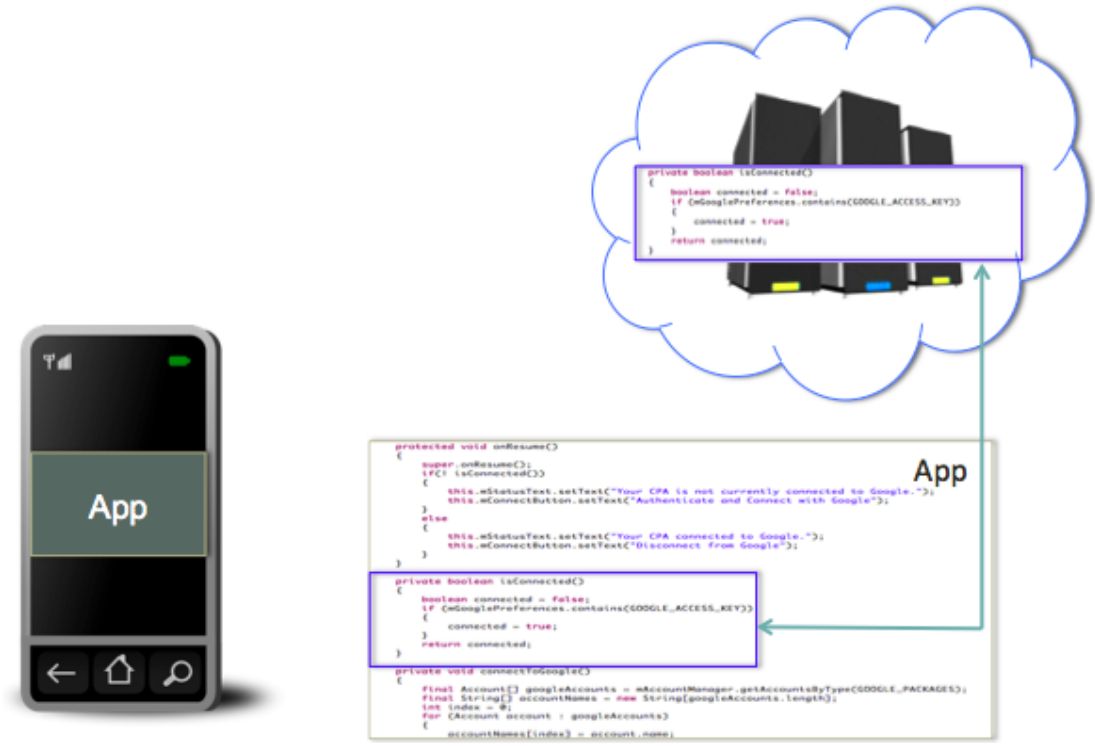


Figure 2.3: Code Offloading Model. Specific parts of mobile application code, normally methods, are offloaded to a cloud server for execution. Results, in the form of changed object state, newly created objects, and destroyed objects, are returned to the mobile device upon completion.

the MAUI platform on the cloud infrastructure, and the resulting state changes are returned to the device, at which point execution resumes. If offloaded code tries to call native code however, execution must be suspended and returned to the device before continuing.

The architecture of MAUI is similar to other code offloading approaches. MAUI carries out various profiling tasks to determine if code should be offloaded or not. MAUI is designed with the goal of saving energy. If energy can be saved by carrying out code offload, then it should take place. The MAUI profiler estimates the energy required to execute a Remoteable method on the mobile device by measuring CPU cycles (assisted by past invocations of the method). At runtime, by profiling the network quality, it estimates the energy cost of offloading the device to the cloud. This profiling data is given to an optimisation solver, to then determine should the code be offloaded. The profiling of the network and the solver run constantly to provide the latest estimates.

MAUI showed great potential for energy savings on the mobile device. In favourable network conditions, always on Wi-Fi, it resulted in significant en-

ergy savings, yet in tests on 3G networks, no energy savings were found. Infact, the optimisation solver always refused to carry out code offloading on 3G networks in all experiments, based on the profiling results, instead opting for local execution. This is a significant problem, as users on the move through cellular networks in a public area would not be able to take advantage of MAUI. It also suffers from the fact it can only offload developer written methods at the VM level. In some cases this makes sense, such as methods accessing the camera, but native code which does not access the physical resources cannot be offloaded. In terms of disconnection, it is handled, but not gracefully. If a timeout period elapses for a return from an offloaded method, MAUI on the mobile device will assume the network connection is lost (or very poor), and will re-execute the offloaded code locally. This is a significant waste of time and energy. Another non-negligible concern is the profiling overhead from determining the network status, and optimisation solver overhead, just as with the application partitioning approaches.

Another example of code offloading by Chun et al [19] is CloneCloud. Like MAUI, profiling takes place and an optimisation solver will decide if code should be offloaded at runtime. Unlike in MAUI, CloneCloud offloads on a method basis, but at the thread level of the VM. The CloneCloud application on the cloud infrastructure is a VM running a clone of the mobile device software, meaning it cannot offload native OS methods. However, offloaded developer methods can call native code, without execution returning to the device. CloneCloud does not require any developer intervention; code does not need to be annotated as Remoteable. CloneCloud can also offload based on saving time, not just energy. Static profiling, which is run once, estimates the time and energy required to execute methods. Suitable offloading points according to constraints are detected and stored in a file during this process. At runtime, a partition of code offload points are given to the CloneCloud software on the mobile device, and by dynamically profiling the network quality, an optimisation solver decides to offload if time or energy is saved. In terms of the experiments, CloneCloud was only evaluated on Wi-Fi networks, and no consideration was given to cellular networks, which of course is of critical importance. Once again, the profiling overhead is a concern for the user experience.

Mobile Augmentation Cloud Services (MACS) by Kovachev et al [71] also performs code offload. MACS requires the developer to write all code that may be resource intensive into an Android service; the service can then be offloaded

at runtime in suitable conditions based on network and resource availability profiling. The services are presented to the rest of the application (UI activities etc.) in the Android Interface Definition Language (AIDL), so the code that calls a service to perform an intensive task has no knowledge of the code possibly being offloaded. Like MAUI and CloneCloud, it uses an optimisation problem to decide to offload, based on minimising an objective cost function, such as time or energy savings. MACS also has this common weakness of not being able to offload native code or code that accesses physical resources such as the camera. Results showed tremendously high speedup as their test applications, such as solving the N-Queens problem, grew in complexity for greater values of N. However for small values of N, it was found that code offload introduced more overhead than local execution. This project shares the same weaknesses as CloneCloud, in that the authors did not test MACS on a cellular network, assuming (as MAUI showed), that results for the cellular network would be poor due to lower bandwidth.

To illustrate the overhead of offloading techniques, Ma and Wang [86] also developed a code offload approach, which uses a technique called stack-on-demand execution. This technique was specifically used to address the overhead of offloading possibly unnecessary data, and the overhead involved in other code offload approaches, even when offloading does not take place. Stack-on-demand, rather than offloading all state within a method's class such as in MAUI, will only offload the stack frame on top of the call stack. Therefore, only the state required for a given execution context (such as a method) is offloaded. It also aimed to reduce the overhead in cases where offloading does not take place, by keeping two sets of methods, one with offloading code and one without. Rather than having offloading code placed at, for example, the method level, it delegated offloading code to exception handling code. So if for example a required library is not found on the device, an exception is thrown, and offloading can then be signalled to take place. The authors found great savings using these techniques when offloading does not take place. Unlike in other works, time and energy considerations for offload are not considered, offload is either requested by an application or a condition for offload such as a missing library is met.

The authors provide an implementation of this concept in [87], known as eX-Cloud. The implementation aims at not just migrating from mobile device to cloud, but between cloud nodes, emphasising application parallelism with the Message Passing Interface (MPI) on cloud infrastructure, so the focus is some-

what split. The results showed offload time for their sample applications between two hundred and four hundred milliseconds. Overhead resulted from the state capturing time like in the other approaches, even with the stack on demand technique. The authors did not carry out any results comparison with the related work like MAUI and CloneCloud.

The Uniport programming support framework by Yuan et al [148] can be considered as another code offloading project. Uniport is designed to allow developers to create cross platform mobile applications with ease, following the Model View Controller (MVC) design pattern. This pattern is used extensively in web applications, and in particular, with the iOS platform. Briefly, the Model contains the data and associated data logic of the application. The View is the presentation layer, which presents the data in the Model, as part of a GUI. The Controller contains the overall application business, event, and flow logic, and passes the data from the Model to the View. The goal behind the Uniport framework, is that a mobile cloud application (MCA), should execute expensive operations on the Model, and the Model should be offloaded to the cloud server for execution. The Uniport client synchronises the local data Model on the mobile device, with the Uniport server application. The Uniport client can send an execute call to the server when required, to start the expensive Model operations. After completion, the updated Model is returned back to the mobile device, and the Controller passes the updated Model data to the View. The authors also developed a code generator, to generate projects stubs with skeleton code for the different mobile and server platforms, along with an analyser, which aims to assist the developer in converting existing mobile applications, into the Uniport MVC architecture, which is a considerable advantage for this project, considering the lack of support for existing applications to be converted for other code offload and application partitioning approaches. The evaluation goal of the framework was response time and energy savings for various implementations of Gomoku, a board-based strategy game, for each mobile platform (iOS, Android, and Windows Phone), which the evaluation demonstrated. Like MAUI, this approach also deals with disconnection based on time-outs, and will re-execute offloaded and synchronised Models locally.

A final example of a code offloading approach for mobile cloud computing, for comparison with MAUI and CloneCloud, is the ThinkAir framework by Kosta et al [70] It bears similarities to MAUI, in that it requires developers to annotate code at the method level, that is suitable for consideration for remote execution. It is also similar to CloneCloud, in that it uses a copy of the mo-

bile operating system located in the cloud (minus some features such as GUI libraries). According to the authors, it improves on CloneCloud, by having an online profiling engine, compared to CloneCloud's approach, where offloading plans must be generated from an application graph during the development phase. ThinkAir uses profilers to monitor network, program, and device conditions/factors, and an Execution Controller then determines if there is an energy or time benefit to the code being offloaded over Wi-Fi and 3G interfaces (consider that the CloneCloud work did not evaluate 3G for offloading). A big focus in ThinkAir for the authors was to utilise cloud scalability for parallelism similar to exClouds goal; the cloud part of the ThinkAir framework can startup or unpause existing VMs, and delegate offloaded work to them to exploit parallel computation. In the results, offloading computationally expensive applications (an image combiner, a virus scanner, and an N-Queens puzzle) improved energy consumption and execution time to a point (8 VMs), at which time the overhead of managing that number of VMs started outweighing the benefits. Similar to MAUI and CloneCloud, ThinkAir carries the cost of having to profile network conditions, along with program and device conditions, in preparation for an offload. The paper also does not discuss what code may or may not be suitable for offloading, in the same way MAUI does, but the authors do mention that their Execution Controller will make the decision on behalf of the developer, so that there will be no errors should a novice developer make a mistake.

Another work, not aimed directly at solving the problems of mobile cloud computing, but similar to both code offloading and application partitioning projects, is the Sapphire framework by Zhang et al [149]. The Sapphire framework is designed to ease and separate the deployment considerations for applications, that can run on both mobile and cloud platforms. The authors recognise a need for users to execute some applications, both locally on mobile devices, and remotely on cloud infrastructure. The authors argue that coding the deployment logic to handle different deployment configurations (mobile, or cloud, or even both), can be difficult for the developer. The Sapphire framework allows a developer to build an application on-top of Sapphire. Sapphire will then take care of handling deployment configurations and features, on behalf of the developer. The Sapphire system features a Deployment Kernel at it's bottom layer. This is a runtime to provide deployment support to the developer. On top of this, the developer uses extensible Deployment Managers, to describe the features they want for deployment. For example, one built in

feature is the CodeOffload deployment manager. The developer creates a Sapphire object, containing the work to be remotely executed. A SapphireObject is an executable object that has its deployment managed by Sapphire, and the object can be executed on mobile, cloud, or even both platforms. In this example, this object, which uses the CodeOffload Deployment Manager, will be moved from the mobile device, to the cloud, for execution. The mobile application is transparent to this; any calls to the code in this Sapphire Object, are made transparently by Sapphire to the cloud, with Remote Procedure Call (RPC). The system features other DeploymentManagers for other required features of application deployment, such as caching, replication, and load-scaling. This work does to a small extent address the disconnection problems that interfere with MAUI and CloneCloud, in that if a device becomes disconnected, the Deployment Kernel on that device continues to run, but cannot make RPC calls. The authors state their expectation that devices are connected to the internet most of the time, which in the mobile cloud computing paradigm, should not be the expectation.

To summarise code offloading, it can be seen from the works discussed in this subsection that code offload is capable of bringing dramatic speedup and energy savings for intensive work, but only in favourable network conditions, and when the amount of work is large enough such that the overhead of offloading is made somewhat negligible by comparison with the overall cost. To the mobile user it can in some cases solve the battery power constraint, and even enable tasks that would not be possible on the mobile device alone [21]. However, in the absence of a suitable quality network it does not solve this problem. MAUI and Sapphire were the only works to discuss what should happen if the device is disconnected while code is offloaded. Any solution therefore needs to weigh up the amount of work and quality of the network before deciding to offload, and tackle the overhead of offloading such as in [86]. As with application partitioning, network profiling costs for the optimisation solver, are non-negligible. Aside from poor performance on cellular networks and the profiling overhead, collectively, application partitioning and code offload solutions meet more of the user experience goals outlined than the Cloudlet-based projects; however these approaches for all practical still do not solve the mobility aspects of disconnection.



Figure 2.4: Remote Display Model. Similar to Cloudlets and application partitioning, a virtual machine runs on a cloud server. The component of the solution that displays the visual GUI output of the virtual machine runs on the mobile device.

2.3.3 Remote Display for Virtualised Desktops

In the work by Simoens et al [124] the aim is to use the mobile device as a remote display, or viewer, for full operating system virtual machines running on cloud infrastructure, which can in turn run user applications. This is similar to the application partitioning approach by Giurgiu et al [45], where the mobile device provides the interface for the remote computation intensive application which runs on infrastructure (in terms of Alfredo, where one component would be the user interface running locally, and the other components with computation would run remotely). The article presents various research which provides solutions to these challenges, such as code offloading and local infrastructure. The authors conclude by stating that to address the challenges, several different solutions they present need to utilised, in such a way that different solutions may be more appropriate to different contexts.

To demonstrate this, Deboosere et al [24] present a thin client solution for accessing virtualised desktops in cloud infrastructure. The architecture contains a service manager with a self management component, that implements var-

ious algorithms for optimising the architecture for access by thin clients running on mobile devices. These algorithms are designed to optimise not only the user experience on the thin client, but to reduce the costs of running the cloud based resources for cloud providers. To enhance the user experience, approaches are used to address the latency concerns - RTT, as well as the audio and visual output, such as again emphasising the need for local infrastructure, and appropriate choice of codec. Various other features include resource allocation determined by an overbooking degree.

While it may be useful to access operating systems running on virtual machines, the size of the screen on the device will always be an inhibitor to it's success. Windows or Linux platforms running on the cloud do not have optimised audio or visual output for mobile devices, they are designed for large screen monitors on laptops or desktops. Also consider, as outlined in the work by Simoens et al, that this approach will require continuous data transfer, for the mobile device to remain up to date with the visual GUI output of the operating system, even if nothing has changed appearance on the GUI, making the approach extremely costly. One of the optimisations proposed by Deboosere et al is that some intelligent transfer should take place; the remote server should detect if the GUI has changed, and only then, send the visual data back to the mobile device. If one looks at operating systems such as iOS on Apple iDevices, Android by Google, or Windows 8, these operating systems have been specifically designed to run on mobile devices with small touch screens, and access to device sensors for context information. iOS is built from the same kernel as OSX, Apple's operating system for laptop and desktop devices. Android is built on a Linux kernel. Windows 8 is the first operating system to be designed for both mobile devices and desktop or laptop computers. It contains a touch interface for mobile devices, and a traditional desktop point-and-click interface for laptop and desktop computers. It may be worth nothing that mobile computing only gained widespread public appeal when Apple released the iPhone, running the iOS software (however it was not known as iOS at that time). Before that, several tablet solutions existed which ran a complete Windows operating system. These met with little success.

With the advent of mobile devices containing various sensors, the potential for cloud services and the device itself to utilise context information, making them context aware, is possible. Virtualised desktop solutions will not take advantage of this. The literature in this path of research such as [124] and [24] does not make any mention of the operating systems running in the cloud

taking sensor information from the device; only touch events to interact with the operating system are taken. Ideally, a solution should be able to make complete use of all the features located on the mobile device. Cloudlets face the same situation, as they are complete operating system solutions as well, and again, will not take advantage of unique context information from mobile devices. From the user experience perspective, remote display solutions could be by far the most costly approach, and will suffer from every issue outlined at the beginning of this section.

2.3.4 Mobile Cloud Middlewares

Some of the solutions for various difficulties of mobile cloud computing can be resolved by the placing of a middleware solution between the mobile device and the cloud itself. This acts as a proxy; requests and responses to and from cloud infrastructure pass through the middleware where some additional work can take place, or features can be provided. This subsection will examine some middleware solutions for mobile cloud computing. Many middlewares use web services corresponding to service oriented architecture to complete work, whereas others do not. Here, examples of both categories are presented.

2.3.4.1 Middleware Solutions with Service Oriented Architecture (SOA)

Web services are a standard method to provide data and communication facilities to clients in the area of service oriented architecture (SOA). Web applications or local applications can connect to web services using HTTP, by sending requests to endpoints, which are simply URL endpoints for sending requests to the service and receiving responses. Clients can pass information and parameters to the service to customise the request for the information or service they need. The response typically contains result information that the application can process in some way and display to the user within the application.

Typically, there are two kinds of web service, a SOAP (Simple Object Access Protocol) service, or a RESTful (Representational State Transfer) service. SOAP services receive SOAP messages from the client in the body of a HTTP request, which consists of a SOAP body, contained within a SOAP envelope. A SOAP response is sent back to the client in the body of the HTTP response. SOAP messages are in XML format.

RESTful services also work by sending and receiving HTTP requests and responses, however REST identifies a service by the URL endpoint, and the specific HTTP method of the request such as GET or PUT. GET messages are normally used with RESTful services which request information from the server, and PUT messages are used to pass information to the server. Parameters can be passed in both the URL of the endpoint, and within the body of the HTTP message, using either XML, or more commonly, JSON markup. JSON has been seen as more "lightweight" than XML, as it does not contain opening and closing tags for each piece of information. This is an important consideration, not just to minimise the size of the data being transferred for the sake of the network traffic and associated delays, but in the mobile context, where the user is often charged per kilobyte of information sent and received. In work by Jason H. Christensen [18], it is proposed that RESTful architectures are a practical means to deliver applications and services to mobile devices, when considering data exchange requirements and constraints.

As web services are such a common standard for communication of information and requests, cloud providers provide public web services to clients to communicate with their respective services [6]. However, these services are currently not interoperable. Developers have to use a specific API for each cloud services provider, which are generally not compatible with other providers. Moreover, the services are not developed with the mobile context in mind, and may contain unnecessary overhead, or the data may be in a format that is expensive to parse. In recognition that the current state of cloud computing access is not completely adequate for consumption of services from mobile devices, the need to have some form of middleware-based proxy, situated in the cloud between the mobile device and cloud services, is recognised in the literature.

Wang and Deters [144] developed an SOA Mashup middleware to be deployed on cloud infrastructure that is capable of invoking RESTful and SOAP-based services. As the XML based SOAP messages are larger in size than RESTful services that communicate JSON, they are more costly for a mobile device to parse. The middleware takes web service requests from the mobile device, in a RESTful style, using JSON. If the service being invoked is SOAP based, the middleware transforms the request into a SOAP message, and invokes the web service on behalf of the device. The result is then stored in the middleware until the device is ready to receive it, a form of results caching. The SOAP response message from the web service is converted into JSON for sending

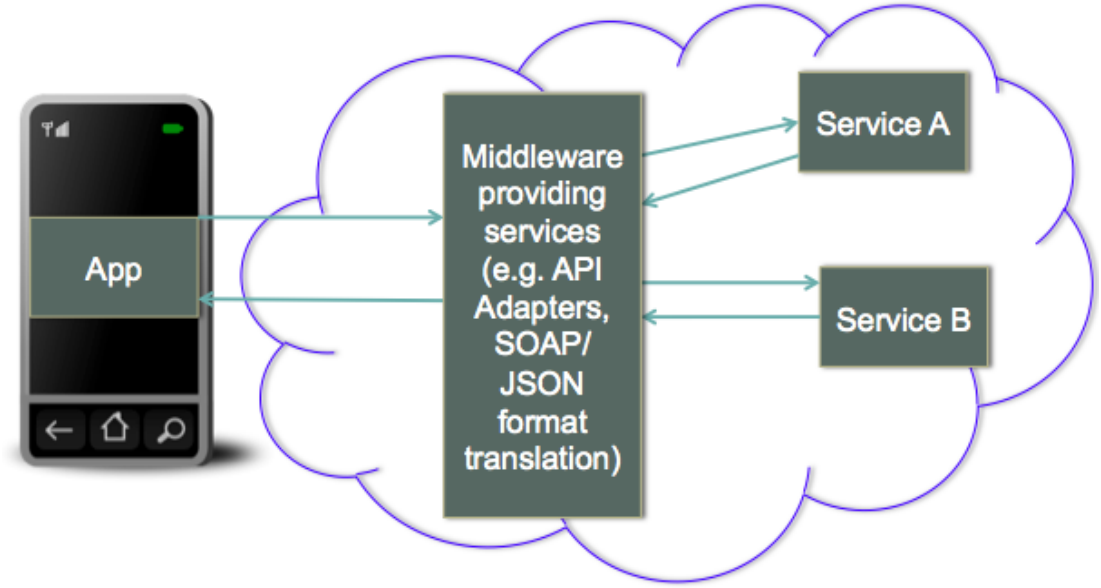


Figure 2.5: A Sample Middleware Model. A middleware is deployed in the cloud, which performs work on behalf of the mobile device. This usually takes the form of the middleware invoking SOA web services to complete work for the user in a asynchronous, non-blocking fashion, freeing the device to perform other work. Results are then adapted back to a mobile friendly format before being returned to the device.

back to the mobile device. The approach is sound but suffers from the common problem that when adding a proxy between the device and the service, a layer of indirection is introduced. As such, the results were slower than invoking the services directly from the mobile device, however this would be expected.

The caching aspects are extremely valuable in the mobile context in the case of disconnections, as is the optimised JSON message size, which will reduce the data processing time and cost. Arguably those benefits would outweigh the disadvantages, as low cost and disconnection support are important to the user experience. However the implementation is not currently practical for the user context, as it requires the mobile device user to provide the URL of the WSDL file describing the service, and to indicate if it is a SOAP or RESTful web service [143]. Non-technical mobile device users will not know what WSDL files are, nor the difference between SOAP and REST. A form of discovery service would be useful here, to take this complexity away from the user. As such the approach lacks deployability to end users in its current state.

Flores et al [39] developed a mobile cloud middleware that is aimed at resolving the problem of cloud provider interoperability. The middleware takes

requests from mobile applications on the device, and will decide which cloud service needs to be invoked. An adapter is used to pass the call to the cloud service. Like Wand and Deter's work, the results can be cached and delivered to the mobile at a later stage in the case of a disconnection. Some sample applications were developed to illustrate the concept, which were designed specifically to use services from different cloud providers. Results were similar to Wang and Deter's work in that the indirection of using a middleware proxy were slower than invoking the services directly, yet as the middleware works asynchronously, the device was free to work on other tasks while the middleware carried out it's workload, freeing the device in the way the authors wanted to achieve. The details on how applications on the mobile device should be written to use the middleware were not clear. For example, how the application should signal what cloud service it requires. There must be some generic API for the middleware, because if this was not the case, then it does not provide a solution the interoperability problem it aims to solve.

2.3.4.2 Other Middleware Solutions

A later work by Flores et al [37] compares what are generally the two overall approaches to mobile cloud computing; delegation of data intensive tasks the cloud (what has been described so far in this thesis as approaches such as code offloading and application partitioning), and the offloading approach (what is described in this thesis as a service oriented approach). Continuing the discussion in this paragraph, to be consistent with the terminology used in this thesis, what the authors of this work refer to as delegation, shall be described here as code offloading, and what the authors of this work refer to as code offloading, will be described here as a service oriented approach. The authors compare examples of these two approaches for their advantages and disadvantages. In terms of code offloading, the authors argue that no code offload approach has taken advantage of the features offered by the cloud platform (scalability, elasticity, parallelisation, etc.). They have developed the EMCO (Evidence-based mobile code offloading) platform to overcome this. EMCO collects context data for virtual machines in the cloud (such as CPU utilisation, memory capacity), and uses this information to determine exactly which VM in a cloud data centre that work should be offloaded to, similar to the M2C2 mobility management system [89]. Having said that, the authors generally are in favour of the service oriented approach. Supporting the code offloading ap-

proach is considered to be difficult because applications do not readily support it without modification, an issue already outlined in this review. In addition, very few applications seem to take advantage of the computational capability offered by the cloud, such as augmented reality and graphics processing applications. Most applications on mobiles do not need a large amount of resources. The service oriented approach is seen as easier to adapt existing applications to, and of course, a service oriented approach, supported by a middleware, can enable disconnected operation in the cloud, where work can be completed for a user while disconnected. As outlined before in this review, the same cannot be said for the offloading approaches.

Another SOA approach which recognises the value of disconnected operation is the work by De and De [23]. This work proposes using a tuple space model, to tackle the problem of decoupling mobile applications from mobile cloud services. The authors argue that traditional service models, such as REST, and data transfer formats, such as JSON, do not provide dynamic flexibility to mobile devices, in the sense that the mobile applications become coupled to these technologies and formats. By creating a tuple space both on a handset, and at a cellular base-station between the mobile device and the cloud, mobile application requests for services, and their responses, can be represented as anti-tuples (a search tuple) and tuples respectively. Tuples are unordered in nature, and because tuples are stored in the handset and/or base-station until the service (for the request)/device (for the result) becomes available, this method also supports mobile devices in disconnected situations. Tuples specify properties of a service being sought by an application, and for the services offered by a provider. No practical implementation details are provided as to how applications specify service requests, or how service providers can describe their services as tuples, in which case, in the view of this thesis work, does not solve the problem the authors claim. The mobile device and service provider would still need to agree on some common naming standards for the properties of services that are searched for with anti-tuples. The work also does not address cellular handoff. However, it is one of the few approaches that does recognise the need for mobility with the mobile cloud; a device will not remain connected at all times. This further demonstrates the usefulness of a middleware in the cloud to perform work for the user.

A final example of a middleware solution is the Avatar middleware by Borcea et al [14]. The authors describe the Avatar middleware as a platform for mobile distributed computation. Similar to CloneCloud [19] and ThinkAir [70], a copy

of the mobile operating system exists in the cloud as a VM. This VM is known as an Avatar; each user has their own, and every mobile device of the user shares that Avatar. Mobile applications can run either entirely on the Avatar, or components can be split between the Avatar and the mobile device. In the nature of the author's aim of truly distributed mobile computing, the Avatar's of several users can collaborate on tasks (if each Avatar VM has a copy of the same application). The work also focuses on the Avatar API, which offers programmers access to Avatar communication, Avatar data storage, and Avatar start/pause/resume/stop commands. Additionally, the cloud deployment architecture is discussed in detail, which focuses on issues such as scalability to billions of Avatars (for reasons such as low latency communication and energy efficiency), and data privacy. This approach is probably the most similar work to the project undertaken for this thesis, except that the Avatar is a VM, where the approach in this thesis work is an SOA-based approach, which overcomes many of the problems associated with mobile cloud computing enabled by means of VM-based solutions.

To summarise, the use of middlewares as a proxy for accessing SOA web services which invoke cloud services on behalf of mobile devices is a sensible one if the solutions to the highlighted problems are solved, as demonstrated in the work by Wang and Deters, and Flores et al. The idea of caching results in the cloud in the case the mobile device becomes disconnected will be a crucial aspect of improving the user experience on these devices, as will optimised services that are both low cost, and can utilise different services from difference providers with ease. SOA approaches can also enable context aware services and are capable of taking context information from the device, which is essential to the aim of personalisation of mobile cloud applications. The use of a middleware solution will be slower than direct interaction, but this concern should be set aside, as the mobile device is not blocked and waiting for the middleware to complete work, it is free to disconnect and perform other work. From the user experience perspective, middleware solutions are the only solutions that can solve the disconnection and mobility issues.

2.3.5 Resource Management

In the mobile domain, resources such as energy, and bandwidth are in little supply. How these are used by the mobile cloud solutions presented in this

review is an important concern. Many solutions have focused on the conservation of various resources such as energy and bandwidth at the mobile device.

Application partitioning and code offloading solutions presented already [21] [19] have cited the energy required in running an application locally on the mobile device as a consideration for offload; they profile the estimated energy cost in running an application locally, and then calculate an energy cost for running an application by offloading it to cloud infrastructure. Only if the estimated energy cost for offloaded execution is less than the energy cost for local execution, will offload actually take place. The consideration itself on the energy cost of offloading is actually dependent on the estimated bandwidth currently available on the network connection. This may vary over time. Disconnections may also occur while the application code is offloaded. In this case, the entire offloaded execution and resulting resource usage was wasted, and the task must be re-executed locally. Very significantly, in the MAUI approach [21], in experiments that were carried out using the cellular 3G connections, the offload decision maker chose never to offload. It always opted for local execution rather than offload onto the cellular network. The application partitioner Alfredo [45] operates by dividing a mobile application up into various components which are distributed amongst local computing infrastructure, where each component is executed. The actual partitioning corresponds to an application graph, which is dynamically created based on what is required to be optimised, such as "optimise in such a way that will result in the lowest energy/bandwidth usage", or "optimise for minimal execution time. A Quality of Service (QoS) and power-management aware framework by Ye et al [147] focuses on mobile cloud service migration based on power and QoS constraints. If it is too costly to use a remote service from the device, or the latency is too high, such that QoS will not be met, the cloud service can be migrated to the device, and can be used locally at lower cost. The work proposes algorithms for migration, along with the decision process for local versus remote execution. The solution also supports chained services, as the user may use many services in combination, and supports services that may not be migrated, due to constraints such as being tied to a database.

The analysis of energy usage by the antenna in the mobile device is also covered to some extent in the literature. As outlined by Schulman et al [120], much of the energy cost associated with using the network interface of the mobile device actually comes from what is known as the "tail" energy of the network interface. Once the communication has actually completed, the net-

work interface stays active for a few more seconds before sleeping, to receive any remaining packets from the server. This can be quite costly for a short communication. The cost of this grows substantially as different network operations take place. The work in the paper is in the implementation of a system called Bartendr, which groups network requests together, so as to reduce the number of tail energy occurrences. Balasubramanian et al [7] describes the high cost in "ramp" energy, to actually transition the network interface into a high power state from sleep or low power states, as also being very significant. Their TailEnder approach also tries to group together network requests so as to minimise the ramp and tail energy. In looking at the actual energy usage figures of the antenna, the work by Heinzelman et al [50] describes the energy used by the antenna of a micro-sensor in a wireless sensor network, as a product of the electrical power required for the antenna components, along with amplification power, the size of the data to transmit, and the distance from the base-station.

In terms of modelling of data bandwidth in the mobile context, while there are published figures, which specify what the upper bandwidth of a given type of connection should be (such as EDGE, HSDPA, or Wireless 802.11b/g/n/ac). Some approaches in the past have looked at the allocation of bandwidth in regards to the GSM networks handoff bandwidth [110] [75] [77]. Liang et al [77] proposed a fair scheme for dynamic bandwidth allocation in WCDMA networks. Tocado et al [113] developed an Android based monitoring application that can collect information about cellular networks on the move, and they used this data to analyse the performance of cellular networks. For example, the authors take a train journey from the city to the countryside to evaluate network changes en-route. Work by Gomes et al [46] examines a splitting and merging approach to cellular base stations that are backed by emerging Radio-over-Fibre technology. By organising cells into multiple tiers, based on splitting and merging of the cells, they aim to meet a number of different optimisations, such as an increase in network capacity during busy periods, and to reduce it when demand is low. The optimisations of the multi-tier organisation can support other objectives such as energy saving and cost saving/revenue maximisation. Kivekas et al [66] ooked at how bandwidth is influenced by the design of the mobile device handset itself, along with the positions of the hand of the user holding the device, and the position of the device relative to a model of a human head.

Some works exist in terms of allocating cloud resources to meet the demands

and QoS requirements of resource requests from clients. Liang et al [76] proposes an economical cost model to deal with offloaded tasks. Their approach aims at identifying an optimal allocation of cloud resources to offloaded tasks, and rewarding those instances that provide resources with an income. Rahimi et al [109] examine using a tiered cloud-based system to execute mobile applications as a Location-Time-Workflow (LTW). Their heuristic algorithm, MuSIC, can reach a 73% optimal allocation of resources for mobile applications to meet specified QoS requirements. Hongbin et al [53] used a Semi-Markov decision process to model mobile cloud domains, where requests for resources could be transferred to neighbouring cloud domains, with the aim of maximising rewards for the providers of the domains involved for completing requests, while also considering the cost of such request transfers. Using resources from different clouds to satisfy requests has been explored in other works. Kaewpuang et al [63] developed a framework that allows different cloud providers to share their server resources to meet the resource requirements of their users, again, with a focus on maximising the revenue for each of the cloud providers that participates in sharing. They also focus on solving optimisation problems that indicate when to share, and the optimal number of resources (such as application servers) required for sharing to meet request demands. The SAMI model by Sanaei et al [115], is a multi-tier infrastructure framework, which aims to meet latency, resource requirements, and security concerns, by utilising existing infrastructure resources from MNO's instead of the cloud. An MNO can also delegate to trusted third party infrastructure providers, and utilise cloud infrastructures, meaning it can also act as an arbitrator for cloud resource requests. In addition, this framework utilises an SOA approach to using services in the cloud to meet service requests from users. The PhD thesis of A. Beloglazov [11] studies distributed dynamic virtual machine consolidation techniques and algorithms to meet energy efficiency requirements, while being able to meet user demand.

2.3.6 Discussion

It has been seen in this section that different approaches to mobile cloud computing all try to achieve mobile-cloud integration using different methodologies. All approaches have different, as well as some common, advantages and disadvantages:

- The Cloudlet approaches, based on the use of local infrastructure, are essential for real-time applications with low latency requirements. They will allow the user to run their own applications and data, but this approach suffers from bandwidth concerns when multiple users are involved, and mobility concerns. VM-synthesis of the base-VM and the user's VM-overlay upon connection to the Cloudlet requires a non-negligible amount of time, and disconnection has not been discussed in terms of any graceful solution.
- Application partitioning and code offloading solutions can be highly desirable for the user experience, as the user need not be located close to a Cloudlet and remain there. Considering that there is no repeated data transfer after the initial offloads of the components/code, these solutions can be bandwidth and energy consumption friendly in this regard. However, profiling costs while determining how to partition application components, or determining should code offload take place, are non-negligible and do not make for an integrated user experience. For mobility, disconnection is not gracefully handled, and as stated in the MAUI work [21], can actually be wasteful due to local re-execution. Also, with MAUI, cellular 3G performance was shown to be practically un-usable due to high latency, and the other works examined did not even attempt to evaluate it.
- Remote display solutions use virtual machines as with Cloudlets, but unlike Cloudlets, are not located on infrastructure near the user, making this the most costly solution for the purposes of an integrated user experience, due to repeated data transfer requirements of the visual GUI output of the virtualised operating system, even if nothing has changed. These issues exist alongside the fact that, as with Cloudlets, desktop operating systems typically are not designed for use on mobile devices in terms of user interaction, form factor on small screens, and not being able to take advantage of on-board sensors for context information.
- Middlewares can be extremely beneficial in terms of mobility aspects. In fact, this is the only approach that really tackles this concern. Middlewares can perform work and save results, while a mobile device can disconnect and perform other work. As such, unlike the other approaches, there is no requirement for continuous connection to the cloud. These approaches often rely on SOA web services that are already deployed in

clouds (and hence, should have a low or zero adaption cost to the mobile device, unlike modification required by application partitioning and code offloading solutions), and transfer data in formats such as XML or JSON, which will result in low data transfer size compared to some of the other works presented in this review. As a result, it should be the case that these approaches will not be impacted so much by poor latency and bandwidth on cellular networks. It is up to the middleware implementation to ensure power and bandwidth resources are utilised sparingly but effectively. Middlewares and web services can also make use of context data from the mobile device sensors. Finally, the middleware can provide content adaption to suit the requirements of the mobile device, for example, as Wang and Deters's [144] have done with the conversion of XML to JSON data formats.

For the user experience goals, it would appear that middlewares, combined with SOA web services, is the most promising approach for the mobile cloud paradigm. In the next section, context awareness with mobile cloud computing is explored.

2.4 Context Awareness

The use of contextual data collected from the mobile user, can allow the device itself, or applications, to adapt their operation or condition to the user's situation. This can occur by either by personalisation, or by providing situationally relevant information or functionality. Chen and Kotz [16], having examined various previous definitions and usage of context data, define "context""with the following: "Context is the set of environmental states and settings that either determines an applications behaviour or in which an application event occurs and is interesting to the user". An application that can utilise user contextual data for its operation, is said to be "context aware". The origins of context aware computing were first described by Weiser [145], while presenting a vision for what has become known as ubiquitous, or pervasive, computing. The goal of ubiquitous computing, is that computers should be located all around us, blending into our environment. The next section explores how contextual awareness has been used with mobile devices. Following that, considerations as to how context awareness plays a role with mobile cloud computing, are discussed.

2.4.1 Mobile Devices and Context Awareness

Mobile devices are the perfect enabler for truly context aware computing. As a result of their portability, which allows them to be carried while on the move, they are also the perfect ubiquitous device. Compared to their desktop counterparts, mobile devices feature a wide variety of sensors and other equipment that the device, and any applications running on it, can use to make informed decisions about their operation for the user, without any personal intervention. This can easily be seen by anyone who visits the Apple App Store, or the Google Play stores, from their mobile devices today. By far the most common sensor on a mobile device is a GPS, which allows the device to detect the physical coordinates of the device location, using triangulation with satellites. For example, travel applications use this information to pinpoint the user's location on a map, and to show places of interest in the user vicinity. Other on-board devices include an accelerometer, which can detect physical user activity (walking, running, sitting, etc.), a light sensor to detect brightness, and even temperature reading sensors. Desktop computers, typically do not feature any of these kinds of devices, and are not context-aware to the extent that mobile devices can be.

Research has harvested this information for mobile users to provide relevant computing services on their devices. For this purpose, there are several specific questions concerning collection and use of context for mobile devices:

- How is context data collected?
- How is context data stored?
- How is user context represented?
- How can context data be used?

The research literature for context awareness with mobile devices, explores these questions with varying different approaches. Going forward, it is important to define *Context Producers*, and *Context Consumers*. Context Producers are a source of context data, which can be a sensor on the mobile device, or even information provided directly by the user. A Context Consumer is an application or service that will consume context data for some specified objective.

One of the earliest works in regards to context awareness on mobile devices is that by Hofer et al [51]. This work developed the Hydrogen approach for context-awareness on mobile devices. This is a three-tier framework for cap-

turing, storing, and providing contexts to consumers, from a mobile device. The aim of the architecture is to separate the operations involved with collecting context data (the producers), from the application who require the collected context data (the consumers). The framework collects a pre-specified set of context data, such as network information, and date/time information, by means of adapters. This is provided to an upper management layer, which runs a Context Server. From the server, the context data can be shared with other applications. It appears that the context data collected, is not persistently stored. Context that is collected is represented as objects using an Object Oriented approach. As a result, the context consumers, in advance, need to know exactly what kinds of context are collected by the framework. In terms of mobile cloud computing, an important observation, is that all of this work is carried out locally on the mobile device. This can be troublesome, as sensing and collecting context data repeatedly results in an undesirable power draw from the device battery, impacting negatively on the user experience.

In work by Lowe et al [85] a Context Directory was developed, which is a directory that stores context as key-value pairs. They key-value pairs approach does restrict the complexity of context data collected, and also implies that a consumer must know a context type by name when requesting it, similar to the hydrogen approach. Contrasting with the Hydrogen approach, this work uses a server for processing and representing the directory itself, rather than storing this data locally on the mobile device. This work does not discuss how context reaches the context directory (and also points out, that there could be several distributed directories).

This Context Directory project introduces two important concepts. One of these is what is known as *Feature Extraction*, also known as *Context Interpretation*. Feature Extraction is the means by which meaningful information can be extracted from the raw context data, and understood. This is a complex subject; the work by Lowe cites several feature extraction algorithms, however, for their implementation, an unspecified rules-based approach is used. The important point to be made is that it is recognised that operations like feature extraction, can be computationally expensive, and long running. As such, these operations are considered costly for a mobile device to perform, hence the desire to offload these operations to a server, such as a cloud-based approach. The Hydrogen approach, does not perform any feature extraction, and overall, may not have been viewed as a framework that was costly to run on a mobile device.

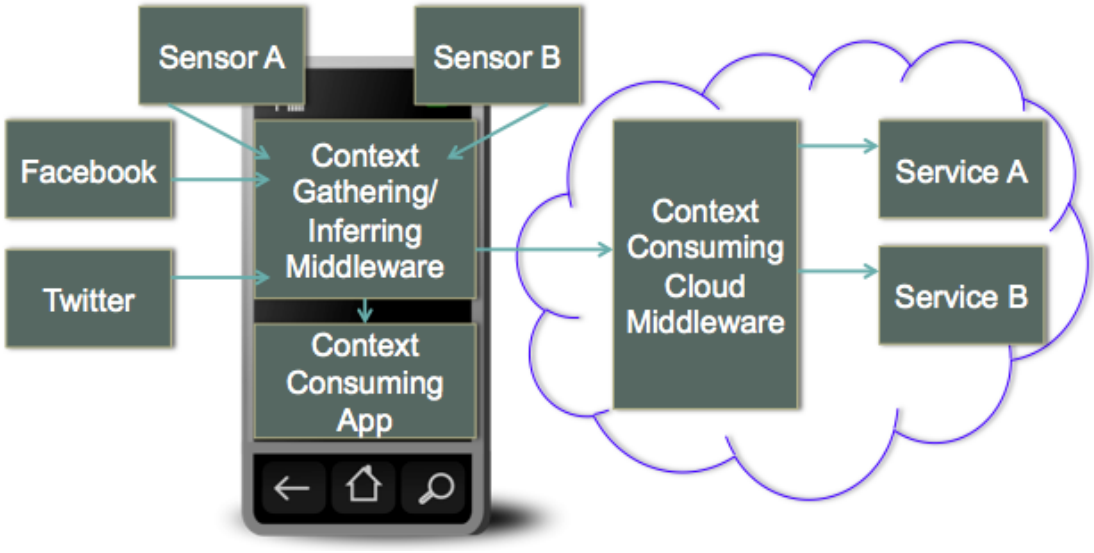


Figure 2.6: Context Awareness with Mobile Cloud Applications. A middleware running on the mobile device collects user context from several sources. This is then forwarded to a cloud-based middleware which consumes this context, and then, in turn, passes it on to various cloud based services. This can be used to personalise the service execution and/or results to the user's own situation or preferences.

For the mobile device, the other important consideration is storage. For the Context Directory, running on a server, context data can be persistently stored. This results in context data taking the form of a *Context History*. The usefulness of this for the mobile cloud work in this thesis will be discussed further in the next subsection. What is important to highlight is that constant collection of context data, along with persistently storing it, requires significant power and storage space, which is also undesirable for a mobile device.

In work by Raento et al [108], what was titled a "ContextPhone" was developed; a context-awareness framework that allows development of context-aware applications. This work focuses on gathering context from a pre-selected set of device sensors. This data can then be shared with applications built on top of the framework. In addition, this work places a heavy focus on context data sharing via network connectivity (i.e. Bluetooth, GPRS, SMS Messaging, which are all supported). For example, one application, ContextContacts, can share the context of a user with others who have that user in their contacts. Two use-cases are presented for this; one is the capability for a user to see the phone noise and vibrate settings of a contact. If the ringer is turned off, and vibrate is on, one could assume that the contact is in a meeting, and

should not be disturbed. The second is location information, where you can see where that user is, should you wish to meet with that user. The authors do not consider privacy options of openly sharing this data, but stated this was something they were interested in exploring. The capability for context sharing is shared with the Hydrogen approach, where the authors implemented a contact business card information sharing application. In terms of the questions identified for this review, data storage, representation, and usage outside the framework, are not discussed.

In work by Oriana Riva [112], the Contory (ContextFactory) was developed, with a focus on retrieving context from various distributed services, and is one of the most complete works in the literature on mobile context. This approach was taken based on the fact that not all devices may have the same sensors, and so required data may not be available locally. Three different strategies are used to collect data; internal sensors, centralised external sensors, or ad-hoc external sensors, over Bluetooth and WLAN connections. The Contory framework is deployed in the device; it can publish context gathered from internal sensors to other devices (in either push or polling fashion), and it can retrieve requested context from other devices using the external sensor strategies. Context data is stored in a local repository, in an unspecified format. An SQL-like query language was also developed to allow a developer to query context information through the Contory API. Like with the Hydrogen approach, all the work is centralised on the mobile device. Details regarding the implementation of the externalised central server implementation are not given. Contexts are stored as types of *Context Items*, and can be stored locally on the device, or on a remote server. One concern may be the willingness for other mobile devices to share their context data, from a privacy perspective, but in the implementation, a mobile device running Contory can tag itself as being willing to share certain types of context with others.

The Mobicon (Mobile Context Monitoring) platform was developed by Lee et al [74]. Rather than a framework for context awareness, it focuses on *context monitoring*. The objective behind the platform was to be able to collect the context data from many sensors and devices in the vicinity of the user, such as those found in personal area networks. Despite the fact that the platform runs locally on mobile devices (it does not make use of clouds or servers for operations such as feature extraction), the work employs many techniques for optimisation to reduce the energy cost of gathering and disseminating context data to consumers. This includes aggregating the context processing opera-

tions in a pipeline together for multiple requests (the Contory framework also performs a similar aggregation to save energy while processing multiple context requests), resource optimisation so that an overview of the system resource usage is maintained to meet objectives (such as not to exceed a certain energy usage), and a method to reduce the number of sensors required to determine a context status to a minimal set, which is called the essential sensor set (ESS). Also similar to Contory, a language called CMQ (Context Monitoring Query) is defined to allow a developer to specify what context situations an application should be alerted to. The platform has been provided to multiple developers to build applications upon it. However, the work does not describe how all the sensors are discovered by the platform. The work also does not describe how context is stored or represented, but it does list a pre-determined list of contexts available.

In the next subsection, techniques for context representation and storage are briefly examined.

2.4.2 Context Representation and Storage

In the research literature, several different approaches have been explored for representing collected user context information. The different options for representation, also determine the possible storage options. The survey work by Strang and Linnhoff-Popien [132] looked at the advantages and disadvantages of several different approaches to context representation. The different approaches were evaluated using the following criteria:

- *Distributed Composition*

  The ability of the approach to cope with the many distributed sources of context information, and the lack of a central system for administration.

- *Partial Validation*

  The capability of the approach to be validated for conformity to an existing model.

- *Richness and Quality of Information*

  The ability of the approach to indicate quality levels of context data which may vary with time.

- *Incompleteness and Ambiguity*

Fresh context data may not always be available (i.e. over time context becomes *stale*). The approach should be able to handle this situation.

- *Level of Formality*

  The ability for all sources of context to be able to share the same meanings of relationships between entities.

- *Applicability to Existing Environments*

  The ability for the approach to scale and adapt to use in existing software systems, such as with web services.

The work explored many of the approaches already discussed in this review, such as Object Oriented approaches for representation, and key value pairs. In their analysis, the approach that met with most of their specified set of criteria, was to represent context data using an *Ontology*. An ontology is made up of a set of logical *axioms*. The axioms can be used to express *relationships* between *Individuals*. For example, a "Father" individual, could have a "hasSon" relationship, which points at a 'son" individual. A *Context Reasoner*, also known as an *Inference Engine*, can parse they axioms to make inferences about what is true and false.

As a side note, under the criteria from Strang and Linnhoff-Popien, it should be easy to see why an approach used in the works explored here, such as the key-value pairs used in the Context Directory work by Lowe et al, would not be considered ideal. With simple String-based keys and values, there is no formality, and no validation. There are several implementations of the ontology approach, such as the SPICE mobile ontology in the work by Villalonga et al [142]. This is a standardised ontology, which can be used to represent elements of mobile context. The SPICE ontology supports distributed composition, as the markup is XML-based, and new context can be written automatically to the ontology by many sources. Validation occurs against the defined SPICE model. The ontology is easily extended with custom properties and relationships to support properties to specify the source and quality of the context. The inference engine can also infer new context, based on old context previously gathered, should no fresh context data be available. The formality and meaning of the relationships is also defined by the model (which comes complete with developer documentation). XML can be stored as flat-files in a file system, and can easily be written and read by existing software systems and programming languages.

Next to the ontology based approaches, the Object Oriented approach, as used in the Hydrogen approach [51], was the second recommendation in the survey. This also met all the criteria, but not to the same level as Ontology approaches, falling slightly short by comparison, in terms of partial validation, and level of formality. An additional example of an object oriented approach can be found in the Java Context Awareness Framework (JCAF) by Bardram [8]. Other works in the area of context representation include work by Korpipaa and Mantyjarvi [69] in which a mobile ontology was developed for representing context information gathered from the mobile device sensors. It is not as extensive as the SPCIE ontology, and the applications of the ontology are purely focused on actions that can be taken by the mobile device itself. Other approaches include a model driven approach, such as the approach taken by Sheng and Benatallah [123], which uses UML-based modelling to represent context.

2.4.3 Context Awareness with Mobile Web Services

For this work which relies on working with services deployed in cloud environments, an important consideration is how context data can be used with existing web service technologies; both in terms of service discovery, and service consumption/execution.

To begin exploring how context data can be used with web services, Truong and Dustdar [137] surveyed existing approaches to how context has been previously used with web services in the literature. Several context frameworks were evaluated in terms of their compatibility with web services under different criteria. The criteria used is a superset of the criteria that was specified at the start of this section as being of interest to the work in this thesis, and is as follows:

- *Type of Context Information Supported*

  Location, Activity, Network Status, Devise Profile, etc.

- *Context Representation*

  Ontologies, Objects, UML, WSDL (an extended version), etc.

- *Use of Sensors*

  Mode (automatic/manual), Sensing Technique (polling/request-based),

Sensor Interface (Web Service, specific API), Data Retrieval and Pushing (Query, Subscription, push-based).

- *Context Storage*

  Storage Model (centralised/distributed), Storage Database (XML, relational), Access Interface (web service, other approach), Request Specification (SQL, SPARQL).

- *Context Distribution*

  Overlay Network Distribution, Direct Transport Distribution, Access Mechanism

- *Context Reasoning*

  Support for Reasoning, Semantic Support

- *Context Security and Privacy Techniques*

  Encryption, Authentication, Authorisation, Privacy Specification

- *Adaption Techniques*

  Adaption Purpose, Adaption Specification, Adaption Layer

In the work, each project from a specific set of web service frameworks that supported the use of context was evaluated under each criteria listed above (mobility was not a factor in the study). None of the projects in the study met all of the criteria that was specified. The main issues discovered were the lack of support for distributed context management (most approaches relied on a centralised server), the use of XML as context representation (note that ontologies can be represented as XML markup, but here it is meant that pure XML is used without ontologies), and security and privacy mechanisms being inadequate. An example of one of the projects studied is a middleware system by Gu et al [47]. This middleware, known as Service-Oriented Context-Aware Middleware (SOCAM) provides an architecture for building context aware systems. As recommended by Strang and Linnhoff-Popien, they define a formal ontology for representing context data that can be used in a smart-home setting; the ontology is marked up using RDF, the Resource Description Framework, a W3C standard, normally used for modelling web-based resources in a descriptive manor with meta-data.

In regards to mobile cloud and context awareness, work by Han et al [49] uses

a client-server proxy approach to adapt the context of the mobile device, to the fetching of web service based resources, based on the state of the mobile network connection, in an implementation called AnyServer. The focus of the work is not a generic use of context for applications, as with other works. Additionally, a work by Hyun-Jung and Kim [57] provides a framework, which detects gaps in mobile context resulting from the changing state of the user, so that missing context, can be provided to services, which require context that may be continuously changing.

There are many sources of context. The use of the previously mentioned context history for inferring missing context is presented in the work by Hong et al [52]. New context can be inferred from context history, so that if new context is missing, or if energy should be saved on the mobile device by not collecting new context, the context engine will not fail to provide context when required, with some level of confidence. In the work by Beach et al [10], the authors implement a framework for fusing context data from the mobile, its sensors, and social networking sources such as Facebook, to develop a full context profile of the user.

There are also existing mobile cloud applications and middleware systems which make use of context. One example of such a work is that of Abowd et al [1], called Cyberguide, which uses the location and location history of a user to replicate functionality and services offered by a real tour guide. The previously discussed mobile cloud SOA mashup middleware by Wang and Deters [144] utilises context for providing web service mash-ups to users, along with experimental applications they developed for testing, which use collected context for their operation.

2.4.4 Discussion

Building context awareness into a mobile cloud solution is an essential factor in personalising the user experience, and for really utilising the sensing devices that mobile devices bring, compared with their desktop counterparts. Consider the Cloudlet, code offloading, application partitioning, and remote display techniques for mobile cloud computing discussed earlier in this review. These approaches are incompatible with context-awareness applications.

Taking the Cloudlet approaches [117] [141], and the remote display works [125] [124], these are based on morphing desktop operating systems, into a form for

use on mobile devices. These approaches do not unique cater to the mobile device. Desktop operating systems are not designed to make use of on-board sensors that feature on mobile devices, and so they must be ruled out of a solution involving context awareness (at least without further extended work and modification to the underlying software). Code offloading works and application partitioning solutions, have the benefit of uniquely catering to mobile users, because these projects work with native mobile applications, that are capable of using sensor data. However, this would mean that the parts of the application that are responsible for communicating to the sensors and retrieving the data, could not be separated from the mobile device, and must be executed locally.

The use of middleware approaches deployed on the mobile device can be a useful solution that uniquely caters to mobile devices. Additionally, the use of context data with web services can also uniquely cater to mobile experiences. Services can receive raw context data from the mobile device collected from sensors. Yet, these are not straightforward solutions either. As seen from the literature, repeatedly collecting data from sensors, and processing that data into context is not a cheap operation, considering the resource constraints present in mobile devices. Works such as the Hydrogen approach [51], where the entire middleware solution is deployed locally on the device, can cause strain on the physical resources such as the CPU, and repeated sensing and data collection drains the device battery. The Context Dictionary approach taken by Lowe et al [85], where the heavy computation is offloaded to a server, is the key idea to save resources, and of course, is fully compatible with the aim of mobile cloud computing.

Other aspects that must be considered in the solution, from a mobility perspective, include device disconnection, which can prevent the server responsible for collecting the sensor data from receiving fresh context from the mobile device. Again, the context history solution by Lowe et al [85], can be used to infer new context from historical context, if fresh context is not available. Another reason why this is important is because even with the server responsible for the computationally heavy work, constant collection of context data, and sending it to this server, is not a cheap process.

In the next section, the use of web services for the mobile cloud computing paradigm is examined.

2.5 Web Services

Many approaches to implementing mobile cloud computing take the form of offloading entire applications or computations to a server, such as the Cloudlet approach already discussed [117]. Other approaches that have also been described, such as the code offloading approaches, require applications to be modified to support these techniques, which requires additional time and developer effort. Even then, not all applications are suitable for offload.

The use of mobile web services has been seen to be a supportive technology for supporting Weiser's vision [145] for ubiquitous computing in work by Bashah et al [9]. In this particular work, the focus is on services as a means of completing a user task, and how services can be dynamically discovered in a mobile computing environment, with devices entering and leaving that environment frequently. In work by Tergujeff et al [135], the authors define "Mobile SOA, which combines service oriented design of systems and applications with the domain of lightweight mobile devices". It is their opinion that "the features of SOA are essential in environments that depend on non-robust connections and multi-device user access to services. This makes SOA based implementations natural components of mobile application composition". For this thesis, the focus is placed on services as would be considered in the Software Oriented Architecture (SOA) literature; specifically, web services. These are already deployed on cloud infrastructures, and are accessible through public APIs. They can provide useful information and functionality to calling software. In this section, the research literature is examined for work on mobile web services. This section also examines existing service discovery techniques, with a focus on mobile service discovery.

2.5.1 Mobile Web Services

In this subsection, research is examined that has used web service technologies with mobile devices. The work by Bashah et al [9], even though not focusing on services from an SOA standpoint, highlights several of the same factors that are of concern to mobile services, including discovery, descriptions, and consumption. In the following subsections, each work is evaluated for the following criteria as appropriate:

- *Service Deployment*

Where are the services deployed; on a server which may be cloud-based, or on mobile devices.

- *Service Descriptions*

  How is the capability/functionality of the service described for consumers.

- *Service Discovery*

  What service discovery mechanisms are used to allow a service to be found by interested consumers.

- *Service Consumption*

  How can the service be executed by the mobile device user.

An example of an early work in the area of mobile applications with SOA technologies is an investigation by Thanh and Jorstad [140], on SOA considerations for providing mobile services, such as the GSM telephony and the Short Messaging Service (SMS) in SOA fashion. They described three primary concerns for mobile SOA; movement of users between devices, movement of devices across networks, and movement of services across domains. While this work was focused on services offered by mobile network operators (MNO's), some of these concerns are valid today, as has already been discussed in relation to network disconnection concerns. To this end, the authors proposed a generic mobile service model that is capable of comparing services for equivalence; this is so that if a user moves between networks, an equivalent service for the user's work can be found.

Another early work is the Mobile SOA Framework by Ennai and Bose [33]. Their MobileSOA framework, running on mobile devices, allows for development of mobile applications using what the authors describe as Web 2.0 technologies. As such, applications are displayed in a mobile web browser. This framework makes both local and remote services available to the application in SOA style, specifically as RESTful resources. MobileSOA is also context aware; the aim of this is for business applications to be able to update in real time with new information from the central enterprise IT server located at a company headquarters. This roughly translates to new information being sent in push fashion to on-device applications, and the display of information updating itself to the current user context. Little information is given regarding the implementation; a real application built on top of MobileSOA is described

but not shown. This application takes advantage of other features of Mobile-SOA such as secured application sessions. The authors acknowledge that for mobile services, mobiles as both service consumers and producers is important, but they do not discuss costs associated with mobile devices offering their on-board features, such as cameras, as a service.

The work by Ennai and Bose is focused on a mobile application making use of services deployed on business servers, and combining them with local services to provide useful functionality and data. This corresponds to a common model today, where mobile applications work with information and applications deployed on cloud servers. However, a mobile application consuming services is not the only model of mobile SOA. Tergujeff et al [135] state that "the subject of mobile WS (web services) can be approached from two directions: mobile devices as consumers, and as providers of Web Services". The evaluation of the research literature for the most part, can be divided up based on the service deployment model; web services executing on mobile devices, and services executing on servers (or clouds) which are called from mobile devices. Tergujeff et al conclude that both approaches are required for seamless SOA on mobile devices, but since that time, mobile device battery lifetimes have not developed significantly, and the idea of a mobile device as an always-available server, providing services to clients, is not consistent with the thinking that necessitates the mobile cloud paradigm.

2.5.1.1 Web Services on Mobile Devices

Many works have explored the use of web services on mobile devices, from the perspective that a mobile device can act as a server, providing services to clients. For example, a use-case for this approach would be that a mobile user can offer a live video stream from the camera on the mobile device, as a service, to interested friends, of concerts or sporting events. In the situation where a friend could not attend, they could view the event from the camera feed. An example of this is the mobile web service provisioning framework by Elgazzar et al [32]. In this work, the cloud is used for web service discovery, but the web services run on mobile devices, rather than in the cloud. The Location-Aware Service Provision and Discovery (LASPD) framework by Zhu et al [151] is focused on services hosted on servers, as well as on mobile devices, in the same way as the framework by Elgazzar et al, and implements locality based discovery and consumption of services for mobile devices lo-

cated nearby. A three-tier approach is used, with a top administrative tier, a second tier which contains servers providing services in a local area, and third tier which contains mobile devices offering services. Discovery can take place, and queries for services can express to find a service within a given distance of the caller. If a service is not present in one area, the super-peer, located in the top tier which governs an area, can send the request to other area peers. Service descriptions are not described in this work.

Another solution is the VOLARE middleware [105], which monitors the mobile device context, so as to dynamically change service providers at runtime to maintain a certain Quality of Service (QoS) level. The implementation is not fully explained, and details regarding discovery of services, as well as service descriptions, are not provided. A video streaming application is provided for evaluation; as the network conditions/quality changes, the middleware switches the video provider to one offering a higher/lower quality video, to meet the required QoS level. The focus of this thesis work is on disconnected approaches, rather than applications that require real-time streaming, influenced by constant context updates.

Regardless of how compelling the use-cases may be for mobile devices as service providers, from a mobile cloud point of view, the goal of using a mobile device as a server to offer services runs opposite to what mobile cloud computing aims to achieve: removing work from the mobile device. As with a web service hosted in the cloud, the same expectation would be present of the mobile devices offering the services. Availability, reliability, and scalability, cannot be guaranteed on a mobile device. The mobile device would ideally need to stay powered on, and connected to a network at all times with a high quality signal, and would need to be able to provide its resources to many users. This expectation is not feasible for a mobile device.

In the work by Elgazzar et al, disconnection support is said to be provided by various means, such as service caching, service replication, or cloud based support. Cloud based service support is essentially the same as the model for the work in this thesis. Service caching is a technique where the service is moved from the service provider, to the service consumer, as a local copy of the service, or is moved to a third party "proxy", which can take over the role of service provider. The work does not give any implementation details on how this takes place, aside from stating WSDL modifications are required. Consequently, the LASPD framework also supports proxies for calling mobile

services, due to valid concerns about mobile devices not being available at all times; consumers call mobile services, located in the third tier of the framework, through the proxy. Services in a cloud setting cannot be assumed to be "movable" to a mobile device, as the service may be implemented by a programming language that is incompatible with the local mobile platform. Service replication is when the same service can run on different mobile devices. This is a more practical approach, and should be invisible to the service consumer. In the use-case example given at the start of this section, it would require more than one person to be present at the event to stream it with the camera.

The work on the Extended Mobile Host Complex Web service Framework (EMHCWF) by AlShahwan and Faisal [3] may shed some light on the processes behind moving mobile web services between devices, as mentioned by Elgazzar et al. This work also focuses on hosting web services on mobile devices, and takes particular focus on partitioning complex web services between devices. Similar to the way Verbelen et al [141] re-defined a Cloudlet for their work to be a collection of nearby mobile devices, the authors of this work redefine mobile cloud to be a collection of mobile devices; each of these devices is capable of running a partitioned, simplified web service, that makes up part of the larger complex web service. This partitioning is specified by a simple properties XML file, which specifies a business workflow, as to how the complex service can be broken up into simple services, and how these simple services feed data into each other. The break-up of the services must be defined by the developer in advance. The main goal of the partitioning in this work is not to provide availability in a situation where a mobile host offering a service has failed, but with overcoming the physical resource limitations in mobile devices, making it difficult for them to execute a complex service for many clients. In the work, an orchestrator delegates the broken down services to "auxiliary mobile hosts", and also oversees the execution defined by the business workflow. The framework is deployed on a mobile device, rather than a cloud server. Service descriptions are not discussed, and neither is service discovery; the evaluation undertaken by means of a demonstration calls the known service directly.

2.5.1.2 Mobile Devices using Server-Based Web Services

The second approach is the use of server-based services, rather than mobile-based services. There are other works that take similar approaches, which do not rely on any services based on the mobile device. The work on Service-Based Arbitrated Multi-Tier Infrastructure (SAMI) by Sanae et al [115], is an architecture that uses cloud infrastructure located at the mobile network operator (MNO), with an SOA approach, to deliver services to users that are provided by mobile network operators. Their argument for the merits of this approach is that MNO's are generally trusted by their customers. Infrastructure can be located at the MNO, or the MNO can delegate the work to what the authors describe as an "MNO Authorised Dealer". This architecture defines other roles for SAMI, such as that of a broker between the mobile device and service provider. A service registry is maintained and updated by providers offering their services; discovery takes place with standard SOAP requests.

Middleware based research works that use web services have already been presented earlier in this review, such as the mobile cloud middlewares by Wang and Deters [144], and by Flores et al [39], which both rely on SOA-based web services on cloud deployments. Wang and Deter's work features no form of service discovery, and purely relies on the user knowing the WSDL file URL for a service, before invoking it. Service descriptions are provided by the WSDL files. For the service mash-up approach of joining outputs of services as inputs to other services, these are represented as named relations in a MySQL database. The work by Flores et al, also does not feature service discovery. The connection to services is provided by a specified set of adapters, and presumably, does not support other web services not covered by those adapters.

Natchetoi et al [91] developed a Mobile Application Framework, with the purpose of delivering backend business information, in the form of business objects, to mobile users. The goals of the framework meet those of the mobile cloud challenges, such as network disconnection, and handling low availability of memory on mobile devices. They achieve this, first of all by transferring the business objects to the mobile device, by serialising them into RDF, a form of compressed XML. They also allow the user to only download a subset of the information from the business server, that is required for the current task, although it is not specified how. Their work also utilises pro-active pre-loading, where, using a simple heuristic of commonly accessed business objects from

the user's statistics, they pre-load the required objects onto a local device cache; this allows the device user to access the information, even when disconnected. The objects are rendered on J2ME mobile devices as forms. In common with work by Wang and Deters, the work recognises that SOAP, which relies on XML, can be computationally expensive for low-resource mobile devices. The work is purely focused on this concept of business objects within an enterprise; its application or benefits to standard SOA technologies for use on mobile devices, is uncertain.

Looking at popular consumer products, other solutions that deliver web based services to mobile devices include Google Now [93]. This can provide the user with information relevant to his/her situation, such as weather at given locations, and traffic en-route to a workplace. Google Now runs on the mobile device, but pulls information from web services. This product is not easily extendable to third party software and service developers, as it only works with Google services and products, and a selected set of third party commercial applications. Ideally, a solution should work with all kinds of cloud-based services to bring benefits to the users. It also carries out work locally on the mobile device. Like Google Now, Apple Siri [126] runs on iOS-based mobile devices, and retrieves data from web services for the user, based on voice queries. It does not work without an Internet connection.

2.5.2 Service Discovery from Mobile Devices

Much of the service discovery literature in relation to mobile devices focuses on how services can be discovered in a mobile ad-hoc network (MANET). Of interest in this thesis is how mobile devices can discover SOA-based web services deployed in a cloud. Traditionally, service providers willing to offer their services for public consumption register a description of the services available with a known *Service Broker* [88]. The broker stores this description within its own service registry or database. The description is normally in the form of some interface (written in the Interface Definition Language for CORBA [129], or a Java interface for Jini/Apache River [114]). A consumer, interested in finding a service for completing some task, queries the service broker (also known in advance), providing the name of a service interface. The broker then returns the interface to the consumer (if it stores a matching service interface), allowing the consumer to communicate directly with the service provider using the in-

terface. This approach of a broker that facilitates binding of clients to services, is known as *service broker architecture*. An example of work which has adopted this model to mobile devices is the paper by Koponen and Virtanen [68], which aims to provision different services for mobile users that may be offered using different access mechanisms and discovery solutions that are not interoperable, as the user moves between what the authors describe as different service discovery domains.

Recent work has taken place that extends the service broker concept specifically to cloud environments. This is recognised by the NIST Cloud Computing Reference Architecture [82], where a *cloud broker* will not only perform the traditional functionalities of a service broker, but also additional features, such as managing the delivery of the services from the providers to consumers, with service aggregation, and intermediation. The Service-Oriented Cloud Computing Architecture (SOCCA) by Tsai et al [138], designed to allow interoperation between different cloud service providers, also features a Cloud Broker Layer, which can provide features such as service ranking and SLA negotiation. Cloud Foundry [42] also provides a Service Broker API to allow the development of brokers for cloud based services to be deployed on the platform.

While service/cloud brokers exist to facilitate service discovery, these models do not consider service discovery of SOA-based web services from mobile devices. An early example of the need for discovery of web-based services from mobile devices comes from Al-Masri and Mahmoud [2]. They developed the MobiEureka system for this purpose. The motivation behind the system is to rank services based on how compatible the service is with a user's mobile device. For example, a service may require a higher screen resolution, or software library runtime, which is not available on the device. At discovery time, a device profile is compared with a service profile, which describes the requirements of a service. Services that are incompatible, because a device lacks the requirements, are assigned a lower ranking. To support this work, they created an extension to WSDL, known as WSDL-M (M for Mobile), which describes the device attributes required by the service. Of course, not all services are going to be optimised for mobile devices, and a device independent solution would be preferable. Also, WSDL files are not used to describe more modern RESTful based services, which typically have no description mechanism, aside from developer documentation.

Tying in with the previously discussed mobile web service framework by Elgazzar et al [32], the same authors also later developed a framework specifically for mobile service discovery, known as DaaS (Discovery as a Service) [31]. In this work, a discovery process takes place to find appropriate web based services for users; the discovery process takes into account features, such as user context, which has also been implemented in this thesis work. This work however, focuses on discovering services deployed on mobile devices, rather than in cloud settings. Although, for the evaluation of the service, cloud-based SOAP services were used, rather than services on mobile devices. The discovery process itself is facilitated by the cloud, as this is where descriptions for the mobile services are stored. These are gathered from service registries located on the mobile devices offering the services, by means of an unspecified crawling method. An algorithm based on the Google Normalised Distance measure is used to calculate the *semantic similarity* between a query search, and the services discovered. This similarity is used to rank the discovered services to the mobile user. The context ranking includes device, environment, and user properties, with weights assigned to whichever is deemed more important.

The ALILI framework by Lomotey and Deters [83] used web services to bring file storage and sharing to mobile devices. The Mobile Web Services Mediation Framework by Srirama et al [131] implements enterprise service bus technologies to deliver features such as message compression, QoS guarantees and transaction support for mobile clients using web services.

A study by Le et al [73] compares various service description languages in terms of what features they provide, and suitability for a cloud environment. One of the largest current research areas is in the area of semantic service discovery. Such works are typically based on an extension to the Web Ontology Language (OWL), known as OWL-S. The "-S" indicates that the language is a variant for semantically describing services. OWL-S is made up of the following:

- *Service Profile*

  This is a human-readable description of what a service is, and what it does.

- *Process Model*

  Similar to WSDL concepts, this described how a client can contact a service, in terms of inputs, and what output can be expected.

- *Service Profile*

  Also similar to WSDL, this describes how the service should be called, in terms of the endpoint and protocols.

Klusch et al and [67] uses OWL-S to describe web services semantically. The focus of this work is on matchmaking, using a hybrid matchmaker developed by the authors called OWL-MX. OWL-MX uses an algorithm to determine if services are matched with the query, based on applying various *match filters* to the services. Filters include *exact match*, which means a service exactly matches the user query. Another filter is the *subsumes match*, which means that a service is more specific than the query. The opposite of this filter, the *plugin match*, means that the service effectively performs more work than requested by the query. To be more precise, these filters are defined in terms of inputs and outputs. User queries specify what inputs are sought after in the service, and the same for outputs. Subsumes match means that a service has fewer inputs and outputs compared to the query, and vice versa for the plugin match. An exact match is where inputs and outputs specified in the query are exactly the same as those in the service. The authors present algorithms for calculating degrees of match between query inputs and service inputs, and the same for outputs. Semantic similarity metrics from information retrieval techniques are used for calculations. Practically speaking, it is unlikely end users will define their needs for services in terms of strict input and output requirements.

Recognising the value of describing services semantically with OWL, a similar work that also used OWL, but without the OWL-S extension, is the work by Nagireddi and Mishra [90]. They developed a generic search engine for cloud services based on OWL ontologies. Cloud Service Providers (CSPs) place service descriptions into an "Intercloud Registry". These descriptions follow a given classification, describing attributes of a service. As a second action, the CSP inserts an XML-based ontology describing their service(s), into an "Ontology Directory". The search engine uses an "Ontology Keyword Index" to extract standardised terms from these ontologies. When the search engine receives a search query, it uses a query re-writing technique, to tokenise the query keywords, and then uses an unspecified method to map them to the standardised service terms that can be found in the ontologies, before performing matching. Compared to the work by Klusch et al [67], the ontologies used in this work, simply describe the service in terms of what it does, and how it provides its services; the cloud ontology used for service descriptions features

attributes such as the service payment type, security features, SLA policy, and licensing, rather than describing services in the traditional way of inputs and outputs as Klusch et al has. There is no implementation featured for evaluation.

Similar to the work by Klush et al, Suraci et al [133] developed the Context-aware Semantic Service architecture, which combines context awareness and semantic matching into the discovery process by means of filters. Given a query for a service (the nature of which is unspecified in the work), a basic filter is first applied to get a first effort list of possible matching services. This filter works by a protocol specified in the query. The approach is designed to be backwards compatible with existing service discovery technologies, such as Jini and UPnP. It would appear the user query should contain one of these protocols to indicate the kind of service being searched for. After this, a semantic filter is applied to the list of matched services. The user query, in addition to the protocol, contains a 'semantic part", for this purpose. The semantic part of the query, is matched to the semantic part of the description. Finally, whatever services survive this process, have a context filter applied. This matches the context of the service, to the context of the user. How these match filters work specifically is also not specified. An implementation or evaluation is not presented to illustrate these processes.

To summarise, the literature in this subsection shows that context awareness and semantic matching is seen as beneficial in the service discovery process. The overhead that semantic matching adds in addition to existing matchmaking algorithms is not explored in these works, considering the use of OWL as an enabling technology. OWL files describing semantic attributes of each service would have to be parsed individually to calculate the degree of semantic matchup, especially considering, as explored in the Context Awareness section, the use of an inference engine to parse axioms. Adding context awareness to service descriptions will also add additional overhead depending on the implementation of the matching. Rather than a separate file to be parsed as with the semantic works, context attributes may be added as part of existing service meta-data. Some context attributes may also be considered redundant. For example, if platform neutral technologies are used, then the attributes taken into account in the DaaS project and the MobiEureka framework, such as screen resolution and device properties, may not be relevant in a ranking calculation.

Furthermore, it is important to consider the architecture of discovery ap-

proaches taken. DaaS is focused on mobile services provisioned from mobile devices, but the discovery takes place in the cloud. While mobile devices as service providers is not consistent with the mobile cloud paradigm, especially from a user experience point of view, discovery can be a complex process, depending on the matchmaking algorithms used, and should not be executed on a mobile device. Additionally, a mobile device should not be responsible for holding and maintaining a service registry, as was the case partially in DaaS.

2.5.3 Discussion

Considering the widely viewed approach of mobile devices as enablers for the vision of ubiquitous/pervasive computing, and the multitude of network interfaces available for always available connectivity, the use of SOA web services from mobile devices is a very compelling proposition, especially with the wide variety of existing information and functionality already available from these services. The use of SOA services from mobile devices in the research literature has centred around middlewares which can make requests to services for the mobile user, using existing SOAP and RESTful service technologies. Such middlewares in early research have been deployed on mobile devices, but can now be hosted on cloud infrastructure. However, the use of these existing web service technologies, along with existing service description methods, such as the use of XML-based WSDL files, is currently not suitable for use in allowing ordinary mobile users to find and use existing services. As such, in the current state, these approaches cannot be adopted as a solution for end users in mobile cloud computing, considering the user experience goal.

The other main point that has presented itself from this literature review is the deployment of services; services deployed on the cloud, consumed by mobile devices, and services deployed on mobile devices, consumed by other mobile users. This thesis focuses on services in the cloud consumed by mobile users. Mobile devices as service providers runs contrary to the mobile cloud computing paradigm, especially considering the user experience. Mobile devices do not feature always on connectivity, they do not scale, and they do not have an endless power supply to support this model.

The cost of running a service discovery and execution model must also be addressed. Service discovery should take place on cloud infrastructure on behalf of the mobile device, and a service registry, which maintains service descrip-

tions, should also be co-located at the cloud infrastructure. There could be many services found to match up with a query, and applying matchmaking algorithms to each of these service descriptions can be time consuming. This can be become more of a concern when the added features of context awareness and semantic matchmaking are also considered and added on to the original matchmaking techniques.

2.6 Conclusions

The mobile cloud computing paradigm is the use of cloud-based computing infrastructure and services to overcome the physical resource scarcity in mobile devices. Utilising cloud-based resources from a mobile device is, in practice, a difficult paradigm to implement effectively; it is compounded by the realities of network disconnection, and limited power supply from the battery. While out and about, the low bandwidth and high latency of cellular networks also contribute to the difficulties in implementing effective solutions. This review has focused on exploring the most common methodologies in the research literature to implement a mobile cloud computing solution. However, these solutions do not address all the current difficulties presented. This fact was the primary factor for evaluation in the review. The goal of this thesis work is to present a solution to mobile cloud computing that addresses all of these on-going concerns, resulting in an integrated user experience for the ordinary mobile user.

The four primary solutions to mobile cloud computing take the form of Cloudlets, application partitioning and code offloading, remote display solutions, and middlewares. Each of these solutions present their own advantages and disadvantages in terms of energy and bandwidth utilisation, mobility management such as disconnection, suitability for the mobile context, as well as user and developer friendliness. Based on this review, from a user experience perspective, middleware solutions, which work with web services in the SOA model, are the most promising solution, especially as it is the only solution that adequately addresses mobility practicalities, due to the asynchronous non-blocking nature of the implementations.

This review also explored research literature in the area of context awareness on mobile devices, with a focus on how it can be adapted to mobile cloud

solutions. Context awareness allows mobile applications to personalise their execution and output to suit the situation of the mobile user. Various works were explored, which can be categorised based on their deployment model (on the mobile device, or in the cloud). Context collection, context representation, and context storage methodologies must also be considered for the mobile context. These can be very costly considerations in terms of computing resource requirements, and are more suited to a cloud deployment model for processing and storage. Representing context as ontologies, is seen in survey work as the most promising solution.

Finally, in recognition of the advantages of a middleware-based SOA approach to mobile cloud computing, this review also investigated literature in the area of mobile web services, to see how existing works could be adapted to the mobile cloud environment, and what extra considerations and concerns there may be. This review highlighted that a common approach is to use mobile devices as web service providers. This runs contrary to the aims of the mobile cloud computing paradigm, and instead, the focus of this thesis work, is on how mobile devices can consume web services deployed on cloud infrastructures. The specific aspects of solutions in this regard that are of importance to mobile devices consuming services, are service descriptions, service registries, service discovery, and service consumption. Existing solutions are not user friendly, and therefore do not meet the user experience goal of this thesis work. Users are kept out of the discovery process, as typical descriptions either consist of XML-based WSDL files, or developer documentation; both unsuitable solutions. Some of the work explored required the end user to know the location of the WSDL file in advance, and know the type of service, SOAP-based or RESTful. More dynamic and user friendly solutions are required.

In the next chapter, the insights from this review are used to form an architecture for a middleware based solution to mobile cloud computing, that makes use of both user contextual data, and web services in the cloud, to deliver useful information and services to the mobile user. The solution addresses the concerns and difficulties highlighted in this review, which are mostly neglected by other works.

Some of this review was included in the following publication: [98].

Chapter 3

A Middleware for Providing an Integrated User Experience of Mobile Cloud Applications: Design and Architecture

3.1 Introduction

The work in the literature review has shown that many factors combine to make the mobile cloud computing paradigm difficult to implement, which ultimately have a detrimental impact on the user experience. The varying quality and availability of network coverage, possibly resulting in disconnection, can inhibit communication between the cloud and the mobile device. The time and energy cost for communication of data over the network is also not negligible. Network operators charge a fee for transfer of data over their cellular networks, and if the energy used for communication is large, a drained battery will naturally render the device useless. Various approaches to solve some of these problems have been shown in Chapter 2, yet these solutions only tend to focus on one or few of the previously described problems. As the work in this thesis considers the user experience as the primary motivation, the solution developed takes a broad view of all the difficulties faced in the related works.

Providing an integrated user experience is an essential consideration, as mobile end-users with limited technical background are the consumers of mobile cloud applications and services. In the thesis Introduction chapter, an inte-

grated user experience of mobile cloud applications was informally defined as *an experience that provides seamless interaction between the mobile device and the cloud, with no detrimental impact or costs incurred by the device*. More formally, the aim of such an experience is to make the interaction between the user and the mobile cloud as seamless as possible, while rich new functionality and models of interaction are enabled by the mobile cloud. Such an experience must also confront all the aforementioned challenges. Client software of the mobile cloud embracing this experience observes the status and condition of the user and device, and intelligently caters its execution to the situation. Such an approach can foster better mobile applications running in the cloud, all optimised for the user experience, and benefiting from the device sensors for context awareness.

In this chapter, based on the conclusions of the literature review, a set of requirements are derived and presented that outline what desirable properties future work in this area should focus on to provide an integrated user experience. Secondly, a middleware solution is presented that can meet these requirements in solving the problems that have been discussed. This middleware solution is the subject of this thesis. The design and implementation, are also discussed.

3.2 Future Approaches to Mobile Cloud Integration

The related work in the literature review has shown that the user experience has not been a primary concern. In particular, each design focuses on one or a few aspects that are important to the user experience. Some of these solutions may contradict others. Aside from the user experience perspective, several of the works highlight challenges that will not be easy to resolve, such as the time and energy cost of profiling, offloading, and distributing of code and application components in the partitioning and code offload approaches. The Cloudlet approach is very dependant on the users position relative to the Cloudlet, along with the network state, and current technologies used in VM synthesis. In order to be adopted on a large scale, any future approach to mobile cloud computing has to be centred on the user experience. As such, a given solution will have to overcome all of the problems that were highlighted in the literature review, otherwise the user experience cannot be optimal.

The approach taken in this thesis to realising this vision and solving these problems, focuses on the cloud carrying out more intelligent work for the mobile user. By using more cloud-based applications and services in such a way that respects the goal of the integrated user experience, the mobile device is relieved of having to perform extra operations resulting in overhead, such as continuous GUI data transfer, frequent offload of application code, and network profiling, seen in the related work presented in Chapter 2. More applications and services are now developed and deployed in the cloud, taking advantage of the resources available from the cloud infrastructure.

3.2.1 Requirements for an Integrated User Experience

For the mobile user, the question is how to achieve this goal, while optimising the experience. By analysing the previous research work, a set of core requirements can be identified for a design that provides an integrated user experience:

1. *The approach has to address the latency between the device and the cloud infrastructure*

   As mobile devices are used more for applications and services that require real time data and actions, the latency between the cloud and the mobile device must be minimal. In the event that minimal latency cannot be guaranteed, approaches must be optimised and lightweight.

2. *The approach has to minimise bandwidth utilisation*

   The time and energy cost over networks such as cellular 3G, is typically high due to the low bandwidth. Furthermore, cellular network operators often charge per kilobyte of data transferred. The rate is even higher if the user is roaming. Therefore the utilisation of unnecessary bandwidth must be minimised, especially if the user regularly carries out tasks such as uploading of photos to social networks.

3. *Device workload overhead must be minimised*

   The actual workload the device carries out to enable such operations must also be minimal. An example is the profiling activities seen in other approaches. The energy drain required by such tasks runs contrary to the idea of the user experience.

4. *The approach must gracefully handle mobility aspects such as disconnection*

   As the user is mobile, they will switch between networks of differing quality, and will at times be disconnected. Applications and services used must be able to handle such occurrences by safeguarding any tasks the user was working on, and associated data.

5. *Provisioning for context awareness must play a central role*

   The unique ability of mobile devices to provide information about the user's context, such as physical, social, work, and environment, will be central to the development of future mobile applications and services, as their operation can be personalised to provide a greater user experience.

6. *The solution must uniquely cater for the mobile user, rather than the desktop user*

   Existing applications and services may not be optimised for the mobile user. They provide services that are costly to the user on a mobile network, and may be costly for the device to work on. They are designed with the desktop computer in mind. New approaches must consider the requirements of the mobile user, and the different input and output capabilities mobile devices provide to the user.

7. *The thin client on the mobile must provide an adaptive UI and services*

   The thin client of the mobile cloud must provide seamless access to applications and services on the cloud through a UI that adapts to the current state of the mobile and user. The client may also provide services to other local applications through an API.

8. *A standards-based solution must be used*

   Currently there are many different kinds of applications and services that can be used, all heterogeneous. They differ in regards to input, output and APIs. A common interface adhering to a standard defining how to describe different types of services and their utilisation must be developed and implemented in an automatic discovery service solution.

In the following section, a mobile cloud middleware solution is presented that can meet the requirements outlined, enabling an integrated user experience of mobile cloud applications.

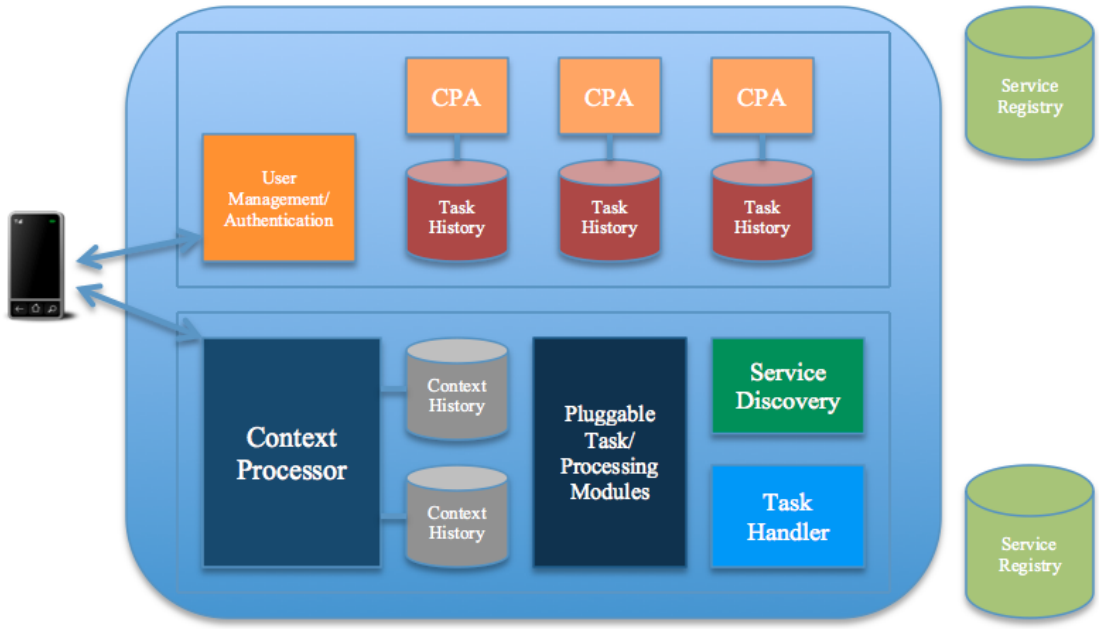


Figure 3.1: The main system architecture of the Context Aware Mobile Cloud Services (CAMCS) middleware.

3.3 CAMCS: Context Aware Mobile Cloud Services

The approach taken in this thesis work is a cloud-based middleware system sitting between the mobile user, and cloud-based applications and services. This system is known as Context Aware Mobile Cloud Services, CAMCS - see Fig 3.1. The main goal of CAMCS is to provide discovery and access to cloud based applications and services for the mobile user. Existing services deployed in various clouds can include third party applications such as e-commerce, entertainment, productivity, and services that enhance the capabilities of mobile devices, by providing access to cloud based resources, such as storage.

3.3.1 Design

CAMCS has several features that specifically cater for enhancing the user experience with respect to the requirements that have been outlined. Specifically, CAMCS consists of several key components:

- *The Cloud Personal Assistant*

  The Cloud Personal Assistant (CPA) runs within CAMCS, and can carry out tasks asynchronously for the mobile user on his/her behalf. The as-

sistant can discover and invoke services to carry out these tasks, receive a result, and save it in the event the user has become disconnected. When subscribing to CAMCS, each user receives his/her own CPA. The CPA is further described in subsection 3.3.2.

- *Task History*

  The CPA has access to task and service history that has been utilised in the past, which it can use to automatically make intelligent decisions about future actions it can carry out on the users behalf, with minimal intervention required from the user. History data shows preferred services and providers, what inputs are typically used, and what times invocation occurs.

- *The Context Processor*

  The Context Processor is another cloud service that handles context aware information from the device. The mobile user chooses how often to send new context data from the device. It can locally infer context from these resources of information and cater for the responsibility of passing context data to the applications and services that require it.

- *Context History*

  The history of inferred context can contribute to a new context as it is gathered. This is also useful if new or complete context cannot be determined.

- *Discovery Service*

  CAMCS incorporates a discovery service, which can be used by the CPA to discover cloud applications and services on behalf of the user. The discovery service uses registries of cloud applications and services, which can provide efficient discovery, along with service ranking based on user preferences.

- *Pluggable Modules*

  Plug-ins can be developed and added into CAMCS to further enhance operation with custom functionality.

3.3.2 The Cloud Personal Assistant (CPA)

CAMCS is based on a previous standalone middleware project by the thesis author, known as the Cloud Personal Assistant (CPA) [96]. In the project work for this thesis, the CPA has become a component of the larger CAMCS middleware. The CPA itself is designed to complete tasks for mobile users, by working with cloud-based services, in an asynchronous, disconnected fashion. Each user of CAMCS assigned their own CPA, which performs the work required to complete user-assigned tasks, on behalf of the mobile user. A CPA can be thought of as the user's personal, trusted third-party representative in the cloud. The use of CPAs is the backbone behind CAMCS.

A mobile user communicates with their CPA using a thin client application, the CAMCS Client, which is installed on the mobile device. The mobile user creates a task to be completed using cloud services on the mobile device, using a task name, and a description. The description is used as a query for service discovery, when CAMCS searches service registries for cloud services. The details regarding the structure and implementation of tasks, is presented in Chapter 5. The mobile user can then choose from the selection of services discovered by the CPA, whichever service they feel is most suited for the task. The mobile user then provides various parameters required to invoke the service (for example, if the task is to book flights between Dublin and London, and a flight booking service such as Ryanair was used, parameters could include the departure and arrival airports, and a date for travel). The CPA then calls the service (typically of SOAP or RESTful web service type), passing the parameter information provided by the mobile user. The service responds with a result, which is saved with the task by the mobile user's CPA. The mobile user is notified of the task completion, and then can view the result on the CAMCS Client on his/her mobile device, at whatever time is convenient. Figure 3.2 shows a diagram of the general flow of steps in completing a new user task using a CPA, within CAMCS.

The main focus of the CPA is disconnected operation. After the mobile user has created the task and sent it to the CPA, the CPA itself assumes full responsibility for task execution. The mobile user is free to disconnect from the cloud (and CAMCS), and all task progress and results, are stored with his/her CPA. Should the mobile user lose the signal, or should the device battery die, the result of tasks will be waiting for them with their CPA whenever they can re-connect.

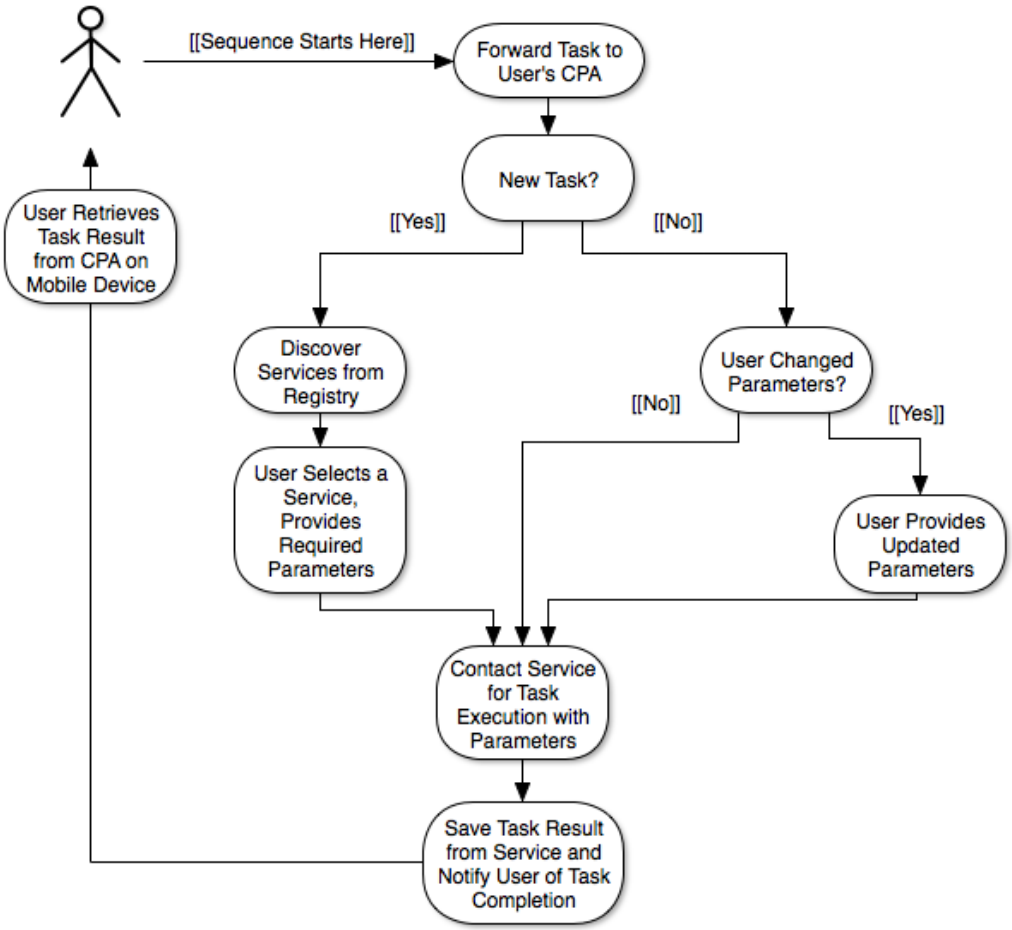


Figure 3.2: A task flow diagram showing the steps required for completing a task sent to a user's CPA, within CAMCS. For clarity, contextual-awareness, which can be used with both service discovery and task execution through the Context Processor, is not shown.

The only interaction required by the user, aside from task creation, is the selection from the discovered services of a suitable service for completing the task, and providing the parameters. The user is notified at each step, and can provide this information whenever they are available. The CPA will store the chosen service, and the parameters provided, for the future. This brings in the storage of Task History described previously, along with the distinction between creating new tasks, and re-running older, completed tasks. Any task that has been created and executed before, can be selected to run again by the mobile user. In this case, the user does not need to select a service, or provide input parameters again (unless they indicate they want to change service, or modify stored parameters). From the CAMCS Client, the mobile user can access the task history, and signal any task to run again.

Re-running a task will typically be quicker than creating a new task, as service discovery also does not need to occur again. This scenario is the more likely use-case with CAMCS; a user will have a set of common tasks they perform each day, such as retrieving their daily schedule in the morning, along with checking the traffic on the road to work, and the weather. The mobile user need only create these tasks, choose the services, and provide the input parameters once, and from then on, can re-run the task every day without performing these steps again. The CPA can also run these tasks automatically on behalf of the user, without an explicit request. Task models such as this, are presented in Chapter 5.

A CPA, as a provider of services to users, may also be compared to service broker approaches that have been described in Chapter 2. Existing service broker approaches are not suitable for the research goal of an integrated user experience that the CPA can provide with CAMCS. Existing solutions, such as CORBA [129] and Jini [114], require that the service consumer (the CPA in this scenario) would already possess the service interface for the service that the mobile user would eventually select for completing his/her task. These solutions are also specifically for use by software applications, where discovery takes place invisible to the user, preventing the browsing of discovered services, and then choosing which service to use for completing a task. A far more dynamic approach is required, where service discovery and communication can occur at runtime without prior knowledge of a standard interface. Additionally, while a broker could adopt the work that is presented in this thesis, the traditional interfaces and XML descriptions of services cannot be understood by ordinary mobile device end users. This is also made more difficult by the lack of standards for interfaces of services; this prevents different services providing the same or similar functionality from being directly compared. This problem is the subject of Chapter 5.

3.3.3 CPA Components and User Interaction

Within CAMCS at a more detailed level than presented in the CAMCS architecture diagram, there are several components that support the operation of CPAs. Figure 3.3 shows an overview of these components, which are divided up into three layers; the *Management Layer*, and *CPA Layer*, and the *Execution Layer*.

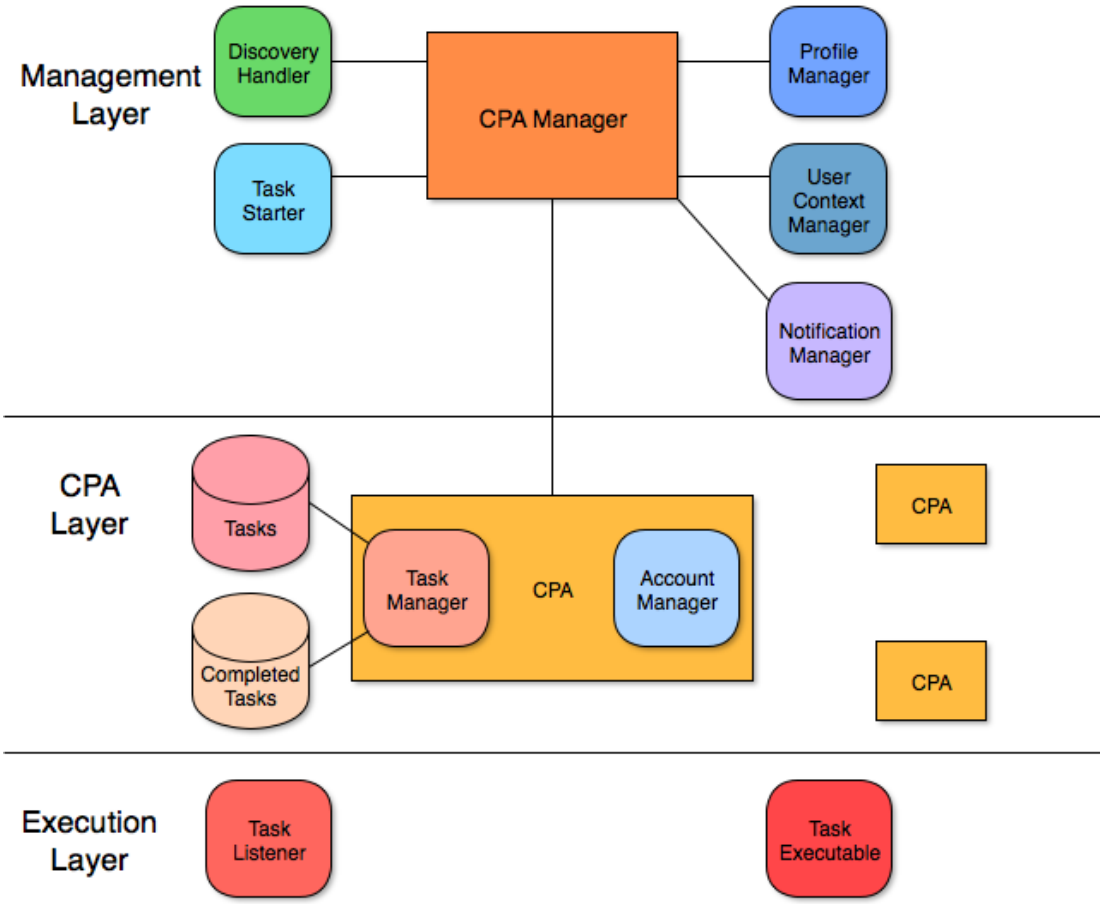


Figure 3.3: A detailed view of the components of CAMCS, which can be divided into three layers. The Management Layer contains many components shared by all CPAs. The CPA Layer contains the CPAs, along with components that each CPA has a unique copy of, and finally, the Execution Layer, containing components used during task execution.

The Management Layer contains common components shared by all CPAs to support their operation. Some of these CPA components can be thought of as interfaces to allow individual CPAs to communicate with a specific larger component of CAMCS, such as the User Context Manager, which just interfaces CPAs to the Context Processor. The other CPA components are not interfaces, and wrap their own functionality, such as the Notification Manager.

A CPA uses the components in the Management Layer through the CPA Manager (as singletons). Every CPA has a reference to the CPA Manager. The CPA Manager is also used for authenticating user access to CPAs when requests are received from the mobile device. User access to CPAs is mediated by CAMCS,

with username and password basic clear-text authentication over HTTP.

- *Discovery Handler*

  The Discovery Handler encapsulates all functionality related to Service Discovery, as seen in the CAMCS system architecture diagram. The CPA passes the user-provided task description to this handler, which contacts the service registry using this description. The registry returns a list of discovered services for the task, to the CPA.

- *Task Starter*

  Once the mobile user has chosen a service, and provided all the input parameters, the CPA passes this information to the Task Starter. The Task Starter creates a Task Executable (discussed in Chapter 5) for executing the task, and passes a map containing the user provided parameters.

- *Profile Manager*

  The Profile Manager is related to the Context Processor, allowing the user to switch between Context Profiles, which are discussed in Chapter 4. Context Profiles can determine what behaviour a CPA should undertake under its own initiative.

- *User Context Manager*

  The User Context Manager is effectively an interface allowing a CPA to query and retrieve user context data, stored with the Context Processor component, when required for service discovery, or task execution.

- *Notification Manager*

  The Notification Manager is used to send Push-style notification messages from a user's CPA to the mobile device, to inform him/her of some event, such as "Discovery Complete", or "Task Complete".

The CPA Layer contains a CPA for every user. The data for all CPAs is stored in a backend database. Every CPA contains its own unique copy of the following components:

- *Task Manager*

  The Task Manager stores a CPAs *Current Tasks*, and *Completed Tasks*. The completed tasks make up a CPAs Task History. All operations, such as

creating new tasks, storing of selected services and input parameters, and retrieving task results, take place through the Task Manager.

- *Account Manager*

  The Account Manager stores a user's credentials for connecting their CPA to third-party cloud services, which are used for features such as Context Awareness, and with some of the pluggable modules. The Account Manager also contains the operations to contact those services, and push/pull information to/from them. This is presented in full in Chapter 7.

The Execution Layer at the bottom, contains all the components related to task execution on behalf of CPAs:

- *Task Executable*

  Task Executables are used to run tasks, and are created by the Task Starter. All information related to running a task, such as the selected service, and the user's input parameters, are passed into the Task Executable as a map. The Task Executable is the component that makes the remote call to the cloud-services over HTTP, passing the parameter information, and it also prepares the result data retrieved back from the service for storage by the CPA. The Task Executable implementation is discussed in more detail in Chapter 5.

- *Task Listener*

  The Task Listener, as the name suggests, is a listener which listens for events in relation to task execution, such as task completion, moving a multi-step task (known as a task using a *Service Flow*, discussed in Chapter 5) to the next step, or a task error. It notifies the CPA of these events (so that in turn, the CPA can notify the mobile user with the Notification Manager). It is also responsible for passing the task result, from the Task Executable, back to the CPA for storage with the Task Manager.

A flow diagram showing how the execution of a new task uses the components in these three layers, is shown in Figure 3.4. This middleware architecture and component design was derived from the need of a middleware solution that was user oriented, to ensure user adoption. The components are centered around CPAs, and the functionality that they require to complete task work for the user in a disconnected, asynchronous fashion. An alternative design

Table 3.1: A summary of how the common approaches to mobile cloud computing meet the user experience requirements outlined in this chapter, including CAMCS. √ indicates an approach solves a problem, ∗ indicates dependence on the individual services.

	Cloudlets	Partitioning	Code Offload	Remote Display	Other Middleware	CAMCS
Latency	√	√	√			√
Bandwidth			√		N/A	√
Device Overhead		√	√	√		√
Disconnection					√	√
Context Aware					∗	√
Mobile Native Solution		√	√		√	√
Adaptive Solution		√	√			√
Standards Solution					√	√

consideration was to use an agent-based approach. However, agent-based solutions typically consist of agents sitting in a pool, waiting to complete only one type of task upon request, rather than a personalised dynamic solution that could be used to complete a range of tasks. Consideration was also given to using specific cloud-based applications as use-case examples, which would deliver functionality to mobile devices through the web browser, rather than the use of a native application. However, once again, applications based on specific use-cases are not dynamic in the range of services that can be offered to the user. The use of a native application on the mobile device was also desirable in order to obtain full un-hindered access to the device APIs.

In the next section, an evaluation takes place to describe how CAMCS can meet the requirements that have been outlined for an integrated user experience.

3.4 Evaluation

CAMCS can address the requirements outlined in this chapter, and therefore provides a comprehensive solution for many of the problems outlined in previous projects, while placing user experience as the primary motivation behind the design and approach decisions - see Table 3.1.

- Latency shall not be noticeable during interaction with CAMCS because of the low-volume data that will be transferred between the mobile device and CAMCS. Task data will consist of textual data of small size. As a result of the disconnected nature, there is no need for continuous data transfer which can make latency more noticeable to the user, and additionally, there will be no real-time requirement in this work. Interactivity with solutions like remote display and middlewares, do not take latency

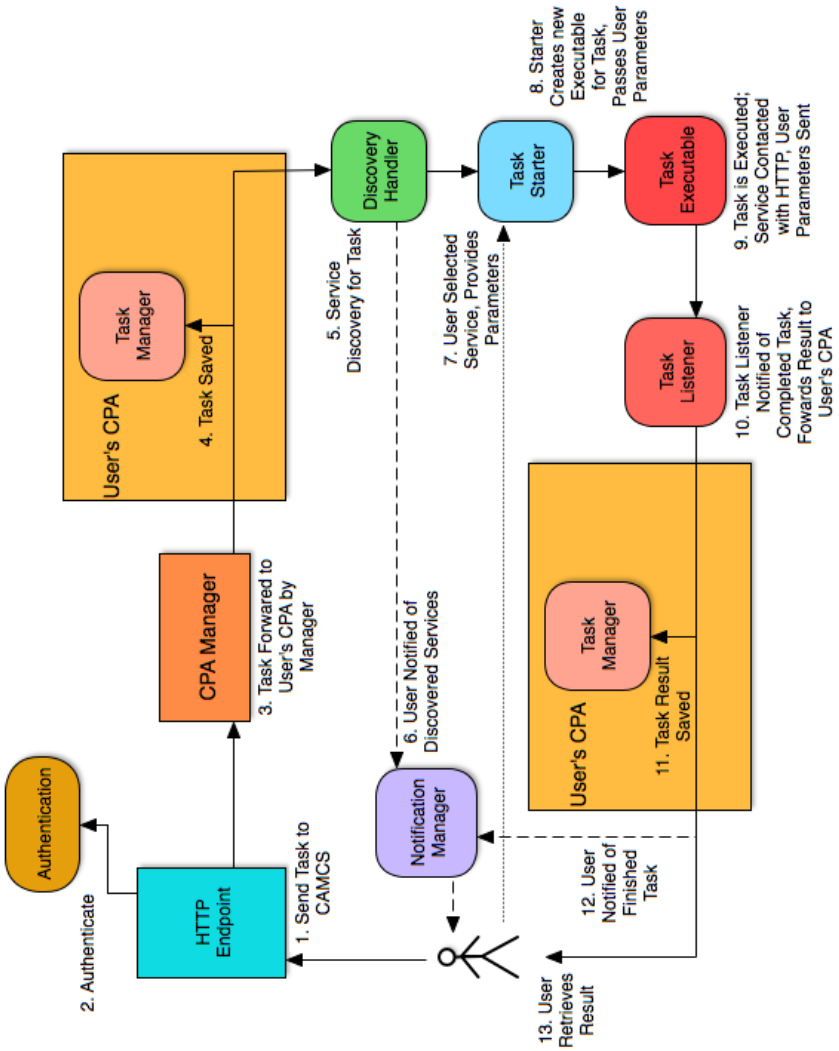


Figure 3.4: A sequence diagram showing how the components of CAMCS are used by CPAs to complete a task. The solid lines show the main flow of steps through the system. The dashed lines show steps where the user is notified of an event using the Notification Manager. The dotted line shows where the user provides more information before the main flow can proceed.

into account.

- Bandwidth utilisation is low, as the user only needs to signal the CPA to carry out work and pass required information, which has already been stated to be in the form of textual task descriptions. VM state, entire mobile applications or code-bases, do not need to be transferred between the mobile device and CAMCS, as in other work. To minimise the workload overhead on mobile devices, the CAMCS Client is used on the device to communicate with CAMCS in the cloud. Solutions such as Cloudlets and remote display, both require large-volume data transfer because of the VM-based approaches used.
- CAMCS uses applications and services deployed in the cloud rather than on the mobile device; resource intensive applications are not executed on the device, which reduces its workload and in turn, will benefit from time and energy savings. The mobile user does not directly interact with a service through their mobile device, as this interaction is delegated to his/her CPA. The mobile device is only responsible for running the CAMCS Client application, which is a thin-client, rather than a fat-client. Not all approaches reduce the device overhead; for example, Cloudlets require the device display to be active at all times for the user to interact with the VM running on the Cloudlet.
- CAMCS can address mobility concerns such as disconnections by storing task results for the user until he/she has an Internet connection to connect to their CPA. A CPA in the cloud carries out all the complex work of discovering and invoking services, freeing the mobile device to perform other work. All the solutions presented in the related work, with the exception of middlewares, cannot tolerate disconnection.
- The ability of CAMCS to include context awareness support will enhance the user experience. It can gather various new contexts in suitable conditions, and infer context from task and context history, all the while taking advantage of the cloud resources such as data-stores and processing power. Gathered context data can be used to personalise both service discovery, and service execution, so that the result of the task is relevant to a user's situation. Cloudlets and remote display solutions, based on desktop operating systems, are not designed to take mobile contextual data as input.

- CAMCS is a middleware designed to serve mobile devices. Specifically, it optimises the presentation of service outputs to the mobile device through the user's CPA, such as optimised request/response data formats using JSON markup, and communication architectures with REST over standard HTTP. Where possible, platform neutral technologies are used.
- The CAMCS Client adapts its operation based on the state of the mobile network at any given time, by delaying tasks being sent to the user's CPA until such a time that the predicted network quality has improved. This is presented in Chapter 6. Solutions such as application partitioning and code offloading, also observe the state of the network before a decision is made to offload, but the methodology the related works use to perform this profiling result in an overhead. Other solutions do not observe the network state.
- CAMCS and other middlewares are the only solutions that can claim to use standards-based solutions. This is because CAMCS, and most middlewares, work with cloud-based services deployed using standard SOA patterns and technologies, whereas other approaches require applications to be modified for compatibility with the proposed solution, mainly with application partitioning and code offloading solutions.

The requirements outlined and evaluated above, do not exclude individual applications from being feasible as a mobile cloud computing solution, if for example, one requirement is not met. It is up to the software developer, or the application itself to mediate how a resource is used. For example, if an application is susceptible to latency, or requires high-volume data transfer, then the mobile user should be made aware that the preferred network connection for operation is a Wi-Fi network, rather than a cellular network. The mobile user should not expect the same level of a user experience over the cellular network in such a case.

In Chapter 8, where CAMCS is evaluated, the requirements will be revisited to determine if CAMCS conforms with them.

3.5 New Mobile Cloud Usage Scenarios and Enabled Features

Several new kinds of applications and services can be enabled by the CAMCS approach to mobile cloud integration.

- *Real-Time Context Aware Services*

  Real-time services, which can respond to the users current and inferred context, possibly from context history, can be enabled by CAMCS. Such applications and services can actively or passively recommend activities or a course of action to users based on context, such as reminders or information about tasks they have to carry out based on location context. For example, the purchase of items from a shopping list, or if in an unfamiliar location, can suggest places of interest based on past activities. In the case of the shopping list reminder scenario, the CPA could actively check store stock for the user, and if unavailable, direct to a new store nearby that has the desired item(s) in stock, without user intervention. As another example, based on the current time of day or even year, such as during the day during school-term, it can notify parents to a routing of least traffic congestion to a school. This would stop occurring out of school term. The CPA can actively consult with traffic services, and perhaps could even make the information available to other applications or devices in the car such as a SatNav. This kind of application is explored in chapter 4.

- *Collaborative Services*

  Each CAMCS user will have invoked and discovered different services. In a vision of the next generation of mobile applications by Sankaranarayanan et al [20], they propose collaborative applications and services that share information to provide a seamless user experience, rather than the current generation of applications and services that work in isolation. An example the authors used was a traveller attending a conference. A calendar application with details of the conference such as location and time/date, could share this information automatically with airline and hotel services to find the best flights and accommodation to a location near the conference venue. CAMCS could enable such an approach, if the CPA can share details of discovered services with each

other. This could be further enhanced if CPA users who share some relationship such as friends or work colleagues are made known to each other, in which case they share similar interests and context. Any collaboration scenario would be subject to strict privacy control. This is explored briefly in Chapter 7.

- *Additional CPA Services*

  The CPA concept can be expanded so that the CPA can carry out its own work, rather than simply looking up and invoking services. For example, the CPA can be extended to provide additional applications for the user, such as data processing and synchronisation with multiple services that the user is interested in, such as the synchronisation of data among social networks or sharing of files in a workplace domain. Some of these application models are also explored in Chapter 7.

- *Extensible Middleware*

  The CAMCS functionality can be extended by adding in components on a plug-in basis to carry out other tasks on behalf of the mobile user, possibly utilising cloud based services, and taking advantage of the cloud infrastructure. Such functionality could include support for computationally intensive batch processing jobs, or a client for long running queries on databases within a workplace or scientific domain, such as has been undertaken with the additional CPA services described previously.

- *API for Developers*

  As CAMCS is a cloud-based middleware, it shall be deployable on any cloud servers supporting the appropriate language runtimes. For example, it could be deployed within the private cloud of a company to provide internal workplace services to employees through their mobile devices. The primary interaction with CAMCS will take place through RESTful endpoints, providing an API for CAMCS to perform various tasks. A developer can extend the RESTful API endpoints to target new (or existing) functionality offered by CAMCS, such as access to new services as previously described, and hook into user CPAs via the CPA Manager.

- *Performance Enhancing Proxy for Mobile Clients*

  In addition to extending the CAMCS middleware with additional appli-

cation and service models, CAMCS can also serve as a more traditional performance enhancing proxy, sitting between the mobile device and cloud-based services. Middleware services such as data caching, content delivery, and compression of HTTP payload data, are examples of features that can be provisioned to mobile clients.

3.6 Conclusions

Future approaches to mobile cloud computing will need to tackle several of the issues that are dominating the current work in this research area. After determining a set of requirements for future mobile cloud works to provide an integrated user experience, the Context Aware Mobile Cloud Services (CAMCS) middleware was introduced, which can meet these requirements. User interactions with CAMCS for completing tasks, mainly through the use of each user's Cloud Personal Assistant (CPA), were presented. The design of CAMCS was presented, including the structure of the middleware, and the various components it uses to complete its work. It was shown how these components are used for the purposes of completing user tasks. Finally, the CAMCS approach was analysed to determine how it can meet the user experience requirements outlined, and new application scenarios that CAMCS can enable, were presented.

In the next chapter, the implementation of the Context Processor component of CAMCS is presented. This will show how user context data is collected from the mobile device, and how a user's CPA can use this data with cloud-based services to personalise their results to the his/her situation.

This chapter was based on [98], with the following exceptions: Subsections 3.3.2 and 3.3.3 are new, and Section 3.4 has been modified. Various modifications have been made to the text throughout the chapter, to fit with the flow of the thesis. Some of Section 3.3.3 will appear in [101].

Chapter 4

Mobile Cloud Contextual Awareness with the Cloud Personal Assistant

4.1 Introduction

The use of context awareness, as identified by the MCC requirements in Chapter 3, enable a personalised user experience of the mobile cloud applications. Context awareness provides information about both the users themselves, such as their current activity, along with location, time of day, week, and year. It also provides information about the state of the mobile device, such as the remaining battery level, and presence of a network connection along with its quality. In this thesis work, the CAMCS Client application is responsible for collecting the context information from the user, which is then sent to, and stored with the CPA of the user in the cloud.

The CPA can then use contextual information in multiple ways, such as to aid in discovering a suitable cloud service for completing a task, depending on the user's preferences. Context profiles can model daily situations of the user, such as a home profile, and a work profile. Different context profiles can be active for the CPA of a user. Based on the active profiles, the CPA can undertake different work, and it can influence its operation, such as in its choice of cloud services to complete given tasks. Additionally, a history of gathered context is stored with the CPA of the user, so that if the mobile device becomes disconnected, new context can be inferred by querying the

historical context for the same time period in the past, such as time of day, and day of the week. This enables additional functionality. The context can be used such that the CPA can intelligently work under its own direction, rather than the user having to explicitly request that it carry out some work. For example, the CPA can choose to repeat a task if the current context matches with the historical context when the user last explicitly ran that task. Taking these scenarios into account, by using the contextual data, service execution in the cloud is personalised and catered to the situation of the user and the current task.

The CAMCS architecture features the Context Processor, which gathers the contextual data from the mobile devices, stores it, and provides it where required to tasks. It also uses a context Inference Engine to derive new context from the stored historical context. This chapter describes the implementation of the Context Processor component. This chapter will show how contexts are structured and used by the Context Processor within CAMCS, along with how other developers can extend the functionality by adding new custom contexts. In addition, examples of two experimental mobile cloud services that the CPA can use, along with the gathered context, are given. They demonstrate how the CPA can use gathered context to operate with such services, to complete tasks. These services are based on two context profiles, a tourist context profile, which makes use of the Foursquare API [43], and a vehicle context profile, which makes use of the Twitter API [25].

Many different approaches have been taken to collect context information from the mobile device, along with use of this context with mobile applications, as discussed in Chapter 2. The contribution of this work is that the focus is not only on how CAMCS handles the context data, but how it can be provided to the CPA to be used with mobile cloud services, how it can be used in different ways, and how it can influence the operation of the CPA to cater to a more personalised and situation-relevant experience for the user, with the help of context profiles. This is extendable for other software developers.

4.2 Context Representation Requirements

Several considerations had to be taken into account when selecting what technologies could meet the requirements of the Context Processor, and how the

CPAs within the CAMCS middleware could make use of the context for completing user tasks. These considerations are now briefly explored.

The choice of how to store, represent and infer context in the Context Processor, was crucial for the operation of the middleware, due to the disconnected nature of the middleware from the mobile device. As described in Chapter 2, Strang and Linnhoff-Popien [132] carried out a review of the different technologies available to store and represent context. From this review, they recommended the approaches that would carry most advantages under the different aspects they studied, included using *ontologies*, in the form of the web ontology language (OWL) or an Object-Oriented approach, to describe context. As part of the design process, two different technologies were evaluated, which implemented each recommendation, the OWL API [54], a Java API which is a Java implementation framework which can work with OWL, and the Java Context Awareness Framework (JCAF) [8], which is an Object Oriented approach to context representation. Each of these approaches are now evaluated in regards to the requirements of CAMCS and the Context Processor.

4.2.1 Context Representation

Ontologies are commonly used to build a *web-of-data* in the vision of the *semantic web*, by representing meaningful relationships between different data resources known as entities. The *Web Ontology Language* (OWL), is a W3C standard for representing ontologies in the semantic web [104]. As it is used to represent relationships between entities, it is seen in the research literature as a viable solution to represent user contextual data. The OWL API can be used to represent context as a document, marked up in XML. It supports various different OWL syntaxes such as the RDF syntax, and the Manchester syntax. An Ontology contains various different types of entities and axioms, which can be used to represent user context. OWL is made up of classes, properties (object and data properties), and individuals (named or anonymous), which are used to represent the entities and axioms in an ontology. Different entities are bound together by logical axioms, which can be used to infer relationships, in the form of *linked data* [22]. For example, a user of the system can have a data location property, with a value of Cork, Ireland. That could also be represented with a more complex, object data property. A location object can have a name, latitude and longitude, and a timestamp; this can be joined with the

user as an object property (more specific examples are given later). The entities represent linked data, as they can be linked in RDF by URIs, for example, to define their relationship through the object property, which knows the URIs to both the user and location object entities.

JCAF represents context in an Object Oriented manner. JCAF is an actual Java program that one runs on a system. The framework features an Entity class, a Context class, a ContextItem class, and a Relationship class. Each user of the system could be represented as an instance of the Entity class. Each Entity has one Context object, and the Context object contains several ContextItem objects, which represent the contextual data. Each ContextItem object is indexed by a Relationship object, describing the relationship the ContextItem has with the Entity. A User Entity could have a ContextItem in its Context, representing Cork, Ireland, indexed by a Relationship object, describing a Location property. Context Clients then query the Context of the Entity, normally using RMI, to obtain the context.

4.2.2 Context Inference

Being able to infer context is crucial to determine the values for a given context from within the currently stored context. Inference is also used to examine axioms to infer new context, if new context is not available. The aim of CAMCS being the user experience, and the realisation that the mobile device will not always be connected to the cloud, implies that new context data will not always be available, so the inference capability is crucial.

The OWL API provides an OWLReasoner interface, which is implemented by an Inference Engine. Several open source inference engines are available which implement this interface. Using the OWLReasoner, one can query an ontology in a document, by specifying for what data or object property should be returned, and the values for those properties are received back from the reasoner, as a collection of literal values (Integers, Floats, Strings). The OWLReasoner can also look at an ontology representing the context history, and given a data or object property expression, it will return all axioms that evaluate to true in a given ontology within a closure. This can occur whenever new context cannot be retrieved from the user. Unfortunately, JCAF does not support any form of context inference in its current form. The context can only be queried and not inferred, although this could be implemented using a third party inference en-

gine. Unlike the OWL API and the OWLReasoner interface, it does not provide any interface for reasoners to hook into the ContextItems and Relationships.

4.2.3 Context Storage

While the current context of a user could be stored temporarily, the context history must be stored for a longer period of time. The solution must consider that large amounts of contextual data could be collected over a short period of time.

The OWL API has the ability to write out ontologies into XML files, marked up in various supported syntaxes. These files can be loaded by the OWL API for parsing, reading, and writing. The files themselves can be stored on any cloud based file system for access by the Context Processor.

JCAF does not use any form of storage beyond the running time of the JCAF Context Service. Once the program has terminated, all Entities and their Contexts are lost.

4.2.4 Summary

Based on the requirements above, the OWL API was chosen for the implementation. It is worth nothing that the choice of taking an ontology-based approach was also the recommended conclusion reached by Strang and Linnhoff-Popien [132].

4.3 Context Processor Design and Architecture

The Context Processor is the central component of the CAMCS middleware that collects and stores context from the mobile devices of users, and provides context to context-consumers (user CPAs) when requested. Figure 4.1 shows the Context Processor components. Each CPA has a Current Context, and a Historical Context. Each of these is stored as an ontology in two respective XML files; these files are stored in the cloud file system of the deployment platform, and are identifiable only by the randomly generated user ID from the CAMCS registration process. A CPA cannot access the context data of another

user, as the user ID is cross-checked with the context data being requested. A database approach would have made it difficult to structure context, along with the relationships denoted by axioms. The Context Manager handles different context read/write operations and logic. The Ontology Manager parses and writes to the XML files.

The current context always stores the most recent data for a given context. The context history will store previous context for a time-period, such as a week, or a month. A user can also purge some/all collected context data anytime they choose. When a context update is received, the old current context is moved into the context history, and the new context information from the update replaces it as the new current context. Different types of context can be updated at different times.

Context updates occur through the CPA of a user. When the mobile device sends a context update, this is first sent to the user's CPA. The CPA then contacts the context processor to store the context. Whenever the CPA needs context for a task, it sends a Context Request to the Context Processor, which in turn will consult the current context. If the required context is available, this will be returned to the CPA. If the particular context is not available, or is found to be stale, the context history is consulted and queried if the request permits the use of historical context. The results will be forwarded to the CPA. Architecturally, the Context Processor exposes a standard interface for both storing and requesting context, within the Context Processor. The design of the Context Processor provides an interface for interested Context Consumers (in the case of the CAMCS middleware, the consumers are the CPAs). This way, if another solution was selected to model the context data, the CPAs will not need to change the way they request and use context.

The Context Processor is designed to be extendable, so that developers can create their own contexts if required. This allows for the quick creation of new contexts for new services and experimentation. The capabilities for extension are presented in Section 4.5.

Following, the process by which context data is sent to the Context Processor is described, to give an overview of the relevant architecture and components. Then, the process of how context is requested and used by a CPA is described, for the same purpose.

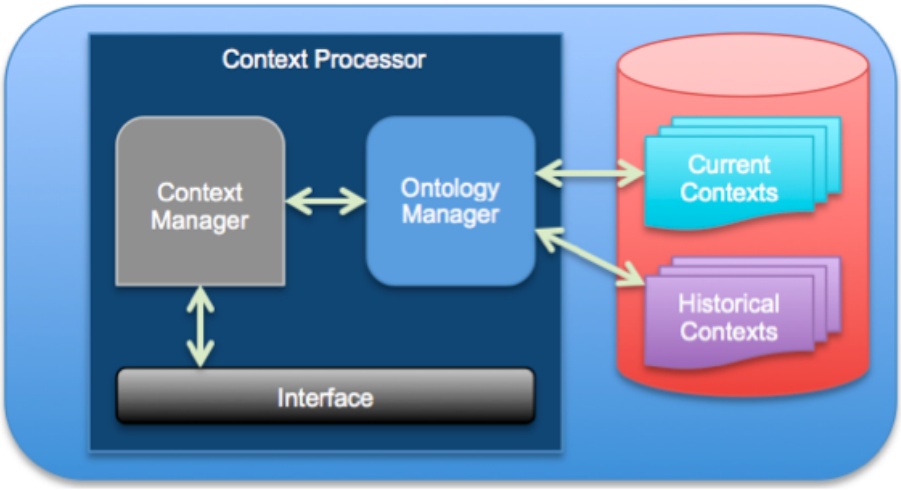


Figure 4.1: The Context Processor within CAMCS. It features an interface for the CPA of a user to request and store context. The Context Manager handles the different context operations. The Ontology Manager reads and writes context to the Current and Historical context ontology files on the cloud storage.

4.3.1 Sending User Context to the Context Processor with the Context Wrapper

The CAMCS Client running on the mobile device can gather context from various sources, such as the sensors, accelerometers, the gyroscope, and the clock. This context is collected at intervals specified by the user in the Settings of the CAMCS Client. Of course, the user can choose to turn off all context collection in the settings, if they do not wish to use any context-driven services. No context is collected which can individually identify the user, or the exact mobile device used (such as IMEI and phone numbers). Assuming context collection is enabled, these contexts are wrapped up into a Context Wrapper. This is a generic descriptive class that takes each context, and sends it to the CPA of the user. The communication between the mobile device and the CAMCS middleware follows a RESTful architecture. Once the CAMCS middleware has been contacted, the contextual data in the wrapper is forwarded to the CPA of the user. The contexts are unwrapped into their various classes. In the architecture, there is an abstract Context class. Each context is built upon a concrete subclass of Context, which is described in more detail in Section 4.5.

The architecture also features a ContextUpdate class. All context update requests are represented as classes, which extend the ContextUpdate class. The CPA prepares a ContextUpdate object containing the context update data from the wrapper. The CPA then contacts the Context Processor, providing the Con-

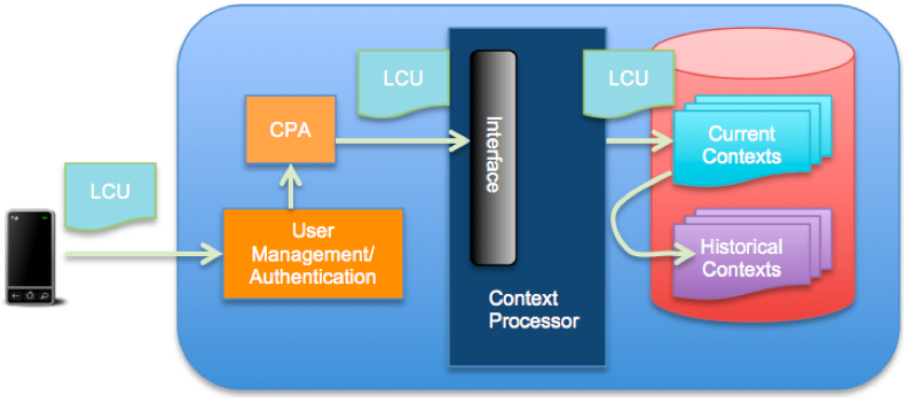


Figure 4.2: Receiving of a Context Update. The Context Processor receives a Location Context Update (LCU), from the user's mobile device, via the CPA of that user. The context update is written to the current context Ontology of the user. The old location context is moved to the historical context.

textUpdate object, through the standard, simple, interface. The Context Processor determines each type of ContextUpdate (for example, location context, user activity context). The Context Processor then writes the new context data into the Ontology in the XML file - see Figure 4.2.

4.3.2 Consuming Stored Context from the Context Processor with the CPA

Whenever the CPA needs to carry out some task with a cloud service that can benefit from knowing a particular context, it simply sends a ContextRequest to the Context Processor interface. ContextRequest specifies the required context by the name assigned to that context within CAMCS. When the Context Processor receives the request, it uses the inference engine to gather the requested context from the ontologies within the XML documents (current and history) as required. The specific context is returned as an object that extends a concrete subclass of the abstract Context class (more details of this process are described in Section 4.5). See Figure 4.6 in Section 4.6 for a graphical example, related to the use-case example given in that section.

4.3.3 Discussion

It is clear that an object-oriented approach, similar to other works, has been taken. One can understand why originally, using JCAF would have been the

first choice, as another object-oriented approach to context representation, but as described, it did not meet the requirements for inference and storage. While context is represented within the Context Processor as extendable Java objects, the representation, storage, and inference takes place in the ontology XML files. The object-oriented abstraction in the code makes it easier to handle at the programming level. One may argue that asking developers to create their own contexts is putting much burden on their time. However, context can only be exploited when meaningfully described as a compound, complex context; that is a context made up of several other contexts, already provided by the Context Processor (e.g. location, time). This way, contexts can be personally adapted accurately to the situation. The Context Processor hides away all the ontology and XML details of the contexts, so the developer need only work with high-level objects. Once the required contexts have been defined in CAMCS using the provided constructs described in Section 4.5, the developer need only work with the Context Processor through the interface.

4.4 Context Processor Implementation

Following, the implementation of the system is discussed. During the work for this chapter, the cloud server used for experimentation was located within University College Cork; the server has a 1.7Ghz CPU, and 2GB RAM.

4.4.1 Contexts Utilised

The implementation aim is to use as few contexts as possible to reduce the amount of data to be sent between the mobile device and cloud. Ideally, the Context Processor will use the minimal current context data stored for the user, and historical context, to infer new context. For example, if the mobile device sends the latitude and longitude of the mobile device location, reverse geocoding can take place at CAMCS, rather than at the mobile device. If the user's location is known, CAMCS can determine the appropriate timezone and time of the day. Such data can be used in its work, such as when deciding to intelligently work at a specific time without an explicit request from the user.

As the primary mobile development platform for the CAMCS Client is Android, the contexts are provided by Google Play Services, namely, location,

and user activity recognition. Using Google Play Services is a choice that considers the user experience goal of the research. Google Play Services can adapt to the current power levels/settings of the device to determine what means it uses to collect context, and how accurate it should be (for example, if the battery level is high, it may use the power-hungry GPS to determine location; if the battery is low, it may use cellular tower or Wi-Fi coordination).

4.4.2 Ontology Implementation

The context representation as ontologies in CAMCS is based on the SPICE mobile ontology [142]. This ontology has been used in several projects and is already well developed to model and represent many contexts associated with mobile devices and users.

For this work, a customised ontology was developed, which imports and extends the SPICE ontology, specifically its existing classes and properties. A NamedIndividual is used to represent each user. A NamedIndividual can be thought of an instantiation of a class. An axiom is used to define that these NamedIndividuals are subclasses of the User class in the SPICE ontology. The NamedIndividuals use properties from the SPICE ontology, such as hasLocation, to describe the location of the user. In this ontology, a custom Place class was created, which subclasses Location from the SPICE ontology, to represent a user location. The benefit of taking this subclass approach is that customised properties can be added to the data and the ontology as required, while using the existing ontology entities and axioms. For example, the Place class contains place name, latitude, longitude, and timestamp properties.

The customised ontology used here, extended from the SPICE ontology, is not a complete ontology solution comprising a complete web of semantic data representing a mobile user and his/her context, nor is it intended to be. The semantic challenge here is to represent the relationship that a mobile user has a given context, as an ontology. For this work, all that is required is to define the relationship between the two entities (the user object and the location object), as linked data, through a property. This relationship has to be parseable by the inference engine used by the Context Processor, while the mobile user is disconnected; this capability is provided by the use of OWL, and the OWL API for Java.

4.4.3 Collecting Context with the CAMCS Client

The CAMCS Client running on the mobile device, as previously described, is used to communicate with the CPA of the user. For the work relating to contextual awareness support, it is responsible for collecting the context information using Google Play Services, which must be installed on the mobile device from the Google Play Store. The CAMCS Client currently features two Android services (one each for location and activity), which communicate to Google Play Services to collect the context, and send it to the CPA.

Services start up on the mobile device at boot-time if the user allows, and they register their interest for the context update with Google Play Services. Google Play Services calls back to the CAMCS Client with the context updates, which are then placed into the Context Wrapper. The CAMCS Client uses the Spring Android framework [40], to send a HTTP PUT request to CAMCS. If the CAMCS Client authenticates successfully with CAMCS using the login credentials sent with the HTTP request, the context data is passed to the user's CPA, which will then be stored in the Context Processor.

The user can specify how often the context updates should be sent to the server. These values are used with Google Play Services to determine how often the CAMCS Client should receive the context update events.

4.4.4 Context Profiles

The implementation is based on the use of context profiles. The profiles correspond to the daily activities or status of the user. Profiles are provided by default for home, work, and for the evaluation with cloud services in Section 4.6, a tourist and vehicle profile. The home, work, and vehicle profiles can automatically activate based on the time, day of the week, and activity. Depending on which context profile is activated, the CPA will provide related functionality and services. During weekdays, in the morning, it can switch into work and vehicle profiles, and fetch traffic information for the best route to work. During the weekend, this mode will not activate. See Figure 4.3 which shows how profiles can be set using the CAMCS Client.

The user manually activates the tourist profile using the CAMCS Client, which informs the CPA to switch on this profile. Based on the different cloud services that can be used with the CPA, the developer of a profile can specify tasks and

behaviours that can run while active. Required work that should take place for an active context profile occurs whenever the CPA receives a relevant context update.

Using context profiles can bring advances to CAMCS, and the CPAs of users, in terms of operation. Depending on the active profile(s), the CPA can operate differently, or choose different courses of action in executing a user task. The active profile can determine the choice and selection of cloud services that the CPA chooses from. For example, if the work profile is active, the CPA may only select from enterprise private cloud services that it knows about, when looking for a service to complete a task. If the home profile is active on a weekend morning, when the work profile would normally be active, the CPA would choose not to remind the user of their daily work schedule, or provide the business user with the latest stock market information gathered. If the current context is stale, and new context cannot be gathered, the context history for that user will be used instead to determine the course of action for the CPA for an active context profile. Multiple profiles can be active at the same time. For example, while on a business trip, a mobile user can activate both the work and tourist profile. It should be noted that profiles exist in isolation from each other, defining their own distinct behaviour, with no knowledge of additional active profiles. Conflicting behaviour may occur, so the user should note what behaviour a profile will enable at his/her CPA.

In this implementation, the user cannot override the default behaviour of the context profile, as these are created by developers. Either the user should choose another profile (for example, one that will provide stock market information every day of the week, rather than weekdays only), or create an automatic task for this purpose, described in Chapter 5.

4.4.5 Context Results

The result of a task that utilises context, like any task, is sent back to the mobile device when requested by the user. CAMCS and the CAMCS Client use Google Cloud Messaging (GCM) for this purpose.

When a task is complete, the CPA sends a message to GCM, which pushes a notification to the mobile device, to inform the user that the task is complete. The CAMCS Client then fetches the result of that task from the CPA. The CPA stores the task result as a HTML web page. This page is generated by the

CPA when a task is complete. More details about task results are presented in Chapter 5.

When the user contacts the CPA with a HTTP(S) GET request for the result, the HTML result for the task is sent back, encoded in JSON. The HTML is then displayed to the user in an Android WebView. The implementation of HTML pages to show task results was not as developed at this point in the research compared with that which will be presented in Chapter 5, with few options for result customisation and refinement by the developers.

Consideration was given to implementing a custom Push service to send the result back to the user. Rather than the user having to receive the Push notification, and then request the result, a Push service could simply push the whole result back to the user, when the mobile device was available. This cannot be implemented with GCM, as it has a message payload limit of 4KB. The task result pages often contain a lot of data, going over this limit. Google does not recommend implementing such a custom Push service, and for each developer to use GCM, as opening many sockets for this purpose would have a detrimental impact on battery life, contradicting the integrated user experience aim of this research. Despite the need for the user to explicitly request to fetch the result of a task from the CPA, Google's recommendation was adhered to for the purposes of conserving battery life.

4.5 Design, Implementation, and Use of Contexts

A design objective of CAMCS is the capability for developers to deploy the middleware onto their own cloud infrastructure as a running service, either standalone, or as part of a larger application. Hence, the developer can extend the middleware functionality to cater its operation to the requirements of a specific project, or an enterprise need. In regards to the Context Processor, consideration is now given to two extendibility objectives: first, the ability to add new contexts into the processor, which may be complex and made up of other contexts (such as Location and User Activity), and secondly, the ability for developers to create a custom context profile, which specifies custom behaviour for the CPA when the profile is activated, based on these contexts.

In this section, use of the Context structure within CAMCS, and how it can be used to represent different kinds of context, is presented. It is then shown

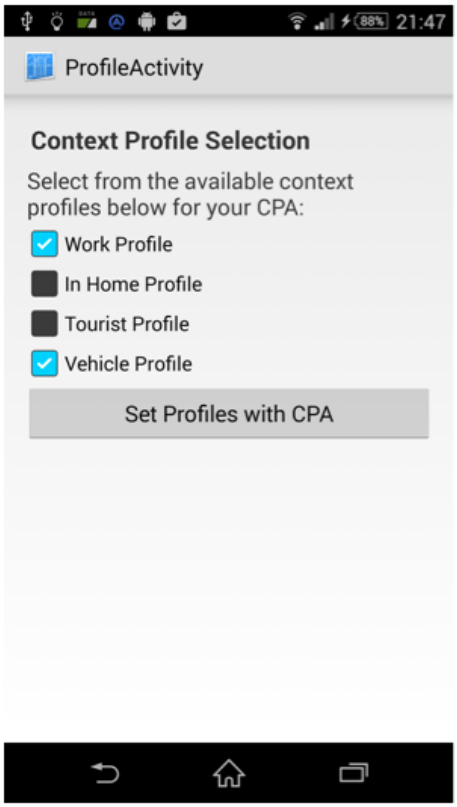


Figure 4.3: Context Profile Selection. The user can manually activate and deactivate various Context Profiles through the CAMCS Client application. The CPA can automatically activate these as well, such as the work and vehicle profiles, based on time and day of week, current activity, and location.

how a developer can achieve these two objects with the developer API for the context processor.

4.5.1 Context Structure

At the top of the hierarchy, one will find the abstract Context class. This features some attributes common to all contexts: a context name, a timestamp (recording when the context was collected), an a Boolean attribute, "isComplex" indicating if the context is a *Complex Context*. A Complex Context is a context made up of several Context Properties. For example, within the Context Processor, the Location and Activity contexts are represented as Complex Contexts, because they are made up of several properties. Location context has latitude, longitude, place name properties. On the other hand, a second type of Context is also defined, *Simple Context*. A Simple Context is one, which only contains a single Context Property. For example, a Simple Context could

simply be "is at home" or "is at work". The corresponding context property in these examples is simply a true/false Boolean indication, and features no other properties. ComplexContexts set the isComplex to true, and SimpleContext set this to false. This helps the Ontology Manager when working with the different kinds of context in determining how each type of Context should be written as OWL (described shortly).

Properties are created from instances of Context Property. Context Property contains two attributes, a name, and a value. A name is simply the name of the property (e.g. latitude in the Location Complex Context example), and value is defined as a type of Object. Depending on what the property is, the value could be anything, such as a Double value for the latitude. One benefit of this is that a Context Property can actually be another Context. For example, a Work Context can have a Location Context as a property. Location Context is of the Complex Context type. A Complex Context contains *a set* of Context Properties, while Simple Context contains *a single* Context Property. See Figure 4.4.

ComplexContext and SimpleContext can be further extended for convenience. For this work, the Complex Context was extended to create helpful User Location and User Activity classes with related methods that operate on the set of Context Properties.

When CAMCS starts up, the Context Manager loads definitions for available contexts (discussed in Subsection 4.5.2 below) into a collection, mapped by context name. The context classes can then be accessed for use through the Context Manager as objects for use in the program by providing the name of the context being sought. The values of the Context Properties can be set, possibly having been received from the CAMCS Client. The contexts can then be written to the ontologies by the Ontology Manager.

4.5.2 Developer Access for New Context Creation

As a result of the extensibility of the context structures provided, developers can create their own contexts for users of the system. A dynamic "context builder" approach taken, allowing a developer to define new contexts and properties.

This process begins with a *Context Definition*, which is class used to define a

new context. A Context Definition provides a name, and a timestamp property. Extending this, there is a *Complex Context Definition*, and a *Simple Context Definition*. A *Context Property Definition* is also provided, which features a name for the property. Complex Context Definition contains a set of Context Property Definitions, and Simple Context Definition contains one ContextProperty-Definition; they are modelled after the Context classes they are used to create. The developer can use these public classes and associated methods to define new Complex Contexts, Simple Contexts, and associated Context Properties. By passing a context definition to the Context Manager, it is inserted into the collection of contexts mapped by name. Any call to the collection to retrieve a context, constructs new Context objects from the definitions, and returns them to the user. In this work, these definition objects were extended for convenience; they were used to create a Location Context Definition and an Activity Context Definition, which both extend the Complex Context Definition. See Figure 4.5.

Allowing developers to create new contexts by defining them this way in addition to being able to extend the complex and simple contexts reduces the overhead associated with the reflection technique which was used in an early version of this work. In the early version, contexts could only be added by extending the Context classes. To map contexts and their properties to OWL, the Java reflection API was used to obtain the names of the properties used for the ontology mapping at runtime. Using a pure object based approach avoids the performance overhead induced by using reflection.

A developer can create new contexts out of the existing Contexts provided by CAMCS. As described, these can be used as a ContextProperty value. To enable this, the Location and Activity contexts are known as *Provided Contexts*. Provided Contexts are defined as enumerated types. By requesting a Provided Context by specifying the appropriate type, the Context, or Context Definition is returned for use by the Context Manager like any other Context or Context Definition. The Context Processor uses the Provided Context type definitions to provide context's for use by the developer, such as the user's location, activity, along with time and date information at the user's location.

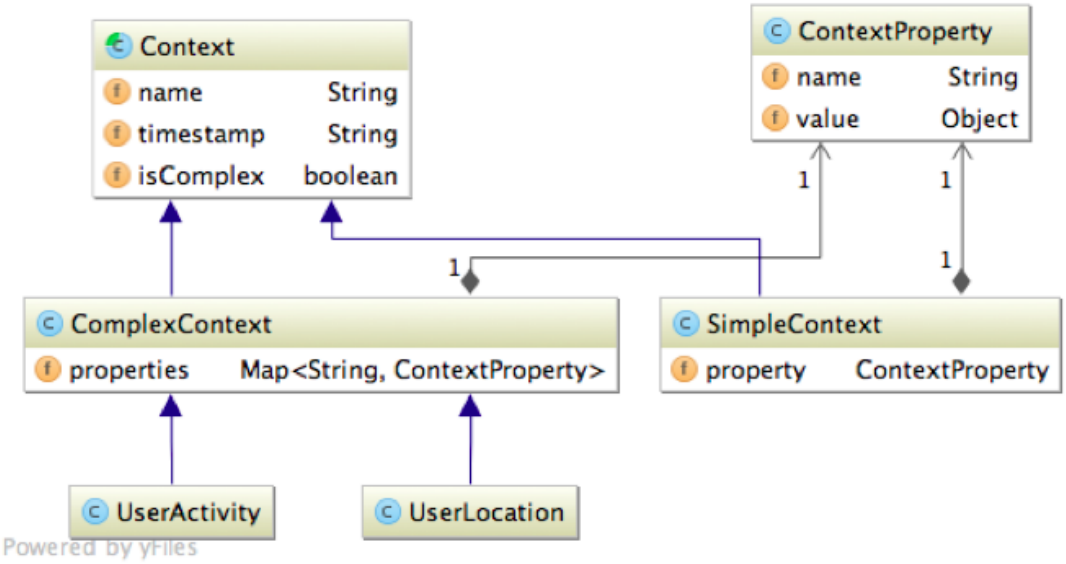


Figure 4.4: Context Structure. From the abstract Context class, there are Complex Context, and Simple Contexts. These contain Context Property objects (a set of them for Complex Context, one for Simple Context). The context classes are used by developers while working with user context data, the values of context properties are read/written from/to the XML ontology files by the Ontology Manager. User Location and User Activity contexts, provided by CAMCS, are extended from Complex Context.

4.5.3 Custom Context Profiles

The developer of a new context can create a new Context Profile to go along with it, and define custom behaviour. This is required to gain flexibility and for adding meaningful features based on context to CAMCS. Custom profiles can be created by extending the abstract Context Profile class. For extending this class, there are three methods that must be implemented by the custom profile, onProfileActivated(), onUpdate(), and onProfileDecativated(). Respectfully, these methods allow the developer to define behaviour that should take place when the profile is first activated on the CAMCS Client, whenever a context update is received from the CAMCS Client, and when the user deactivates the profile on the CAMCS Client. Custom profiles are passed to the Profile Manager of CAMCS, to be inserted into a named collection of Context Profiles, similar to how the named collection of available contexts are found in the Context Manager as Context Definitions. The Profile Manager, at system start-up, loads the default context profiles available in CAMCS into this collection automatically, such as the profiles used for the evaluation work in this Chapter.

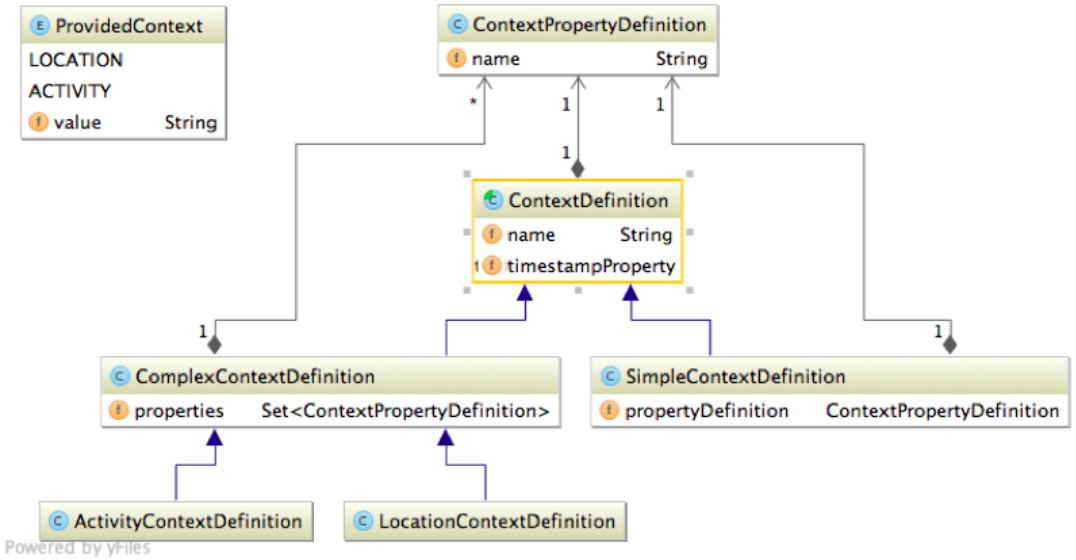


Figure 4.5: There are Definitions for Complex and Simple Contexts, which extend Context Definition. Context Property Definition is used to define Context Properties. Modelled like the Context classes in Figure 4.4, Complex Context Definition contains a Set of Context Property Definitions, and Simple Context Definition only has one Context Property Definition. A timestamp Context Property Definition is provided to both types of Context Definition. The system also features built-in Provided Contexts, of type Location, and Activity; each has their own Definitions, extended from Complex Context Definition.

Storing context is meaningless without doing something with it, either by using it with services as described as part of the evaluation Section 4.6, or defining behaviour in a profile. For example, if a developer creates an "At Work" profile, which could be based on a Simple Context (either true or false), then it must work with Provided Context for location and time. The ontology does not store the valid location or time/date range for when the user can be considered at work, it just stores the true/false Context Property value. The developer, in the custom profile, must check the user's context using the Context Manager to see what is the user's current location, and the time/date at that location, and use custom logic based on the results to determine should the At Work context Property be true or false, and then set it. Of course, in this example, the developer would have to infer that a given location was visited every day during the same time range during the week, and assume this is the workplace location, in order to set the value to true.

4.5.4 Ontology Mapping

The final implementation aspect of the Contexts is how they can be mapped to the ontology files as OWL. This varies for Complex and Simple contexts.

A Complex Context is inserted into the user current context ontology file as its own NamedIndividual. Each element of the Context Property set is created as a property of this individual, named in the format of "has{PropertyName}". Then, the NamedIndividual representing the user, is updated by adding an object-property named in the format of "has{ContextName}", which refers to the NamedIndividual of the specific context by it's URI identifier, as an RDF:Resource attribute.

A Simple Context, as a context that only has one Context Property, does not have a NamedIndividual created for it. Instead, it is added as a data-property of the user's NamedIndividual, with a name equal to that of the Context itself, and the value of the Context Property is given as the value of the data-property (i.e. the name of the Context Property itself is not used in the Ontology mapping).

A sample of location context representation as OWL in the current context ontology file of a user can be found in Listing B.1 in Appendix B. Note that some pre-amble has been removed for space. This is an example of linked data; the user entity is linked to the location entity by a hasLocation property on the user entity, by the RDF URI of the location entity.

4.5.5 Querying Context

The use of a Context Request has already been explained; this is sent to the Context Manager to obtain a user context from the ontology, via the Ontology Manager, which reads the context XML files containing OWL. The Context Request can be parameterised to customise how context is returned.

To obtain any context, the name is provided with the Context Request. For a Provided Context, this is obtained from the Provided Context type. The developer must know the names of their own context for querying. The ID of the CPA is used to access the correct XML Ontology files for the user, so one CPA cannot request the context of another user (if attempted, the Context Manager would deny access because the ID of the requesting CPA would not match the

ID of the user context requested). When a request is received by the Ontology Manager, the OWLReasoner is used to query and return the contexts. A Context Request object provides a method for the Ontology Manager to obtain an instance of the correct Complex/SimpleContext object stored in the context definition collection from the Ontology Manager, by using the name of the requested context (recall, that each context definition is mapped in this collection by name). The Ontology Manager will pass the retrieved context values from the ontology file to the Complex/SimpleContext, which will set the values of the properties to the actual values.

The use of historical context can also be requested. The Context Request can provide a Context Validity type, which contains ranges, such as "CURRENT" (i.e. do not use historical context), "RECENT" (use current or all historical context), "LAST WEEK" (use current or context from the last seven days), or "MONDAY" (use current or context from the previous Monday), to name a few. All historical contexts stored in the historical context XML file for the user are stored as NamedIndividuals, with random URI identifiers. If the request has a valid Context Validity set that can be used for historical context, it will search the historical context XML file for NamedIndividuals which have the same type as the requested context, and timestamp within the Context Validity period. These are returned as lists of Context objects.

It is worthwhile to note here, that only Complex Contexts are moved into and stored in the historical context. Simple Contexts (such as in the aforementioned At Work example) are not. The reason this approach was adopted is because Simple Contexts only have one Context Property, and offer little useful historical information as a result, especially as Simple Contexts tend to have their single Context Property set based on the values of Context Properties of one or several Complex Contexts, which are stored historically (for example, At Work, a type of Simple Context, has it's single Context Property, a Boolean true/false indication, set based on the values of of User Location, a type of Complex Context). As such, any given historical instance of a Simple Context can be inferred from looking at the historically stored Complex Contexts; these will imply the Context Property value of the current Simple Context.

4.6 Evaluation

Attention is now turned to use-cases for evaluating the operation of the Context Processor. The aim and novelty of this work, is how context collected by the CPA, can be used with mobile cloud services. For this purpose, two experimental mobile cloud services have been implemented, with related context profiles. The first is a simple tourism-based location information service, which is similar to other contextual service work in the literature. This work is based around the Foursquare API. The second is a traffic information service that uses the Twitter API to find traffic information for the user while they are in a car. Also shown is how the Context Definition and Context Property Definition classes are used to create the "In Home" context and related context profile. The benefits and limitations of the work are also discussed.

4.6.1 Tourism Mobile Cloud Service

4.6.1.1 Location Services with Foursquare

Foursquare provides an API for developers to gather information about locations, using either its Venue or Explore API. A developer can send a HTTP request to a URL endpoint, providing attributes, such as either a place name, or latitude and longitude coordinates. The API will then return information about this location. It comes in the form of a list, marked up in JSON, which can be parsed to extract information about different venues and attractions near that location. For each venue, it provides a name, description, location, and contact information, along with reviews provided by Foursquare users. To evaluate this work, this API was used to gather information about a location context that has been stored by the CPA of a user, which can operate either by sending the coordinates received from the mobile device, or the reverse geocoded name of a place, given by the coordinates. This will show how location context data collected by a user's CPA can be used to personalise the service result from Foursquare to the user's location.

4.6.1.2 Tourism Service Implementation

To implement this tourism service, a cloud-based service that can be used by a CPA was developed. For the purposes of this evaluation, to forego factors such

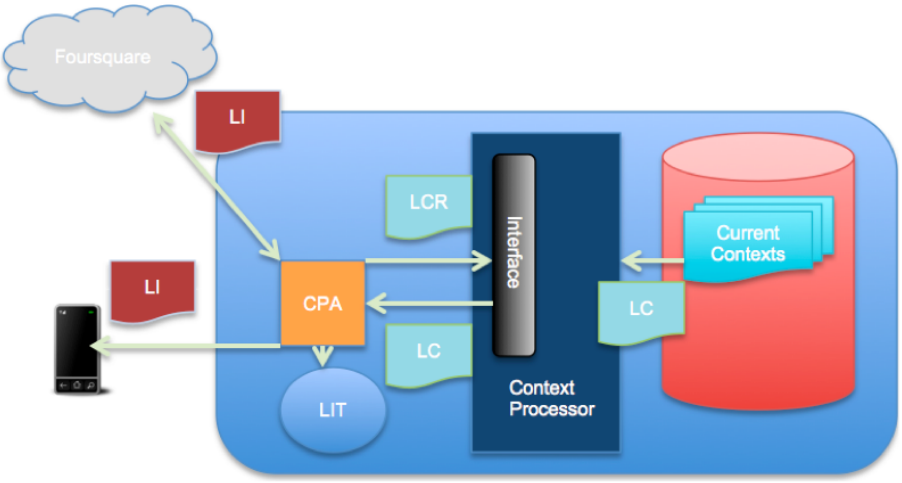


Figure 4.6: Running a Location Information Task. With the Tourist Profile active, the CPA is running a Location Information Task (LIT). It sends a Location Context Request (LCR) to the Context Processor, which reads the users Location Context (LC) from their Current Context Ontology. The CPA contacts Foursquare with this information, which responds with Location Information (LI). This is returned to the device.

as service discovery, this service project is simply a dependency of CAMCS, so that CPAs already know of the existence of the service.

Whenever the tourist context profile is active at the CPA, the CPA assumes that the user is on some holiday or trip, and the user is interested in receiving information regarding places of interest around the user as they move around. Additionally, when a location update is received by the CPA from the mobile device, it will check the context history to determine if the location has been visited recently (in the past week). If not, the tourist service contacts Foursquare with the coordinates to gather information about the area. When the service receives the response, it parses the JSON and extracts the location information. This information is then used to generate a HTML result page, containing a list of the venue information returned from Foursquare. The service then notifies the CPA of the user that the location information gathering task has completed, and a notification is sent to the mobile device using GCM.

When the user opens the notification, the HTML web page result is converted to JSON, and sent back to the user's mobile device for viewing in an Android WebView - see Figure 4.6, which shows the series of steps involved. Figure 4.7 presents a screenshot from the mobile device, displaying the location information result sent back by the CPA. Again, this is a simpler implementation of the complete task result HTML features presented in Chapter 5.

4.6.2 Traffic Mobile Cloud Service

4.6.2.1 Traffic Services with Twitter

Many services such as Google Now provide traffic information, such as journey time estimates on Android. Google Maps provides graphical maps depicting traffic build-ups along various roads. For this evaluation, a different approach was taken. Several agencies, such as the AA in the UK and Ireland, provide traffic information through their Twitter accounts. The Twitter accounts of local radio stations in cities also often publish Tweets containing traffic updates based on reports from station listeners in their cars. Also, users who encounter traffic-hold ups, queues, and accidents, often Tweet this information themselves while stopped, that a radio station or agency may not have information on at the specific time.

To demonstrate the Context Processor, the CPA will use user activity and location context to gather Tweets containing traffic information for the user. In many countries, it is illegal for a driver to use their mobile device while driving. For this purpose, once traffic information has been pushed to the mobile device from the CPA, it is automatically read out, using the built-in Android Text-to-Speech engine.

4.6.2.2 Traffic Services Implementation

To implement this service, an "InCar" context profile was created. Whenever the CPA of the user switches to this profile for the first time, the CPA will use the traffic service to fetch the traffic information for the area where the user is located. The switch into this profile is taken from the activity context of the user; one of the user activities provided by Google Play Services is "in_vehicle". When the CPA receives this context update from the CAMCS Client, the current context is checked; if the previous activity was different, and the previous location was home (e.g. not a train station for example, suggesting the vehicle is a train), the CPA then calls the traffic service.

As will be presented in Chapter 7, the CPA integrates with external service providers such as Facebook and Dropbox. Twitter is one of these providers. The CAMCS Client authenticates with the Twitter account of the user, and the authentication details, from the OAuth protocol, are sent to the CPA and stored there. For this evaluation using the Twitter traffic service, the CPA will



Figure 4.7: Foursquare Result. Screenshot of Foursquare location results from the CPA, displayed on the CAMCS Client.

use these stored authentication details, to query the Twitter search API. This is simply a query with search keywords for the user location given from the Context Processor, along with the "traffic" keyword. The Tweets can be filtered by recent or popularity, and from the Tweet location also being in the area of interest. The Tweets are returned to the CPA, which stores them with HTML markup for the result to be sent to the mobile device.

4.6.3 Traffic Mobile Cloud Service

Google GCM is used once again to send a notification to the user, to inform him/her that the traffic details are available. However, a flag is also sent with this notification, indicating that this result should be read out using the Google Text-To-Speech engine. Upon receiving the GCM message with this flag set, the CAMCS Client will fetch the traffic information result from the CPA, without the user having to tap and open the notification. When the traffic result is returned, the HTML tags are stripped out of the result, and the Android Text-To-Speech engine reads out the Tweets - see Figure 4.8. Note that use of Android Text-To-Speech requires a speech engine on the device. The Samsung Galaxy S3 device used in this evaluation has such an engine.

Such an approach is open to abuse, as some Tweets may not be directly related to current traffic conditions, or there may be un-trustworthy users tweeting invalid information. Many Tweets returned contained advertisements, and profanities uttered by frustrated drivers. The functionality can be extended to allow users to select trusted sources on Twitter for such information, possibly by specifying filters as inputs to the Twitter API call through the user's CPA. Tweets also often contained other useful transport information not related to drivers, such as information about trains, subways, and buses.

4.6.4 Custom Context Creation

In Listing B.1, it can be seen that the user's NamedIndividual has a Simple Context attribute, "InHome", set to false. This Simple Context was not built into CAMCS for this evaluation; it was implemented using the Definition classes for developers. Behind the scenes, both the User Location and User Activity contexts are built on with the Complex Context Definitions and Context Property Definition classes as well. These are constructed at start-up time within the Context Manager. To develop the "InHome" context, a separate project package and classes were created, rather than being built into the Context Manager. This package is scanned at start-up time, and has an initialisation method that passes the Simple ContextD efinition for the "inHome" context to the Context Manager, for insertion into the mapped collection of Context Defintions. The "InHome" context, being a SimpleContext, has one ContextProperty, called "Location", which is the Boolean true/false value seen in Listing B.1.

To go with this context, a custom Context Profile was created which is also added to the Profile Manager at start-up time, the "Home" profile (the in-built Home was removed profile for evaluation). The Home profile knows about the existence of the "InHome" context, and accesses it through the Context Manager. To determine the user's home location, it looks at the historical location context for the previous week, between 12AM and 6AM, and uses the coordinates found for the majority of the location results returned (which should be the same if the user is at home in bed at these times, with the mobile device left in the same place), and assumes this to be the home location coordinates. When a context location update is received from the CAMCS Client, the onUpdate() method of the profile is called, and it carries out this check



Figure 4.8: Twitter Traffic Task Result. Screenshot of traffic information provided by Tweets for Cork City, Ireland, displayed on the CAMCS Client.

and compares with the coordinates received in the update, and sets the ContextProperty value accordingly.

4.6.5 Discussion

The complexity and potential is in the ability to create services that the CPA can use with the gathered context. These services can be developed and deployed on clouds to work with existing web-based services exposed by service providers to complete work, or provide information, to the user, through their CPA. Such actions can occur as a result of a user requested task, or, as in the case of the tourist service, the CPA can undertake this work automatically without user intervention.

The use of web and cloud-based services is difficult, but there is great potential here when compared with other mobile cloud approaches. Existing services do not accept visits from the CPA, nor do they readily support the functionality. Therefore, wrappers need to built around these existing services, for the purposes of contacting and utilising them. This is exactly what was done for the services created for the evaluation. They serve as wrappers for the developer to specify how the CPA should work with an existing cloud-based service. In

the past, this had to be carried out for any existing service. While it may seem like a disadvantage, it does allow the developer to implement custom functionality using the existing services, in the form of a mash-up. However, a more generic approach to working with mobile cloud services, without using wrappers, is the subject of Chapter 5.

The time commitment for developers required to create custom contexts and profiles is worthwhile, as this will deliver a high level of personalisation, benefitting the user experience aim of CAMCS. The development goal of providing the Context Definition and Context Property Definitions as demonstrated is to make this process as quick as possible, while developers still have the power to extend these classes as well for completely custom implementations. To gain relevant and useful functionality, the development of custom context profiles to go with custom contexts should be considered essential.

4.7 Conclusions

In this chapter, considerations for and the implementation of the Context Processor component of CAMCS was presented. This component will personalise the use of mobile cloud services with the context of the user, bridging the gap between them. As a result, service execution can vary with the user's situation. When a CPA wishes to complete a task for the user, it can query the Context Processor to get the context of the user, which may personalise the task execution to his/her situation. This is aided by the use of Context Profiles, which can influence how the CPA operates with cloud services based on an active context profile and situation. It also allows the CPA to undertake work for the user without his/her intervention, based on the Context History, which is stored with the CPA.

Several requirements of the Context Processor were analysed to determine what kind of framework was required for context representation, under the areas of context representation, inference, and storage. An ontology-based approach was chosen because of its storage capability, and the ability to use a reasoner for context inference. This is useful when the mobile device is disconnected from the cloud and new context cannot be gathered. The architecture of the Context Processor, along with the structure of Context was presented, with Complex and Simple Contexts, as well as how current and historical context is

used within CAMCS. Mobile context is collected by the CAMCS Client, based on Google Play Services, and is sent to the CPA to be stored by the Context Processor. Experimental evaluation was also presented showing how collected context can be used by the CPA to work with mobile cloud services; two experimental services were presented, a Tourism Places of Interest service (based on providing Location context to Foursquare), and a Twitter-based Traffic Information Service (using location and activity context data).

In the next chapter, a generic solution for CAMCS to discover and consume mobile cloud services for completing user tasks with CPAs is presented. A unique services description format is also presented, which features support for describing context data that can be used by a service during its execution. The chapter also shows how context data can be used with service discovery.

This chapter was based on [99], with the following exceptions: Section 4.2, Section 4.5 which was heavily modified and extended, and Subsection 4.6.4. Various modifications have been made to the text throughout the chapter, to fit with the flow of the thesis.

Chapter 5

Mobile Cloud Services - Description, Discovery and Consumption

5.1 Introduction

The CAMCS middleware operates by making use of web services already deployed in the cloud. Such services form the basis of service-oriented architecture (SOA), and can be used for delivering useful and relevant functionality and information to the mobile user. Service providers expose public APIs using SOA patterns and technologies, which allow developer access to the data and functionality offered as part of their platforms. This is in addition to native applications provided by the same service providers, for the various mobile platforms. The CAMCS approach of CPAs using these APIs to complete tasks for users, can therefore compliment the existing applications available to mobile devices; service providers do not exclude the possibility of using both approaches to access their platforms.

As with other MCC application models, taking an SOA approach presents problems of its own. Services deployed in the cloud conform to web service standards, such as the Simple Object Access Protocol (SOAP) [130], and Representational State Transfer (REST) [36]. How to describe, discover, and consume these services from a mobile device, presents several challenges. In terms of description, services are often described using the Web Services Description Language (WSDL), which is XML-based, and cannot be understood by non-

specialist users, and therefore, on its own, is useless for describing a service to an end user. As a result of these XML-based descriptions, the user has traditionally been unable to take part in the discovery process. However, it is widely agreed that XML was never supposed to be directly presented to end users, and is only for use by software and developers; hence, a new approach is required. RESTful services often do not have associated descriptions at all, aside from API documentation. In the area of service discovery, existing research has shown that automatic discovery of appropriate services is simply not mature enough for widespread use, and is therefore still a very manual, developer-oriented process. For service consumption, solutions must be able to work with various different kinds of services and web service technologies. Clients can be developed to access SOAP and RESTful services. To-date, standards do not exist for comparing similar services, nor for invoking services that may take similar input parameters, and output similar content.

CAMCS addresses these problems by implementation of a user-oriented service discovery process, which allows discovery of existing web services from a custom service registry solution. The primary contribution of this registry is the means by which stored service description records are structured; the design goal being a user-oriented approach. The mobile user can utilise this solution provided by CAMCS with his/her own CPA; web services discovered by CAMCS are then used to complete tasks that they have offloaded from the thin client running on the mobile device.

In this chapter, the MCC registry model is introduced, along with the service description structure used to store service information within. Also presented is a walk-through of how a CPA uses this solution to discover services by querying the registry; a user can then choose from among these whichever service they believe suitable for the task. How the data required for consuming the service is collected from the user is demonstrated, as well as how a service is consumed for the purposes of completing a task. Additionally, this chapter presents the model used by the CPA, for representing user tasks, and how these tasks run with discovered services. Enabled features are also presented, such as automatic task execution, which allows the CPA to execute tasks without any request from the user, enabling disconnected operation.

5.2 Mobile Cloud Service Registry for User Oriented Service Descriptions

Many services providing various functionalities already exist in the cloud. These conform to existing web service access technologies, and service registries exist containing descriptions of these services; these are not designed for human interaction. By means of the CPA, mobile users can take advantage of these services, references to which are stored in a mobile cloud service registry, with a user-oriented mobile cloud service description format, also created for this work. This registry was also created uniquely for this work. The user-friendly descriptions will allow an average user with little or no technical experience to easily find and use cloud-based services. The registry implementation is presented first, followed by the service description structure used to store service records.

5.2.1 The Service Registry

The service registry implementation is a JavaEE based application that is deployable to any application container running on a cloud-based server. The registry provides an API for querying services by search terms, which then returns a list of matched services. The registry is built on top of a NoSQL database, MongoDB. A NoSQL database was chosen because it is a document based data-store. Therefore, each service record is represented by a document in the database. This is easier to work with rather than using several various tables in a relational database such as MySQL for the required entities, which then need to cross-reference each other for relationship mapping.

The search operation takes a string-based query provided by the user. The Apache OpenNLP library [79] is used to tokenise the query, and this is matched against a set of descriptive terms that are stored as part of each service record within the registry (discussed further in Subsection 5.2.2). The results of the query, which will contain a list of matching services found in the registry, are returned to the caller in JSON format. Taking this approach provided greater flexibility with querying capabilities when compared with querying a UDDI registry; the limited scope for querying with a UDDI registry, which allows only exact or approximate match queries on service names, was one of the issues that necessitated a new registry for this thesis work. With UDDI, the mo-

bile user would be required to know the exact name of the service in advance of searching for it.

Service providers or developers need to be able to add services into the registry in order for users to find them. An API endpoints allows developers to add their services to the registry by means of a HTTP PUT request; if the service is described according to the description format used in this work, the service is stored in the registry, the service will be added. The service description will again be marked-up in JSON and sent in the body of the HTTP request. Future possibilities that can be considered include automatic converters to extract the required information from existing WSDL files in order for them to be automatically added to the registry. Another possibility includes a web-based interface where developers can describe their service without having to provide any existing description file for the service. This would be far more appropriate for RESTful based services, which typically do not use service description files. Both of these approaches will allow existing web services to be added to the registry with little additional effort; their descriptions would not need to be manually converted to the service description format.

5.2.2 Service Description Structure

The structure of the user-oriented mobile cloud service descriptions used within this registry gives flexibility to this solution more so than the registry itself - see Figure 5.1. The aim of the descriptions in this registry is that they will allow simple user interaction to discover and utilise a service, with no technical details or mark-up of any kind presented to the user. The registry will store references to two types of service, SOAP and RESTful. For this work, only RESTful services are considered for evaluation. Ultimately, the high level descriptions will be the same for both.

- *Service*

  A Service is simply a high level record/abstraction for a service offered by a provider. A Service features a *name*, a *description*, and a *provider*. These are in place for human consumption in the service discovery process. The Service abstraction also features a *type*, which can either be "soap" or "rest", indicating which web service technology is used to implement the service. As briefly mentioned in the Subsection 5.2.1, a Service also contains a collection of descriptive *terms* that can be specified

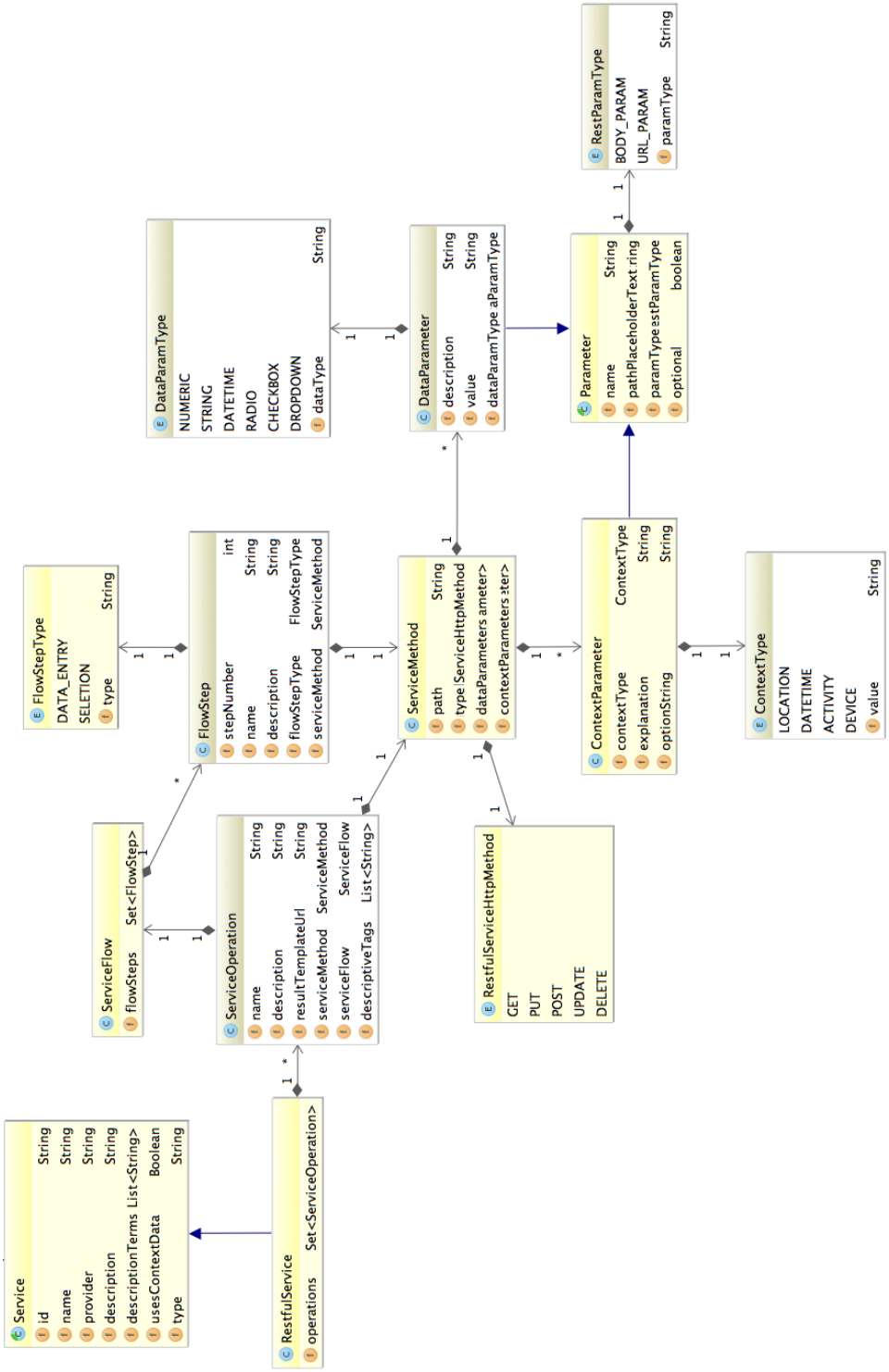


Figure 5.1: Service Structure UML Diagram. The diagram should be read in a left-to right direction, starting with Service and Restful Service. Services are made up of Service Operations; these contain Service Methods, which in turn contain Parameters. Each entity contains user-friendly names and descriptions for use during the discovery and service selection process by the user. In this implementation, only RESTful services are considered, and specific entities for these services can be seen.

by the developer. The name and description are also queried for matching terms in the query, along with the collection of descriptive terms. A Service contains one or many *Service Operations*.

- *Service Operation*

  A Service Operation is a high-level view of one specific operation provided by a Service. If one were to compare with WSDL, there is one Service Operation for each operation in the WSDL file. Like the Service record, it also features a *name* and *description*. These are also high-level descriptions for human consumption in the discovery process, and describe what the operation does. A *Result Template* URL can also be found. This is a URL to a HTML template webpage, which the service developer specifies. This template will be used to display the service results, and can be customised to a company's own branding. Results are described in more detail in Subsection 5.4.3.

  In terms of the functionality offered by the Service Operation, this can take two forms. The Operation either consists of work that can be completed with one call, or work that can only be completed with a varying number of steps, known as a *Service Flow*.

  In both cases, the functionality of the service is encapsulated by a *Service Method*. An Operation that requires only one step to complete, contains one Service Method. An Operation that requires several steps to complete, contains a Service Flow.

- *Service Method*

  A Service Method corresponds to the function that can be called to consume the service. A Service Method also features a *name* and *description*, in this case describing the individual method. They also feature a *path*, which is the URL endpoint for calling that Service Method as a function/operation, over HTTP. Where the URL takes parameters, these are represented in the path string with "$" placeholders, such as /users/get/$userid. Service Methods also feature a *type*, which, for RESTful services in this implementation, represents which HTTP verb the Service Method will respond to (a GET/PUT/POST/DELETE request). Finally, a Service Method features two different collections of *Parameters*, *Data Parameters*, and *Context Parameters*.

- *Parameters*

  As the name would imply, a Parameter is an input data parameter to the service. Once again for human consumption in the discovery process, a Parameter features a *name* and *description* (how these are used is shown in Section 5.3). A default value can optionally be specified. A Parameter also contains a *type*. For a RESTful service, a parameter type indicates if it is a URL parameter to be passed to the service in the URL, or a body parameter, to be passed marked-up in the HTTP request body. If it is a URL type parameter, the name will match up to the respective placeholder used in the path attribute of the Service Method, and will replace it with the actual parameter data value at call time.

  For complex parameter data types, a Parameter can include a sub-collection of Parameters, which make up the complex type. These are not presented to the user in any different way in the discovery process compared to simple data type based Parameters; the user will not see anything which resembles the underlying data types.

  The two types of parameters are now explained briefly.

  Data Parameter - A Data Parameter is an input parameter to the method, where the parameter itself is of a traditional datatype, such as a String or numeric type. They feature a *Data Param Type* attribute to indicate which data type they are. Mobile users need not be concerned with required data types; the CAMCS Client uses this for collecting valid data from the user. It supports Strings, numeric types, and datetime types. Support is also provided for radio button, checkbox, and and drop down choice selections (of Strings and numeric types). This is described in more detail in Section 5.3. Listing C.1 in Appendix C shows an example of a user-oriented service description for a Google Calendar service (seen in figures throughout this chapter) which takes data parameter inputs.

  Context Parameter - A Context Parameter is an input parameter to the service, where the parameter itself is some data representing user context. They feature a *Context Type*, which corresponds to the Provided Contexts in Chapter 4, as well as an *explanation*. The explanation is for the developer to provide information to the mobile user on how the context data will be used. Finally, an *option* String is also

present, which allows a formatting pattern to be specified, indicating how the data should be provided to the service (for example, for a DateTime context, in what order should days, months, and years, be provided). Context Parameters, as will be shown in Section 5.3, are not provided by the user on the mobile device when entering data parameters; the mobile user simply consents to the use of the context data. The actual data is provided by the Context Processor of CAMCS when task execution is about to occur with a discovered service. Listing C.2 in Appendix C shows an example of a user-oriented service description for the Foursquare Tourism/Places of Interest service seen in Chapter 4, which takes context parameter inputs.

Both types of parameter can be required, or optional. If the user refuses to provide required data parameters, or does not provide consent for required context data to be used, then the service cannot execute the task for the user.

Some trouble was encountered during implementation, from the requirement of several web service APIs to have a developer key sent with the request. Every end user with a CPA will not have a developer key. To overcome this, the CAMCS developer key was inserted into the registry as a Parameter of each Service Method that required a key (unseen by the end user). This is not ideal; more open web and cloud service APIs will be required for a mobile cloud computing approach based on SOA.

- *Service Flows and Flow Steps*

  For a Service Operation that requires more than one step to complete, a Service Flow is used. These correspond to web flows in the area of web development. For example, a flight booking service; this contains several steps, from selecting departure and arrival airports, selecting a flight based on time, the number of passengers, passenger names and details, payment details, etc. A Service Flow, contains a set of *Flow Steps*. There is one Flow Step for each step in the flow. Each flow step contains a name and description as with the other entities in this service description structure. They also contain a step number, indicating ordering.

  Each Flow Step also contains a Service Method. Each Flow Step can be thought of a wrapper of a Service Method, as each step in the flow is

essentially a Service Method that is called to execute that step.

Consider the selection of departure and arrival airports as step 1, and flight selection, as step 2. The important factor here, is that the options presented to the user in step 2, will vary based on what information was provided in step 1; there is no static, pre-determined choice of options that can be set ahead in advance. For this reason, a Flow Step contains a *Flow Step Type*, which distinguishes between two types of parameter entry for mobile users, when the Service Operation is a flow; a *Data Entry Step*, and a *Selection Step*. How these are presented to the mobile user on the CAMCS Client will be shown in Section 5.3, but briefly:

Data Entry Step - A Data Entry Step is a step in the flow, where the user provides parameter input for the specific step; either data parameters or context parameters. In the case of the flight booking example, this would be a step where the user enters the names of the departure and arrival airports, as String-typed data parameters (this is Step 1 in the previous example).

Selection Step - A Selection Step is a step in the flow, where the user must make a selection/choice from options presented. These options are computed by the service, based on the input from a previous Data Entry Step. In the case of the flight booking example, this would be a step where the user picks which flight they would like to take between the two airports chosen in the previous step (this is Step 2 in the previous example).

Once the user has provided either Data Entry parameters, or made a selection, depending on the type of Flow Step, the information is provided by the CPA back to the service. The CPA then moves onto the next Step in the flow (the server can also keep a record of this, which is discussed later).

Listing C.3 in Appendix C shows an example of a user-oriented service description for a Lufthansa Flight Information Service (seen in Figures 5.7 and 5.8) which uses a Service Flow with 3 Flow Steps; Steps 1 and 3 are data entry steps, and step 2 is a selection step.

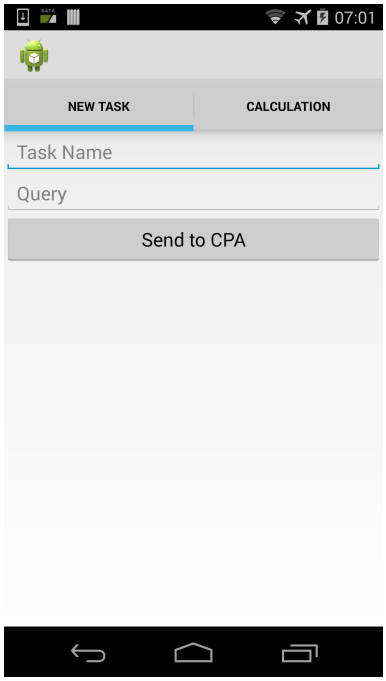


Figure 5.2: New Task Creation on the CAMCS Client. The user provides a name for the task, along with a description. The description is used as a query by the CPA, when performing service discovery.

5.2.3 Querying

As will be shown later, when a mobile user creates a task on the CAMCS Client, they provide a name, and a description, see Figure 5.2. The description provided is used as a user-query by the CPA, when it queries the service registry through CAMCS, when trying to discovery a service to complete the task. In this thesis work, two optimisations are used - part-of-speech tagging, and context aware discovery, by making use of context data from Facebook (optional). These are now discussed in more detail.

5.2.3.1 Part-of-Speech Tagging of User Queries

Part-of-Speech (POS) tagging is the process by which the individual words in a String of several words, are tagged, indicating what kind of word it is, such as a noun, a verb, or an adjective.

The aim of this is to remove words from a user query (i.e. a task description) that will not be useful in the discovery process, while querying the registry. For example, if the user were to type "I am looking for a service that provides flight bookings", then, in terms of the Service descriptions and descriptive tags

previously discussed in Section 5.2.2, tokenised Strings from phrases such as "I am looking for a service that provides..." will add to the search time, and not yield any greater choice of services. "flight bookings" is the only useful part of this query, and ideally, only these tokens should be used to query the registry.

Before the registry is queried, POS tagging is performed to try and remove unhelpful tokens from the query String. Apache OpenNLP contains a POS tagging facility, which is used with a dictionary, to tag each token in the query.

Any tokens that the tagger identifies with the following tags are kept as part of the query:

- NN (Noun, singular or mass)
- NNS (Noun, plural)
- NNP (Proper noun, singular)
- NNPS (Proper noun, plural)
- VB (Verb, base form)
- VBD (Verb, past tense)
- VBG (Verb, gerund or present participle)
- JJ (Adjective)

Tokens matching other tags are discarded.

When queried for services, the registry can either return complete Service records as described, or reduced versions of the Service records, which just contain the names and descriptions for the Service Methods and corresponding Parameters.

5.2.3.2 Semantic Similarity Calculation for Context-Aware Discovery

The aim of context-aware service discovery is to rank the list of discovered services, such that those that are most relevant to the user, appear first in the list. Two methods can be considered for achieving this ranking; the service registry could perform the operation at query time before the results are returned to the user's CPA, or the CPA could perform the operation on the unranked list returned from the registry. In this work, the latter approach has been taken. If it were the case that the registry was to perform the ranking, then user context

data would have to be sent to the registry. For user privacy, the CPA will use the context information stored in the Context Processor, to perform the ranking, before the user is notified that discovery for a task has been completed.

Further to the use of Location and Activity context explored in Chapter 4, in order to use more sources of context for ranking, user data from Facebook was utilised. Using user-provided data on social networking profiles was highlighted by Beach et al [10]. In Chapter 7, work is presented on another application model of CAMCS, which can integrate with the mobile user's file storage and social accounts. This integration was expanded, in order for context data to be pulled from the mobile user's Facebook account, to be used for the context-based service ranking. For this, the following permissions on Facebook are used to collect relevant data:

- user_work_history
- user_education_history
- user_likes

The names of the permissions used should be self-explanatory. The education (such as college course taken) and work details (company name and type) of a user are used to determine what kind of services the mobile user would be interested in using to complete tasks sent to his/her CPA, as well as the information on pages that he/she has "liked". "Like" information can be used to create a picture of what hobbies he/she may have. For example, if the Facebook page of airlines such as Aer Lingus or British Airways, or pages such as BBC Travel or Lonely Planet, are "liked", then one could assume that the mobile user is interested in travel. This is enabled because Facebook pages such as these examples, feature a *category*. One such category is travel/leisure, for these pages. Each of these categories of pages is stored as an *Interest* context.

For a CPA to rank services for the user, this work has adopted and modified a technique from the DaaS work by Elgazzar et al [30], based on *semantic similarity*, using the Normalised Google Distance (NGD) measure. NGD is used to calculate the semantic similarity measure between two words. This works based on Google's own search corpus, based on the number of times the two words appear together on the same page, against the total number of hits for each word separately. Words that have high semantic similarity most often occur together on the same page, and the formula returns a number close to 0. Words that are not related are considered never to appear together, and the

formula returns a value close to 1 in this case.

The formula used in DaaS makes use of properties; user preferences, device profile, environment context, and user rankings. Weightings are assigned to these properties based on their level of importance. The properties of the mobile device/user, are compared with the equivalent property of the web service (for example, mobile device screen resolution against the resolution required by the service), using the NGD, and the weight is applied to this result. A summation of the product of the weighting, and the NGD for each property of each service, is used to determine the ranking. In the approach taken in this thesis work, the solution, as shown later, is platform independent, and therefore does not take into account properties such as device profile or environment context, and user ranking are not used. As a result, no weights are applied.

To perform the ranking, a summation is used of the NGD between each of the user's interest contexts, and the descriptive tags of the services and the service operations retrieved from the registry. The list is then ordered according to this ranking. At this time, the user's education and work do not feature categories, and so are not used in the calculation. Further work on this is possible, such as including page descriptions from Facebook (paragraphs of text explaining what the page is for), and utilising the POS tagging technique to remove useless words from being part of the ranking.

Ranking user context data against the discovered services only takes place if the user has linked his/her CPA with their Facebook account, and if the user has turned on a setting in CAMCS Client for ranking to take place. If the user does not provide this permission, or does not link his/her CPA with their Facebook account, ranking does not take place. The mobile user can also stop this integration at any time, and any stored context information from Facebook is deleted from the user's context ontology. Future work can include a mechanism for a user to provide feedback on how effective the service ranking operation is, when compared with no ranking. The mobile user should be able to indicate how relevant or useful the services discovered are, enforcing the aim that the most relevant services appear at the start of the list.

5.3 Service Discovery and Consumption with the CPA

The CPA will query the registry, with the task description as the query string. Interactions can be seen in Figure 5.3.

5.3.1 Discovery

With a user-provided description of a task to complete, consisting of a task name and description, a CPA will query the registry with this description (or "query-string"). The registry will return the full Service descriptions for matching services, including the Service Operations offered by a service, and the Parameters for each of the Service Methods wrapped by the Service Operation. The initial query of the registry by the CPA, given the query-string provided by the end-user, is the beginning of the discovery process.

With this list of discovered services from the registry, the CPA notifies the mobile user that services have been discovered for a given task - see Figure 5.4. If context-aware discovery is enabled, ranking is performed by the CPA at this point before the list of discovered services is sent to the mobile. The user is presented with the discovered Services, and each of their Service Operations - see Figure 5.5. The user will see the name and description of each operation. This is something of a mobile cloud computing service market model, where the user has indirectly searched for a service through their CPA, and can browse the results until they find an appropriate service. The user selects the operation to be used for the task, and this is sent to the CPA. At this point, the CPA differentiates between tasks that are new, and tasks that have run before, but for now only new tasks are considered. For the chosen Operation, the CPA extracts the parameter record from the Service Method. This is sent to the CAMCS Client. A form is rendered and displayed to the user with a field for each of the parameters that are described in the Parameter record - see Figure 5.6. The user is provided with the name and description of each parameter, and is prompted to fill in each of the parameters required by that service. For parameters with Data Param Type attribute numeric, the user is restricted to numeric data entry on the keyboard. For datetime parameters, a time and date selection spinner is presented to the user. This is a Data Entry step. Once the user has filled in this form, the values for the parameters are sent to the CPA

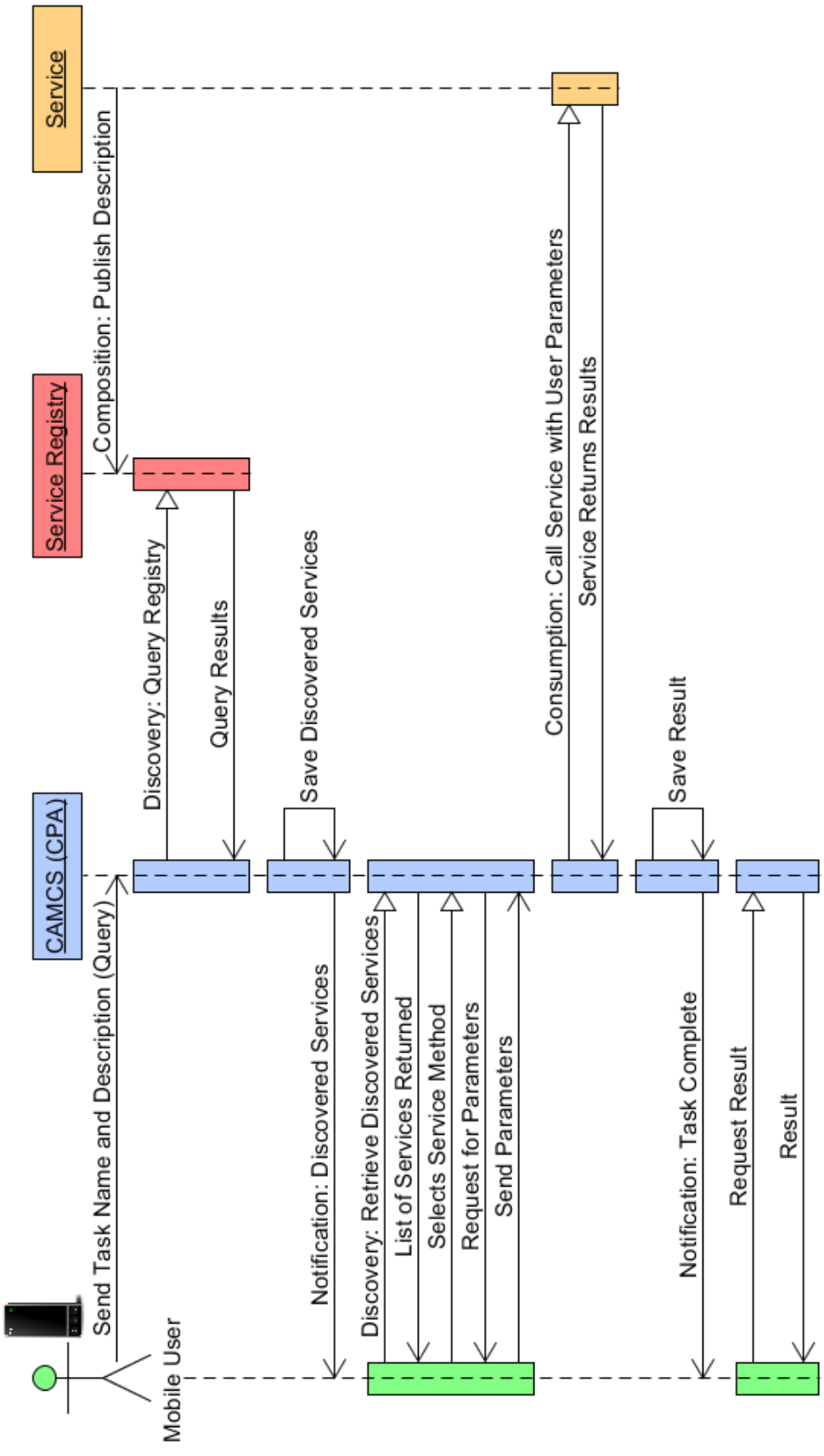


Figure 5.3: Sequence/Interaction Diagram. This shows the sequence of events in the description (composition), discovery, and consumption (execution) processes for a user task running for the first time. Tasks re-running, or tasks automatically started by the CPA, will not require service selection or parameter entry.

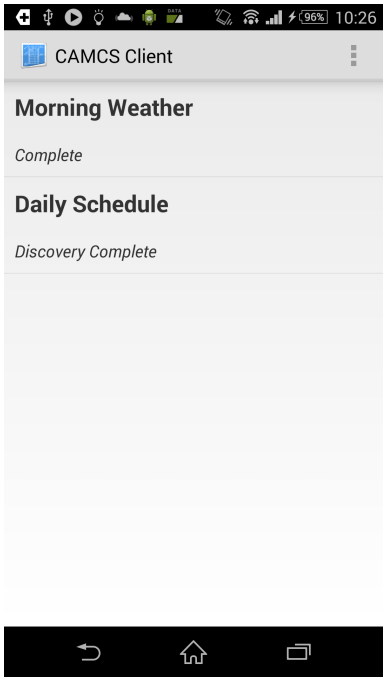


Figure 5.4: Current Tasks List. Shows all tasks currently executing at the CPA for the user. Here, this task has finished service discovery.

for storage.

If the Service Operation chosen by the user is a Service Flow, the user will receive notifications from the CPA, that the service requires more information to complete the task, for each Flow Step that is part of the Service Flow - see Figure 5.7. A Selection step is also presented as a form to the user, but no data entry is required; selecting from available options provided by the service, in the form of a radio button for example, is the only action required here by the user - see Figure 5.8. Their choice is sent to the CPA, which is then forwarded to the service. This will probably be followed by another data entry Flow Step from the Service Flow.

As described previously, services that can make use of user-context data, as described by Context Parameters in the Service Method, can receive this data from the user's CPA. This works by means of the Context Processor. This data is not collected through the form-based interface; this is gathered from a service within the CAMCS Client, as described in Chapter 4, and is sent separately at user-defined intervals to CAMCS, for storage with the Context Processor, on behalf of the user. For services that take contextual data as parameters, the CPA will gather the relevant contextual data for the user from the Context Processor, and send then as an input Context Parameter. The user must grant

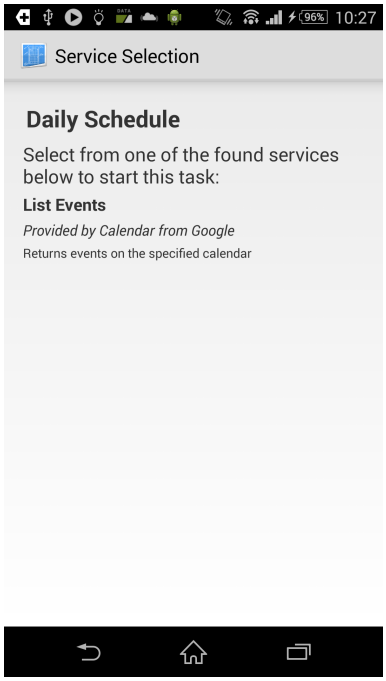


Figure 5.5: Discovered Services List. This shows the operations (service methods) provided by discovered services that the user can choose from, including their names and descriptions from the registry.

permission/consent for a task to use contextual data first, on the user interface of the CAMCS Client. See Figure 5.9.

Before task data is sent to the CPA, it is queued. The CAMCS Client observes the state of the mobile network for the quality of the connection, and depending on the evaluation (which takes into account signal strength, payload size, and current battery status), will either send the task data to the mobile user's CPA, or wait until the network quality has improved. This is explored in more detail, in Chapter 6.

5.3.2 Service Consumption

Now that the CPA has all the user-provided input values for the parameters required by the service, the CPA can consume the service, by sending it a HTTP request. As only RESTful services are considered in this work, this request will use one of the previously described HTTP methods (GET, PUT, etc.); any parameters that are declared in the Parameter record as being a URL parameter are inserted into the location defined by the placeholders in the path attribute of the Service Method being used. Parameters provided that are declared as being of type body are converted into JSON, following a "paramName": "param-

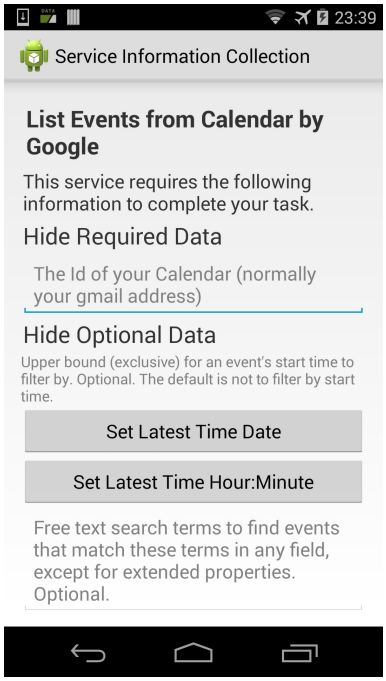


Figure 5.6: Parameter Data Value Entry. The Activity is divided up between required and optional data entry parameters, and required and optional context data parameters (out of view). The user enters the values for each parameter that the service requires for task execution. The description of parameters is displayed for the user. This is the list events API call from Google Calendar, which fetches events from a calendar. Upper/lower bound times are specified as datetime parameters, so buttons are presented for time/date selection spinners.

Value" format, and inserted into the body of the HTTP request.

If the operation is a Service Flow, this interaction can occur once for each Flow Step.

The result of the service execution is sent in the body of the HTTP response to the request. The CPA will store this result for the user, and a notification will be sent to the mobile device. Upon opening the notification, the result is retrieved and displayed on the mobile device. At this point, the task is considered complete, and is moved to a completed tasks list. If an error occurs, or some exception is thrown during task execution, the task status is changed to "Error". The user can view the reason for the error on the CAMCS Client by tapping the task entry in the current tasks list, before the task is moved to the completed tasks list of the CPA. The user is free to attempt the task again later (if, for example, the service chosen for the task is temporarily unavailable), re-run the task and select a different service, or delete the task altogether. CAMCS also logs errors for diagnosis by the developer; should CAMCS throw

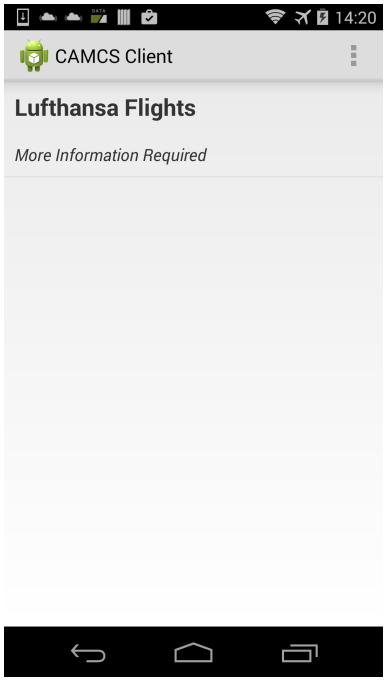


Figure 5.7: This Flight Information task is using a Lufthansa flight information service; this service is part of a service flow, and contains multiple flow steps. Here, the CPA prompts the user that more information is required for the next flow step, to continue task execution.

an exception, the developer can use this to rectify the problem and re-deploy CAMCS. Logging is provided in the CAMCS Java code by Log4j, integrated with The Simple Logging Facade for Java (SLF4J), and saved on the file system with weekly rotation.

5.3.3 Service Sessions

In cases where the selected operation is a Service Flow, the service may need to remember information about a CPAs interaction with it until the final step has been executed, and a result sent. To do this, service sessions are utilised. There are two models to support this:

Server-Side Session - The service maintains session data about each CPA currently interacting with it on its own server. For the experimental services created for this work, the services maintained a collection of *Session Data* objects for each CPA; each was identified by a randomly generated ID. The object contained a map, which used key-value pairs for required data. With the first call to the service for the first Flow Step, the session ID is sent back to the CPA in a "service_session" HTTP header. When the

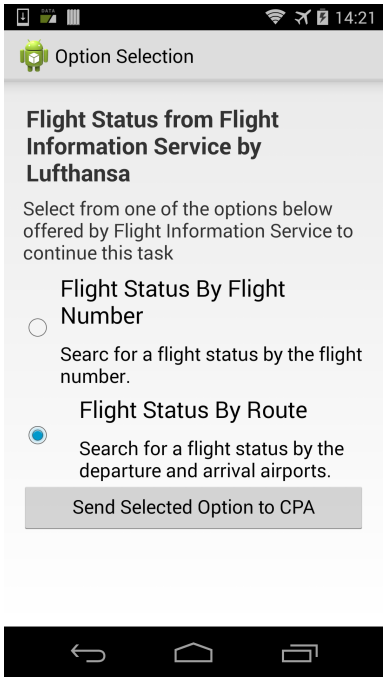


Figure 5.8: An example of a Selection Step. This flight information task is using a Lufthansa flight information service, which is a Service Flow. A selection is required by the user to continue task execution. The user can pick from the provided options to proceed to the next flow step. The options are calculated by the service, based on earlier data inputs from previous steps in the flow. The options are sent to the CPA in the session data, which presents them to the user on the CAMCS Client.

CPA sends data back to the server for any subsequent selection or data entry steps, the CPA includes the session ID with the HTTP request, so that the service can retrieve the appropriate session object from the collection. When the service is finished and returns a result, the header is removed, indicating the end of the flow.

CPA-Side Session - The CPA maintains session data stored, with the task data. This allows the service provider to effectively become stateless, except that each piece of data, stored in the Task Data map for the task (described next in Section 5.4), has to be re-sent to the service for each Flow Step.

In this implementation, these service sessions are not used for Service Operations that only contain one Service Method (i.e. they are not Service Flows).

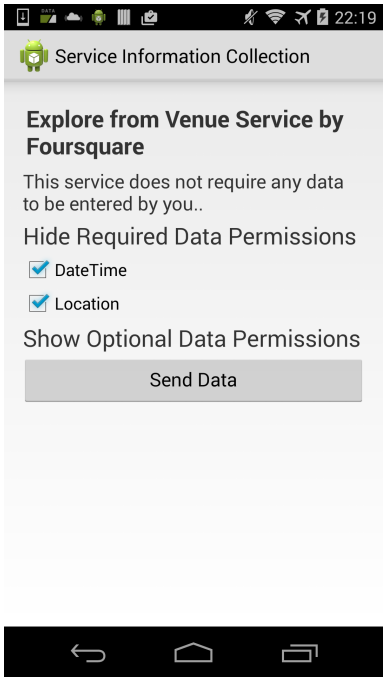


Figure 5.9: Context Parameter Permissions. For services that use context parameters, the user is not required to type context data into the form; this data is collected by the CAMCS Client separately, and sent to the CPA for storage with the Context Processor, as described in Chapter 4. The user only has to grant permission/consent for their context data to be sent to the service, which can be seen here. The service cannot be used if the user denies permission for a required context parameter to be shared with the service.

5.4 CPA Task Model: Features and Implementation

The task model of the CPA is implemented around the disconnected operation principle. Once task details have been sent to CAMCS, they are stored with the CPA of the user, and the mobile device does not need to remain connected to the cloud deployment for the task to execute. The structure of a task will now be presented, followed by the task execution process, along with the automatic task execution model. The storage of results, and their presentation, are also described.

5.4.1 Task Structure

The CPA and all of its related data (such as details for users and tasks) are stored. Once a user has sent a task description to CAMCS, this task is created at their CPA, and placed into a current tasks list. This is represented as a subdocument within MongoDB. Each task contains several attributes:

- Name: a user specified task name, entered into the CAMCS Client by the user on the mobile device
- Query: the query associated with the task, which is the description entered into the CAMCS Client by the user on the mobile device. This is the query string sent to the registry for service discovery.
- Discovered Services: a list of discovered services returned by the registry in response to a query search, so that the user can choose a different service to re-run the task in the future, without performing service discovery again (unless explicitly requested).
- Service Record: this is the service selected by the user that will be used to complete the task, as chosen from the discovery results. Contains the technical data required to invoke the service.
- Operation Name: this is the name of the operation (ServiceMethod) offered by the service that the user has chosen to use for completing the task.
- Results: a list of results. Tasks can be executed more than once with different results each time.
- Task Data: a map which stores the values that the user has provided for each parameter required by the service. Note that user context data is not stored with the Task Data, as this is always taken directly from the Context Processor when required.

5.4.2 Task Implementation

Within CAMCS, a runnable task is defined by a TaskExecutable class. The Quartz Scheduler [41] is used in this work, which provides a scheduler for "jobs". TaskExecutable subclasses the Quartz Job class, and is executed in its own thread of execution. All tasks in CAMCS, from all CPAs, are handled by a Task Handler, which is responsible for taking the tasks from the CPAs, and starting them as Quartz jobs using the TaskExecutable. The required information to execute a task is passed by the Task Handler to the job by means of a map. This data will include all the required data outlined previously.

Tasks can be executed either by an explicit request from the user (running a new task or explicitly re-running a completed task), or as an automatic task,

whereby the CPA chooses to run a task without any explicit user request.

1. *User Requested Task Execution*

   A user creates a new task with a name and query using the CAMCS Client. Once this is sent to the CPA, the CPA starts the task execution by passing the task to the Task Handler. The Task Handler checks if a valid Service Record has already been associated with this task. Being a new task, this will not be the case, and so the Task Handler will begin service discovery by taking the user provided query for the task, and contacting the MCC service registry.

   Once the user has selected the appropriate service and operation, and task execution has completed, the task result is returned to the CPA from the Task Handler, and the task status is set to complete. When the user views the task result on the CAMCS Client, the CPA moves the task from the current tasks list to the completed tasks list.

2. *Automatic Task Execution*

   The benefit of the CPA model is the ability to perform work for the user without an explicit request. This furthers the disconnected operation goal. In this case, even if the user is disconnected from the cloud deployment, and hence the CPA, work can still take place for the user. Based on times specified by the user, the CPA can also automatically run a previously completed task again, using the previously chosen service and provided parameter data values.

   In this implementation, when the user has re-executed a task a given number of times (three currently), the CPA sends a notification to the user, asking would they like to schedule the task to run regularly on an automatic basis. The user can set the days of the week, and times, when the task should automatically execute - see Figure 5.10. The repeat task data is stored with the CPA. This data is used to schedule cron triggers, with Quartz. These use regular expressions to define when a trigger should run to start a given job (or task in this case). The user can stop the automatic task execution at any time. Whenever an automatically repeated task is executed, the result is added to the results list of the task with the given date and time of execution. The task is moved from the completed tasks list, back to the current tasks list. A completion notification is sent to the user. Another future possibility is that tasks can

automatically run based on patterns of when the user has explicitly requested that the task should run in the past. This way, the user would not have to manually specify when a task should execute.

5.4.3 Task Results

Tasks can run several times. The results of a task can differ depending on the different times it is executed, or if the user changes the input parameters. Therefore, each result is stored with a timestamp of execution. When the user opens a task, they can view the results for each of the previous executions.

A result may range from textual data, (a hotel booking reference number), or something numeric (statistical results from a data processing service). To provide flexibility and customisation for service results, a HTML solution was used. Companies such as Amazon could advertise other products that they offer within a task result HTML page. The Result Template of a Service Method record contains a URL to a template HTML page. This page, which should be stored on a publicly available server, makes use of the JSON2HTML library [81]. This library uses JavaScript to convert JSON data into a HTML representation. This is accomplished by means of specifying a transformation, which will convert a JSON string to HTML. The transformation is already stored in the HTML template page, along with any other mark-up/formatting details (CSS, JavaScript) that the service developer has chosen to include in the page at development time. When the CPA has received the JSON result data from the service, it will fetch the result template page given by the URL. Using the JSoup library [62], the JSON service result is written into the <head> section of the page. The result HTML page is added into the result list for the Task, and stored in this marked-up format.

Upon task completion, the user is notified of the result; this is displayed in an Android WebView - see Figure 5.11. When loaded, the transformation is applied to the JSON result data.

5.4.4 Task Re-Runs

As mobile users create tasks and discover services for them, on a day-to-day basis, mobile users will more often re-run old tasks from the task history, rather than create new ones. As task data (parameters) are already stored with the

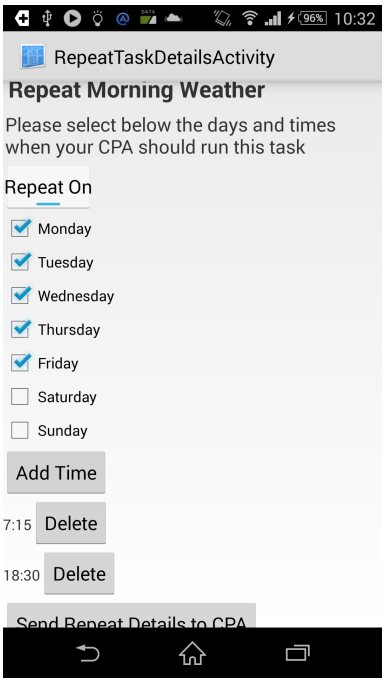


Figure 5.10: Automatic Task Repeat. The user can enter the date and time details for when the CPA should automatically repeat a previously completed task. The Quartz Scheduler within CAMCS is then scheduled to re-execute the task, with results returned to the CPA.

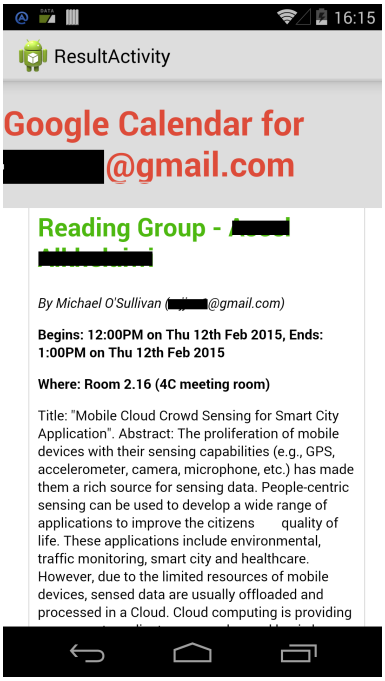


Figure 5.11: Task Results Display. The result of the task execution is displayed as a HTML webpage using an Android WebView (personal details hidden). Here, the transformed output from the Google Calendar List Events API call is shown for the daily schedule of the user.

task at his/her CPA, this becomes quicker and cheaper, as the user does not have to send parameter information again, unless they specify that they want to change a parameter input. Therefore, a user can either re-run a task as is, or, re-run it, and modify the input parameters. For task re-runs, fresh context data, if available, is always taken from the Context Processor. Service discovery will not take place again (unless the user specifies they wish to choose a different service); task execution with a Quartz job using TaskExecutable will take place immediately.

For re-running a task where the user selected a Service Operation with a Service Flow, selection steps must be repeated by the user again, even if they have chosen not to change the input parameters, and they will be prompted with a notification from the CPA to do so. They do not need to repeat data entry steps again (unless they specify that they want to).

5.5 User Privacy and Security Considerations

As a trusted third-party representative of mobile users in the cloud, a CPA must provide authentication and authorisation mechanisms to ensure that no user of CAMCS can access/use the CPA of another user. Additionally, all data supplied by the mobile user for the purposes of completing a task is stored with his/her CPA in the cloud. This can include personal data, and in a cloud environment must be secured to ensure privacy. Furthermore, the results of tasks must also be stored securely.

5.5.1 CAMCS Authorisation for User CPAs

To register with CAMCS, the mobile user is required to provide an email address, and a password. Registration takes place on the CAMCS Client with the Registration activity. The email address and password supplied during registration is stored in the backing MongoDB database used by CAMCS. Passwords are encrypted using the *bcrypt* hashing algorithm. The implementation of the algorithm uses a randomly generated 16-byte salt value, and produces a hashed password encoding of length 60 for storage in the database. The algorithm is designed to run slowly over several iterations, based on a *strength value* ranging from 4-31. The closer to 31 the strength value is, the longer the

algorithm will take to run; it is designed to deter hackers from performing repeated password checks. The default value used for strength is 10.

Successful registrations will automatically log the user into the CAMCS Client. The user can logout and login again at any time. Logging out will prevent access to all CAMCS functionality, including CPA access. This means that while logged out, new tasks cannot be sent to the CPA, and all current and completed tasks cannot be viewed, including results. Logging out will not impact the CPA in completing any work. A consideration for future work can include login timeouts, such that after a certain period of time, the user is automatically logged out, and forced to login and authenticate again through the CAMCS Client. Otherwise, if the mobile device was ever stolen or shared with another user, the third-party could access all data and tasks stored with the user's CPA.

All requests sent from the CAMCS Client to CAMCS are subject to HTTP basic username/password authentication, implemented using the Spring Security framework [121]. Each request to CAMCS, be it creating a new task, providing service parameters, or retrieving a task result, is authenticated by the email address and password combination of the user, which must be provided with each HTTP request. Only if authentication succeeds, will any CPA operations take place. If authentication fails, no operations can be called on the CPA by the user. This mechanism also prevents one user from accessing the CPA of another user, as each CPA is identified by the email address used for registration. TLS, which is not used in the current implementation, can be added in future work to secure communication over the network, and prevent replay attacks on the HTTP authentication mechanism. This was not implemented due to a technical problem with the Spring Framework RESTful HTTP client on the Android mobile device; the client failed to make connections to servers with unsigned certificates when using TLS, and so for simplicity, TLS or SSL were not used,

A combination of private and public key security is used by CAMCS, based on work by Michael Lones [84]. When registration is completed successfully, CAMCS will generate a new 256-bit AES symmetric key. This key is then encrypted, using the RSA public key algorithm. CAMCS returns this RSA-encrypted AES key to the user on the CAMCS Client, and also stores a local copy. This key is used when the user requests that his/her CPA starts task execution, and when the user wants to retrieve the result for a task. This will be explained in the next subsection.

5.5.2 Privacy of User Data

The task model of CAMCS requires user data in order for CPAs to complete task work for the user. All data provided for completing tasks, along with task results, must be stored privately, such that no other user of CAMCS can access that data, i.e. one CPA can only access task data and results, for tasks assigned to that CPA. Any individual CPA must not be able to read user and task data stored by any other CPA. In terms of access from a running copy of the CAMCS Client, this is handled by the authentication mechanisms described in the previous subsection.

Additionally, the use of private and public key security, discussed in the previous subsection, is used for the purposes of encrypting the task result HTML files. The operation of starting the execution of a new task, once parameter data has been provided, requires the further step of the user providing the RSA-encrypted AES key, that was provided to the mobile device upon CAMCS registration. This is sent with the parameter data. Once the task has finished executing, the CPA will decrypt the RSA-encrypted AES key, using it's own private key. Once the user's AES key has been decrypted, this is used to encrypt the result HTML file generated by CAMCS.

When the user wishes to retrieve and view a task result HTML file, the RSA-encrypted AES key must be provided from the mobile device again. This is used by CAMCS to decrypt the result HTML file, so that it can then be sent to the user. If the user provides an invalid AES key, then access to the result file is prohibited.

Features are also provided by CAMCS to store task results with a third-party cloud storage provider, such as Dropbox. Dropbox provides 256-bit AES encryption of all files stored with a user's account. If a user is conscious of where his/her data is stored in the cloud, such as keeping all data in one place, or chooses to trust one provider over others, then this feature is very useful.

The mechanisms described and implemented for user authentication and data privacy are basic; they are not a complete security solution for CAMCS. For example, an important consideration must be made in future work. As the user's data is stored with his/her CPA, it can be considered a single point of failure, such that it is a single target for attacks. A complete security evaluation with experienced researchers in cloud security, can be performed to identify any remaining vulnerabilities and risks associated with the current solution.

It is important to note that many strict security measures can inconvenience the user when accessing their CPA with the CAMCS Client, such that time and effort required to authenticate frequently can be considered detrimental to the user experience goal of the thesis project. Therefore, a balance must be found between security, and ease of accessibility, when a complete security review is undertaken.

5.6 Conclusions

In this chapter, a mobile cloud based solution for enabling description, discovery, and consumption of mobile cloud services was presented. Several contributions have been made. First, the service registry was presented, which stores service descriptions in a mobile cloud service description format, which was designed with user-oriented discovery in mind. It will allow users to be a part of the discovery process, not previously practical with mark-ups such as XML. This structure of this description format was shown, including the Service structure, which supports simple one-call services, and more complex services requiring multiple calls, as part of a Service Flow. Secondly, the discovery process allows the user to benefit from cloud-based services through the device.

How the solutions are used with CAMCS and user CPAs was shown; a mobile user can search for services to complete tasks; the user-oriented mobile cloud service descriptions from the registry are presented to the user for this, bringing them into the discovery process. Also shown was how data for service parameters (data inputs and context inputs) are collected from the user, and how task results from services are stored and presented, and possibly ranked with user context data from a social network such as Facebook, should the user allow it. Support for server-side and CPA-side sessions was also presented to support Service Flows. The user task model for CPAs was also presented, as well as automatic task execution. Automatic tasks enable the design objective of disconnected operation for CPAs.

User security and privacy considerations with CAMCS was also shown in terms of user authentication and authorisation. Each user can only access their CPA once they are registered and logged in on the CAMCS Client. As CPAs are tied to the user's registered email address, a user cannot access or call op-

erations on other CPAs. The use of secure public and private keys was also shown for starting tasks, and encryption/decryption of task result data. The user's AES key must be presented with HTTP requests in order for the user to start tasks, and retrieve task results, otherwise the user is blocked from access.

While this solution uses existing technologies and protocols, due to the user-oriented discovery process and services descriptions, this work opens a new direction in accessing cloud services by mobile users.

In the next chapter, resource usage considerations for mobile cloud computing, in terms of energy usage, bandwidth usage, and cloud-server resource usage, are analysed. Models will be presented, and these will be applied to CAMCS and other existing mobile cloud computing solutions for comparison, to understand how these solutions make use of the limited resources available.

All sections of this chapter were based on [100], with the following exceptions, based on new functionality added since the original publication: Section 5.2 was heavily modified and extended, Subsections 5.2.3.1 and 5.2.3.2 were added as new; Section 5.3 was extended, Subsection 5.3.3 was added as new, and Section 5.5 was added as new. New figures were also added that were not present in the original publication, and some figures were modified to show developments/modifications since the original publication. Various modifications have been made to the text throughout the chapter, to fit with the flow of the thesis.

Chapter 6

Integrating Mobile and Cloud Resources Management

6.1 Introduction

In the complex equation of communication between the mobile device and the cloud, the management and use of the limited resources available on the device plays a central role. Many mobile cloud solutions published in the research literature require continuous, high-quality connectivity to the cloud, and involve large amounts of data transfer. There are several resource considerations that come into play. For example, the continuous transfer of data between the mobile device and the cloud will come at a high energy cost from the device battery. This cost only grows if the device is using the cellular network connection, due to variation in signal level, along with the variable network bandwidth, and resulting data-rate. It may not be feasible for a mobile device to exhaust such energy continuously for the duration of the cloud interaction. As another example, as the bandwidth of the network connection may vary over time, levels of performance from the resulting data-rate cannot be guaranteed. This is very significant in the areas of real-time applications where minimal latency is crucial. As offloaded tasks and applications execute on cloud infrastructure, cloud resources such as storage are used to compliment the local resources of the mobile device. The management of resources allocated to the mobile cloud at the cloud infrastructure also has to be taken into consideration. If all mobile device users offload large and complex data to support such operations, then the management of energy and physical resources at the cloud must be a

factor.

The objective of this chapter is to devise and examine the best practices in resource management, in the area of mobile cloud computing, and to derive a model for data and task offloading to the cloud, while considering the limited available resources. Some of the previous approaches and models to energy and bandwidth resource management in the research literature will be examined, in the context of mobile cloud. These models will be modified with mobile cloud considerations, and these will be applied to existing approaches that have been taken for mobile cloud implementations, to understand how these approaches utilise the resources being studied. From this, the best practice approaches for solutions in the mobile cloud domain for managing these resources will be devised. These approaches and the resulting implications from the models will be contested with CAMCS, to highlight how a disconnected approach from the cloud can be of great benefit to the conservation of resources on the mobile device, by minimising the usage of the scarcely available energy and bandwidth. CAMCS can avoid a large allocation of resources to the mobile cloud at the cloud server side, by the use of cloud services in SOA fashion, rather than allocation of entire virtual machines.

Furthermore, several experiments performed with mobile devices to derive a decision model for data and task offloading to the CPA will be presented. This model considers the available resources and the nature of the data to be offloaded, as part of the offload decision process, without requiring additional overhead from the mobile device. As a result of the adoption of these best practices and the derived model, CAMCS can achieve the goal of efficient resource management in the mobile cloud.

6.2 Mobile Cloud Energy Management

Attention is now turned to examining the management of energy in the mobile cloud computing domain. The limited energy provided by the mobile device battery is used by the hardware on the mobile device, as well as the software. Some software and services running on the mobile device can drain more energy than others. This, combined with large displays and the network communication make it a key resource. It was described in the introduction how many mobile cloud solutions require a continuous connection to the cloud-

based infrastructure. This continuous connectivity is very undesirable. First, the energy usage of the antenna of a mobile device to send and receive data is examined, before augmenting this usage with other aspects. This model is then applied to the existing solutions, before being applied to CAMCS.

6.2.1 Antenna Energy Modelling

As a starting point for analysis of what factors are important for energy modelling, Heinzelman et al [50] described the energy usage of the antenna of a micro-sensor in a wireless sensor network to send data, as follows:

$$\begin{aligned} E_{Tx}(k,d) &= E_{Tx-elec}(k) + E_{Tx-amp}(k,d) \\ E_{Tx}(k,d) &= E_{elec}k + \epsilon_{amp}kd^2 \end{aligned} \tag{6.2.1}$$

where $E_{Tx}(k,d)$ is the energy required to transmit a k-bit message a distance d to the base-station. The energy usage to receive a message is given as follows:

$$\begin{aligned} E_{Rx}(k) &= E_{Rx-elec}(k) \\ E_{Rx}(k) &= E_{elec}k \end{aligned} \tag{6.2.2}$$

where $E_{Rx}(k)$ is the energy required to receive a k-bit message. The assumed energy dissipation E_{elec} is the same as in equation 6.2.1, but can vary with different radio antennas. While these equations were originally applied by the authors to wireless sensor networks, mobile devices follow the same model of transmitting data from an on-board NIC, which requires an internal antenna with a signal to a cellular base-station.

In the Bartendr project, Schulman et al [120] estimates the energy cost for transmitting a stream of data. The power required by the transmitter to send this data will vary depending on the strength of the signal of the mobile device; the weaker the signal, the more power that would be required. This also corresponds to equation 6.2.1, even though it does not consider signal strength as a factor. This is understandable, as a sensor position may be fixed in the wireless sensor network, unlike a mobile device, which will constantly move. However, even though signal strength is not a factor, the amplification will increase as the signal strength weakens, in an effort to increase the quality of the

signal.

Continuing to look at Bartendr, the system tries to schedule a data stream of size S to be sent over a given time period T. They divide up the data-stream into N frames of fixed size chunks, and the time period T is divided into slots. One slot is defined as a time period when one frame can be sent. The data rate and width of the slots will vary; the predicted signal strength and the observed median throughput over the time interval T are used to estimate the power consumption and width for each slot. With a given predicted $signal_\zeta$ in slot ζ, they estimate the communication energy as:

$$\frac{Signal_To_Power(signal_\zeta)\frac{S}{N}}{Signal_To_Throughput(signal_\zeta)} \tag{6.2.3}$$

where $Signal_To_Power()$ maps the given signal strength to a power value, and $Signal_To_Throughput()$ maps the given signal strength to a throughput value. The mapping values are based on empirical measurements carried out by the authors, by examining and recording the power required for different transmitter states. They found that the power values were smaller for a stronger signal to the cellular network, and that the signal itself varied by location, as one would expect.

In the TailEnder project work by Balasubramanian et al [7], their experiments recorded the actual energy values, along with the times, for the ramp-up and tail energy, using both cellular 3G and GSM networks, as well as Wi-Fi. These were used to construct an energy model for each network, where the energy to transfer an x-byte file was given as:

$$R(x) + M + E \tag{6.2.4}$$

where $R(x)$ is the sum of the ramp energy, and the energy to transmit the x-byte file, M is the maintenance energy required to keep the transmitter on, and E is the tail energy. For the Wi-Fi connection, there is no ramp or tail energy as the interface is kept active, but there is an energy cost for scanning and association with an access point. An interesting observation, is that the tail time for the 3G connection with their tests lasts 12.5 seconds. This value is actually set by the network provider. With such a high value, they observed that up to 60% of the total energy that went into a file transfer on the 3G connection, was

from the tail energy alone, where no actual data-transfer was taking place. To contrast, the tail time from the GSM connection was only 6 seconds.

6.2.2 Mobile Cloud Considerations for Energy

Looking at these other works in the previous subsection, one can see there are many factors in modelling the energy required for network communication on the mobile device, such as the size of the data, the distance from the base-station, the throughput, along with the ramp and tail energy characteristics of the network interfaces. In applying these parameters to the mobile cloud domain, how the mobile cloud makes use of the network interfaces must be considered. Does it differ from traditional-client server approaches where one simply makes a request and receives a response?

The idea of the mobile cloud is to offload complex calculation and applications to the cloud. In Chapter 2, several approaches to this were described. One such approach is code offload [21] [19]. Such approaches package up the code of some application, normally methods in object-oriented languages, along with the state of objects, which are then transferred to the server. Once the server has completed execution, the results are returned, in the form of the changed state of the objects; these need to be merged with the local copies of the objects on the mobile device.

The work by Kumar and Lu [72] has already proposed an energy saving model for the decision to offload some computation to the cloud from the mobile device. The energy saved on the mobile device by offloading the computation to the cloud is given by:

$$\frac{C}{M}(P_c - \frac{P_i}{F}) - P_{tr}\frac{D}{B} \tag{6.2.5}$$

where C is the number of instructions, M is the number of instructions the mobile device can execute per second, P_c is the power required to execute the computation at the mobile device, P_i is the power the mobile device uses while idle, F is the number of times faster the server is compared to the mobile device, P_{tr} is the power required to transmit the computation to the cloud, D is the number of bytes that make up the data, and B is the network bandwidth. Energy is saved when the formula produces a positive result.

This formula does not consider the technicalities of actually performing a code

offload. For a method code-base size S_m, outgoing object size S_{oo}, and incoming returned object size S_{oi} then going by equations 6.2.1 and 6.2.2, the total energy cost of offload E_{total} can be considered as:

$$E_{total} = n((E_{elec}(S_m + S_{oo}) + \epsilon_{amp}(S_m + S_{oo})d^2 + E_{prof}) + (E_{elec} + (S_{oo})) + E_{merg}) \quad (6.2.6)$$

where n is the number of times a method offload occurs, E_{prof} is the profiling energy required by the decision model used to determine if the offload should take place (based on current network quality), and E_{merg} is the amount of energy used at the mobile device to merge the changed objects back into the virtual machine (such as adding new objects, updating existing objects, and removing old objects, along with the corresponding memory allocations/deallocations). Equation 6.2.6 takes into account the energy requirements identified in equations 6.2.1 and 6.2.2 for the offload and the return of the result to the mobile device. If one considers that n may occur 10 times in the course of one application execution, then the energy usage clearly increases ten-fold with such an approach.

Consider the Cloudlet approach. Here, there will be energy expended to transfer the VM overlay to the local cloud infrastructure. Eventually, any changes made to the application settings and data will ultimately need to be returned to the mobile device before the user leaves the venue where the Cloudlet is located. Looking at equation 6.2.1, this energy usage can be estimated as:

$$E_{total} = O + I + n((E_{elec}(k)) + m(E_{elec}(k) + \epsilon_{amp}(k)d^2) \quad (6.2.7)$$

where O will be the initial energy to transfer the virtual machine overlay to the Cloudlet, and I will be the energy required to transfer the modified overlay back to the mobile device. Energies O and I could correspond to equations 6.2.1 and 6.2.2. n will factor up the energy consumption for the number of times the Cloudlet sends data to the mobile device. It is hard to predict this value. It could be extremely high, if the Cloudlet actually displays the visual output of the virtual machine on the mobile device; consider how often the

display GUI would be refreshed, and this would need to be updated on the device display, possibly even if no change has taken place. m will factor up the number of times the mobile device sends data to the Cloudlet, which could be much smaller than n. The primary energy saving aspect of the Cloudlet approach is that d will be very small by contrast compared to sending data to a cellular base-station for offloading to remote cloud infrastructure, as a Cloudlet is defined as being one Wi-Fi hop away from the mobile user.

Consider the disconnected approach of CAMCS. If a user wishes to offload some task to their CPA, to send the task, there will only be one outward communication to the cloud, describing the task. This will be a simple message of k bytes, with no application code. There will be no immediate response data except for a request acknowledgement. When the task is complete, the result will be pushed to the CAMCS Client. This will be a message of m-bytes, which simply carries a result message, with no changed object states.

Therefore, the task offload energy is given by equation 6.2.1, and the response, given by equation 6.2.2, will consist only of a HTTP Status with no data. When the task has completed, the result will also be given by equation 6.2.2, $E_{Rx(m)}$. As the focus here is on disconnected operation, there is no continuous communication that needs to be factored into the formulas as in equations 6.2.6 and 6.2.7, along with no additional overhead (such as initial virtual machine offload).

6.3 Mobile Cloud Bandwidth Management

During operation, the mobile device will be connected to a multitude of different networks, be it a Wi-Fi network, or different base-stations in the cellular network. Bandwidth allocation, along the lines of guaranteeing QoS, is a topic that has been researched extensively. However, it is difficult to predict what the bandwidth will be on a given network.

At the cellular network level, the bandwidth will vary depending on the type of cellular connection used (GPRS, EDGE, 3G, HSPA). While each of these has a defined upper limit on the bandwidth specified, the actual bandwidth that the user will experience in practice will vary, and will depend on aspects such as signal strength, interference, and location.

At the wireless LAN level, depending on the different version of the 802.11 that

a Wi-Fi network adheres to (b/g/n/ac), the user will experience a far more stable bandwidth with little variation from the nature of the network; what may cause trouble is the number of users accessing the Wi-Fi network from the same access point.

The question for mobile cloud is that given the variable nature of the bandwidth of these networks, how can the available bandwidth be managed in such a way that this will not impact the user experience with the mobile cloud. At first glance, since no given bandwidth can be guaranteed with the mobility factor of a mobile device, solutions must be designed to use as little bandwidth as possible. This can be a troublesome aspect given some of the existing approaches to mobile cloud that have been presented, due to the required continuous connection, and large data transfer.

Additionally, from the perspective of a software developer creating mobile cloud applications, they will not have any control over bandwidth allocation to their applications, as this work is typically carried out at the network layer [110] [75] [77]. The allocation is decided at the base station or access point. If the developer anticipates that an application requires a large amount of data bandwidth, there is no ability to build a request for high bandwidth allocation, into the application code.

6.4 Bandwidth Requirements

It cannot be stated with certainty what bandwidth is required by many of the mobile cloud approaches reviewed, however an estimate can be made on how much will be used based on how they use the network connection.

Approaches that require a continuous connection will undoubtedly require more bandwidth than what is used in an approach with disconnected fashion. Looking at the Cloudlet approach, where the network connection is repeatedly used to transfer the output of the virtual machine to the mobile client, then the expected bandwidth usage is going to be large. On a Wi-Fi network, which is the premise of the use of Cloudlets, the bandwidth can be expected to initially be large, and relatively constant, compared with the data connection; however this may change as the number of users of that Cloudlet grows. As the limited bandwidth of the Wi-Fi connection is used up, the user experience will suffer from the time delay induced as a result of the drop in the data rate. The same

principle can be applied to cellular networks where remote display technologies may be used, except in this case that the bandwidth will be very small to start with, and shared by far more users of the base-station. As a result, the main concern with these approaches is that they have a requirement that a certain amount of bandwidth must be available for use when required, to ensure that they work as expected for the user, without any delays from sub-optimal bandwidth allocation.

Approaches such as code-offload and application partitioning may not be as dependent on the amount of bandwidth as Cloudlets or remote displays. A continuous connection is required to either the cloud infrastructure running the virtual machine, or to the other computing nodes in the network where the application partitions have been sent to, but data does not need to be repeatedly sent over the network connection. Once code or partitions have been offloaded from the mobile device, it need only receive the results back when the remote execution is complete. The actual data to be sent/received between the mobile and the infrastructure is the serialised application code and objects. These will likely be much smaller in size when compared with the graphical output of virtual machines. Also, in the way that the updated graphical output of a virtual machine needs to be sent to the mobile continuously (even if the display has not changed), the same does not apply for these approaches, because the mobile device only receives the results of a completed execution. As a result, unless a disconnection occurs, there is no negative user experience impact, as it is not as time sensitive as the virtual machines.

Considering the disconnected operation aspect of CAMCS, then the bandwidth is something of little concern. Users offload a task description to their CPA within the CAMCS. The description is just made up of a limited number of ASCII characters that describe the task, the type of task, and any parameters. Results in Chapter 8 will show that the size of this data is very small, when compared to data sent on code-offload or application partitioning approaches, so CAMCS brings the same advantages as far as bandwidth is concerned as these approaches; low data usage, no time sensitive delay, and no continuous use of the data connection.

Other features of CAMCS further reduce the requirement for data transfer, and hence bandwidth utilisation, such as the task history; a user can simply signal an old task to run again with a HTTP request signal using the task ID, without sending all the parameter data again (unless the user wishes to change any

parameters), therefore reducing the request size and the bandwidth utilisation even further. Also, the automatic task execution model, where the CPA runs requests for the user, without an explicit requires; therefore there is only one communication required, that is fetching the result HTML page.

There are scenarios where the size of the data sent to the CPA can be very large, such as presented in the file synchronisation application model to be presented in Chapter 7. For example, if the user sends large files such as images or datasets to the cloud for some processing task, or possibly for storage, then the size of the data to be sent to the CPA can be expected to be larger than task descriptions. Next, how the available bandwidth can be managed in such a situation is discussed.

6.4.1 Bandwidth Management

When considering bandwidth utilisation as part of an integrated user experience of mobile cloud applications, the solution must assume that available bandwidth is at a minimum. Use of this bandwidth must be mediated.

Where a high amount of bandwidth is required, it can possibly be reserved in advance of needing it, if it can be reasonably anticipated. Cloudlets or remote display approaches could reserve and allocate an amount of bandwidth as needed to cope with their expected demand. In this way, it could ensure bandwidth is available so that they function optimally with no time delay. Of course, this may not be fair to other users of the Wi-Fi access point, and can reduce the amount of bandwidth available to them. A financial cost may also be incurred for this reserving of bandwidth in advance. However, reserving bandwidth will still fail to deal with unexpected bursts in required capacity.

Code-offload and application partitioning approaches may not need to reserve bandwidth in advance. They can simply make the best use of bandwidth already available, which can speed up the offload of the code, and return of any changed object state. It may be argued that if these approaches did reserve bandwidth, it may still reduce the wait time of the user, but it may not reduce the time enough to justify the high cost of reservation.

It has already been discussed how the mobile uses little bandwidth as a result of the small request size made to the CPA in the cloud, and because of its disconnected operation, it is not time sensitive. However, despite the fact that

little bandwidth is used, bandwidth that is available must be used intelligently. Furthermore, there are cases when the size of the data sent to the CPA from the mobile are very large, such as when sending large files to be stored as described earlier. In this case, if bandwidth is low, then there is certainly going to be a delay for the user. If using the cellular network rather than the Wi-Fi, it will take even longer. The additional cost will be incurred to the user if for example, the low bandwidth results in not only a slow upload, but if data needs to be re-transmitted across the network if there are errors.

The thin client application running on the mobile device, must adapt to the current situation. If, for example, the available bandwidth is low, transfer of a file must be deferred to a later point in time, when the available bandwidth increases (the same principle applies to other resources such as energy, if the battery is low, upload should be deferred). This has already been implemented in many apps today, such as Dropbox, where files will only be uploaded on a Wi-Fi connection if the user chooses. However, if there is a need to upload on a cellular network, this should be possible, but only when the bandwidth is suitable. The same principle of deferring upload can be used on a busy Wi-Fi network, or when the Wi-Fi signal is poor. This will benefit other users of the network, as bandwidth is not used up with a large task upload that may fail if the signal is very poor and ultimately drops.

The offload of tasks that may not be urgent can be deferred. The user can specify how urgent a task is, and even give a time when it should be completed by. Urgent tasks, or tasks that have a specified completion time coming up, can be offloaded immediately, even on a network with poor bandwidth, if this is the case. Tasks that are not urgent, or tasks with a specified completion time at some point later in the day, can be deferred until the bandwidth is available.

In scenarios where the battery energy is low, it may consume energy to actively test the bandwidth available on the connection. Even though data transfer is small, the network interface must be activated. In such cases, it is best to apply a policy based on the connection (Wi-Fi or cellular), and the current measured signal strength, which can normally be determined from an API on the mobile operating system. This is explored and developed further in Sections 6.6 and 6.7.

6.5 Cloud Resource Management

This section will look at how resources on the cloud can be managed while provisioning services to mobile clients for the various mobile cloud approaches. At the cloud, one will find resources at the different "Service Layers" (IaaS and PaaS) that can be provisioned as resources to clients, such as virtual machines providing compute capacity, storage in the form of file systems and databases, memory, and networking capacity. Developers can also find resources for developing and deploying tools, such as application runtime environments, message queues, replication and backup facilities, and load balancers for scalability.

6.5.1 Cloud Resource Cost Model

The services offered by cloud deployments are normally offered through the interface of a virtual machine (VM); that is, for each resource, such as a hardware resource (e.g. CPU), or a developer resource, (e.g. a database), the user or application interacts with them by using a VM. CPU time is allocated to each VM for computation tasks, and database systems run on these VMs. To run each VM, a hypervisor is required to sit on top of the hardware. Therefore from an energy and financial standpoint, the number of VMs required for the mobile cloud requirements, and how much of the physical resources are required, will be of concern.

It is difficult to precisely approximate VM hardware costs, as described in [11]. Therefore, for each VM running on a server in the cloud, a simple cost model can be estimated for the purpose of this analysis, as the summation of the costs required to run the physical resources required on the server. There will be a CPU cost, C_{cpu}, network input and output costs, C_{input} and C_{output}, a storage cost, C_{stor}, and a memory cost, C_{mem}. All of these costs can be viewed as both energy costs and financial costs, as ultimately the financial cost will be the energy required to power these physical resources. If n is the total number of VMs, possibly different, running on a server, the total server cost, C_{server_total} for a cloud server is:

$$C_{server_total} = \sum_{i=1}^{n} (C_{cpu} + C_{input} + C_{output} + C_{stor} + C_{mem}) \tag{6.5.1}$$

For m, the number of servers required on the cloud for scalability, equation 6.5.1 can be factored up by m. As a result, to keep the energy and financial cost as low as possible, as much of the hardware costs C as possible must be minimised, as well as n and m.

In the next section, how each of the discussed approaches to the mobile cloud makes use of the server resources will be examined, and costs will be estimated based on equation 6.5.1.

6.5.2 Mobile Cloud Usage

Several of the previous approaches take different paths in terms of requirements at the cloud side, depending on what role the cloud will play.

With VM-based approaches such as the Cloudlets and Remote Display technologies, virtual machines capable of running end-user operating systems with GUI's such as Microsoft Windows or Linux distributions such as Ubuntu or Red-Hat will be required. They will also require resources such as imaging and storage, to store a user's VM image and for storing the image itself and any files. Cloudlets, as described in Chapter 2, only have a base VM stored on the infrastructure. The overlay image, containing the user's applications, data, and settings, is stored on the mobile device of the user. It is combined with the base VM to form the full VM at runtime. As a result, it will not need as much storage at the cloud deployment. However, the overhead of having to transfer the overlay images between the mobile and the server may be undesirable. Furthermore, breaking up and combining the base and overlay VMs may need special modification of the images and operating system software. There is also the issue of having to determine what should be offloaded to the remote cloud infrastructure, and what should be kept local on the Cloudlet. The energy requirements for this decision making process, along with merging the VMs, are unclear.

From this, one can determine that for Cloudlets, there will be two "server" costs, the costs incurred by the local Cloudlet, and the costs incurred at the remote cloud server. Modifying equation 6.5.1 to categorise the costs by indicating if the resources are local and remote, the cost will be:

$$C_{total} = \sum_{i=1}^{n}(C_{r_cpu} + C_{r_stor} + C_{r_input} + C_{r_output} + C_{r_mem}) + (C_{l_cpu} + C_{l_input} + C_{l_output} + C_{l_stor} + C_{l_mem}) \quad (6.5.2)$$

In equation 6.5.2, an r in a cost C indicates the resource is remote on the cloud infrastructure, and l indicates the resource is local on the Cloudlet. For this equation, the remote storage cost will only be required for the VM image that can run whichever software is required to carry out the work offloaded by the Cloudlet; what form this takes is unclear. The local storage on the Cloudlet C_{l_stor} has to store the base virtual machines. As described earlier, applications and settings are stored on the overlay VM on the mobile device of the user. The local input and output costs C_{l_input} and C_{l_output} need to accommodate the energy required not only for transferring the visual VM GUI state to the mobile device and received input from the user, but also the costs to transfer the VM overlay to and from the mobile device. C_{l_stor} will account for any temporary storage of user data for the VM operation; no user data required for VM operation (except for temporarily offloaded jobs) will be stored at the remote server, so there is no contribution to C_{r_stor} for this. C_{l_cpu} will also have the overhead of the offload decision process.

If one considers remote viewing approaches, the cost model will be modelled as equation 6.5.1. For remote viewing, there is only a remote server cost. Each of n users will require their own virtual machine. The VM for each user and all user data will need to be stored on the server, so C_{stor} will be very high. Again, C_{input} and C_{output} can be expected to be large as to accommodate the energy required to transfer the visual VM GUI state to the mobile device and receive input from the user.

Code offload and application partitioning approaches may be more desirable. These need only have specific software running on the cloud resources to support their operations. However, it is generally not described what resources or specific software is required at the cloud. For example, for CloneCloud [19], a copy of the mobile device operating system must be available and running on the server. MAUI [21] has an MAUI runtime required at the server, which is presumably not as resource-hungry as an entire mobile operating system. How these are deployed (standalone executable, application containers, operating system services) is unknown. MAUI was implemented in C#, so it stands

to reason that the .NET CLR was used. For Alfredo [45], as an application partitioning solution, each local computing node must have OSGI available, as this is the framework that is used to split up and distribute the application components.

For application partitioning approaches such as Alfredo, the resource usage can be modelled as equation 6.5.1, except that n will be the number of remote computing nodes used for offloading the application partitions. Each node may or may not run virtual machines with the supporting OSGI platform (a desktop machine node could, but a mobile device node is unlikely to). C_{cpu} at a node will be smaller than in VM approaches because each node only runs a smaller part of a whole application on the platform, and if no VMs are deployed on the nodes, then each node only runs one operating system. C_{input} and C_{output} will correspond to the transfer cost of serialised application code and data. C_{stor} will not have any additional contribution from the mobile cloud approach. Cmem will be contributed to by the OSGI runtime.

Code offload approaches are similar to the model for application partitioning. n will be the number of cloud servers required as scaling takes place for the VM running the software or complete mobile operating system. C_{cpu} will be small, as the server only has to execute offloaded code, along with the required software for the offload process; there is little contribution to this cost from the mobile cloud. C_{input} and C_{output} will again correspond to the transfer cost of serialised application code and data. C_{stor} is tricky to estimate; if like MAUI, the server runs a software application to execute offloaded code, it would be small, but if the CloneCloud approach is used, this will be greater, because the entire mobile operating system is cloned into the cloud server; how many operating systems are required for each user and their software is unclear. C_{mem} will depend on the amount of code and data offloaded and will vary with each offload.

With each of the discussed approaches above, disconnection is a concern. If the mobile device becomes disconnected, then the exhausted energy at the cloud in completing work that has been offloaded will be wasted.

Similar to the code offload and application partitioning approaches, CAMCS only requires a runtime application container running in a virtual machine, along with NoSQL database storage for storing user and task details. The VM can be scaled as required to cope with demand, and one would expect it to be replicated at cloud deployments globally, for user proximity benefits. CAMCS

does not need to provision virtual machines and the corresponding required hardware for each user individual user.

As a result of the architecture of CAMCS, going by equation 6.5.1 again, n can be expected to be as low as one but will become larger as scaling occurs. One area where C_{cpu} may grow is in the case that CAMCS makes a decision process to carry out work on behalf of the user, without the user requesting it (which will also result in C_{input} being zero). C_{input} and C_{output} will be very small; as has been described, the nature of the character data will likely be even smaller than serialised application code. C_{stor} will be mainly attributed to the database storage. There is no significant storage required by CAMCS itself. C_{mem} will not incur any significant overhead from CAMCS. An important point, is that one cloud deployment can serve several users and their CPAs.

Aside from making independent decisions for the user, C_{cpu} will not result in any significant overhead from CAMCS per user, because the work of completing a task will be distributed to existing SOA services located elsewhere in the cloud and the web. This is an important consideration; as outlined in Chapter 5, various cloud and web services are already available in the cloud, and CAMCS relies on these to complete work for the user. As an example, if a user wishes for their CPA to find restaurants based on a meeting scheduled in a calendar and make a booking for them, services such as Google Calendar, TripAdvisor, and online booking systems such as booktable.com can be utilised (for this, they must expose a public API). These providers offer their services with their own server infrastructure, so VM resources allocated specifically for this computation do not need to be considered. The CPA relies on the nature of distributed SOA services, and does not operate under the same constraints of a VM that has to provide dedicated resources to carry out a piece of work. The CPA simply contacts services running on other servers, and stores result data, along with implementing some coordination logic. A CPA does not perform demanding computation itself that operates subject to available VM resources and constraints. In this sense, CAMCS acts as a trusted third party mediator in the cloud, to bring dispersed cloud services together, driving the concept of a mobile device as an intelligent, portable, lightweight-computing terminal.

6.6 Thin Client Modelling Experiments for the Mobile Cloud

In order to realise an integrated experience of the mobile cloud for the user, a model must be developed which considers the resources outlined in this chapter. A mobile user will primarily be concerned with the energy consumed on the mobile device, and time taken for mobile cloud communication. In light of this, several experiments were performed to evaluate task offloading over a cellular 3G network, to CAMCS deployed in the cloud.

The experimental devices used were a Samsung Galaxy S3, and a Google Nexus 5, running on the Vodafone Ireland network. The CAMCS Client was installed on each device. The Galaxy is the primary development device; the Nexus was introduced at a later stage, and the reasons for this will be explained shortly. Power saving mode was disabled on the Galaxy, the Nexus has no such mode. The research questions that were set to be answered with these experiments were the following:

1. What effect does the varying quality of the cellular signal (signal strength) have on task offloading time?
2. What effect does the distance between the mobile device and the mobile cloud application server location have on task offloading?
3. What effect does the varying quality of the cellular signal (signal strength) have on the power draw of the device while task offloading is in progress?

The results from the experiments performed to answer these questions, drove the creation of an offloading decision model for the CAMCS Client on the mobile device.

6.6.1 Signal Strength Experiments

Evaluating the effect of the signal strength on the performance of the task offload has been performed in various ways in related work. Code offloading approaches normally evaluate the state of the network before the offload decision is made. The result of this evaluation is given to an optimisation solver. The solver then makes the decision to offload the code to the cloud, if an objective

function can be met, such as saving energy. This network evaluation/profiling introduces overhead at the mobile device. The aim of this work is to eliminate this overhead to benefit the user experience. Nevertheless, some information on the network state must be known in advance for the offload.

The Galaxy S3 mobile device runs the Android operating system, as does the Google Nexus 5. Android provides some information through its PhoneStateListener API, regarding the state of the cellular network. According to the documentation, this includes signal strength (as an RSSI value or measured in Decibel-milliwatts [dBm]). By registering with the Listener, an application can receive updates whenever the strength of the signal changes. The CAMCS Client was adapted to listen for these updates. With this approach, no additional overhead is required to profile the network state, as this information is already collected by the Android OS. The first experiment was performed to answer research question 1, by evaluating offload performance with varying signal strengths.

In the experiments, the GSM signal strength was divided into ranges of dBm values: [-110, -100], [-99, -90], [-89, -80], and [-79, -70] dBm. The [-110, -100] range is the poorest quality signal range, while in the experiments, the [-79, -70] dBm is the best quality signal range. The signal range can go higher than this, though the mobile devices used, it was difficult to get a steady signal in the [-69, -60] range and above. The reason for using the GSM signal strength, rather than the CDMA signal strength, considering the use of the 3G network, is discussed in Subsection 6.7.1. As will be presented in Chapter 7, a file synchronisation application model of CAMCS was modified to evaluate the timing of the offload of image files of varying size to the CPA; the CPA then forwards these files to cloud storage providers such as Dropbox and Facebook. However, for these experiments, this forwarding was disabled; as soon as the file was offloaded to the CPA as part of the file synchronisation task, the server immediately responded without doing anything. Files are converted into a String format with Base-64 encoding for offload. This encoding adds overhead, therefore the images were sized beforehand such that when the overhead was taken into account and added to the offload data size, the size of the five image files used was 2MB, 1MB, 500KB, 250KB, and 125KB. For each of these files, 25 offloads were performed over the cellular 3G network, to a cloud server, for each of the 4 signal ranges, totalling 100 offloads for each of the five files. This experiment was repeated twice, once for a cloud server located at the Virginia, USA site of Amazon EC2, and one for a cloud server located within University

College Cork where the research took place. The aim of this was to answer research question 2. The Amazon EC2 server is a t1.micro instance, featuring 613MB RAM, and 1 vCPU with 2 EC2 Compute Units. The University server features a 1.7Ghz CPU and 2GB RAM. Offload experiments were performed between the hours of 10am and 5pm on weekdays.

The timing of the offload was captured using logging placed into the CAMCS Client code. This data was imported into IBM SPSS Statistics version 21, and boxplots were generated to study how the offload time varied for each of the file sizes, under the varying ranges of signals.

For the following results until stated otherwise, the Samsung Galaxy S3 device was used for the experiments. Figure 6.1 shows the boxplots for the offload time (in seconds) of the 2MB image to CAMCS deployed on the Amazon EC2 server (left boxplot) and the University College Cork (UCC) server (right boxplot), against the four GSM signal ranges. It can be noted from this that the median time for the [-99, -90], [-89, -80], and [-79, -70] dBm ranges are very similar, just under or at 10 seconds. The only notable difference is for the [-110, -100] range, where the median time for the Amazon server is just over 30 seconds, and just under 30 seconds for the UCC server.

This kind of result was also repeatedly observed for the offload times of each of the other file sizes (Figures 6.2 - 6.5). When the GSM signal strength is in the dBm ranges [-99, -90], [-89, -80], and [-79, 70], the median offload times are almost the same, but the median offload time for the [-110, -100] range is always higher. Of note, is that for the 500KB, 250KB, and 125KB files, there is barely any difference in the median offload times at all for the three strongest signal ranges. At this point, the offload time would appear to be dominated by network overhead, rather than payload transfer.

Figure 6.6 shows the same boxplot for the 500KB file offloaded to the UCC server, where the experiment was performed on the Google Nexus 5 device, instead of the Samsung Galaxy S3; this was to see if the same timing trend held true for other devices. It can be seen that the trend does hold. The median offload time was very close for the [-99, -90], [-89, -89], and [-79, -79] dBm signal ranges, 2-4 seconds, where the signal strength for the [-110, -100] dBm range was about 9 seconds.

From this, to answer to research question 1, it is concluded that if the GSM signal strength is greater than -100dBm, there is no significant difference in the

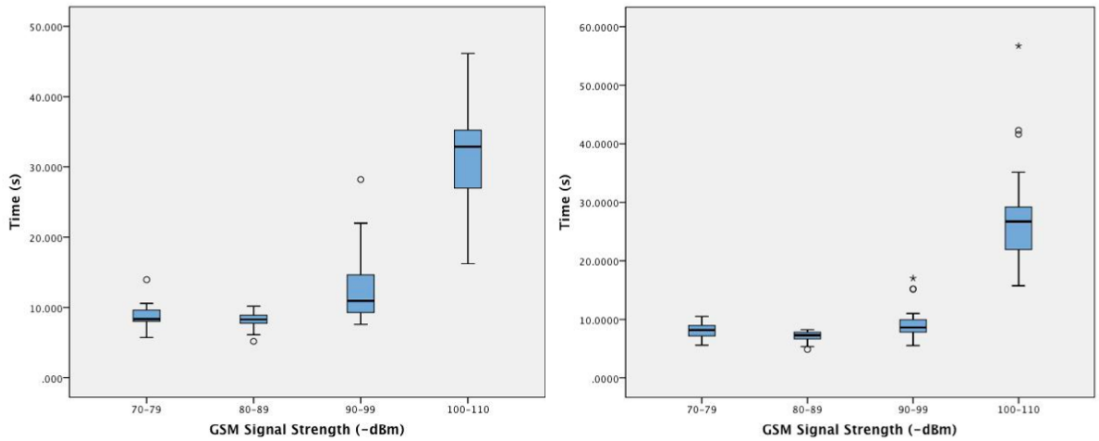


Figure 6.1: 2MB Image Offload Boxplots. Plot of offload time (in seconds) against GSM signal strength (in -dBm) for offloads to an Amazon EC2 VM (left) and a UCC cloud VM (right) over the cellular 3G network. For Fig. 3-8, 25 offloads were performed for each range of signal strength, totalling 100 offloads of the file. For Fig. 3-7, the Samsung Galaxy S3 device was used.

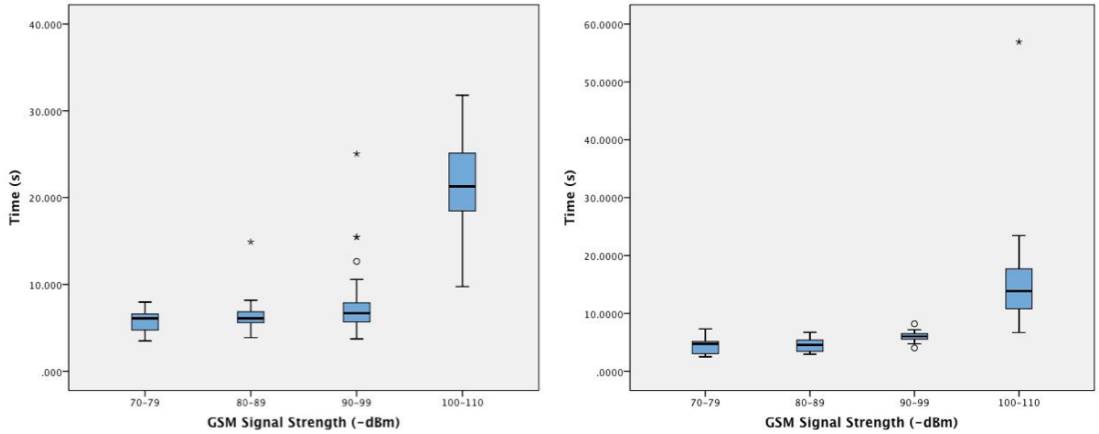


Figure 6.2: 1MB Image Offload Boxplots. Plot of offload time (in seconds) against GSM signal strength (in -dBm) for offloads to an Amazon EC2 VM (left) and a UCC cloud VM (right) over the cellular 3G network.

offload times as the signal strength varies.

6.6.2 Mobile Cloud Distance Experiments

To answer research question 2, further analysis was performed on the data collected to answer research question 1. Figure 6.7 shows a boxplot, comparing the offload time of a 2MB file to CAMCS, running on the Amazon EC2 deployment, and the University College Cork deployment in the [-99, -90] dBm signal

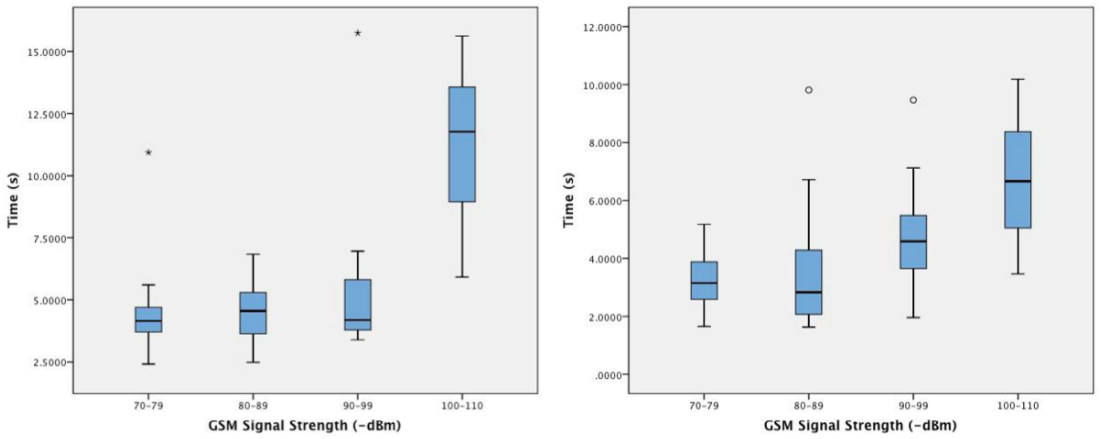


Figure 6.3: 500KB Image Offload Boxplots. Plot of offload time (in seconds) against GSM signal strength (in -dBm) for offloads to an Amazon EC2 VM (left) and a UCC cloud VM (right) over the cellular 3G network.

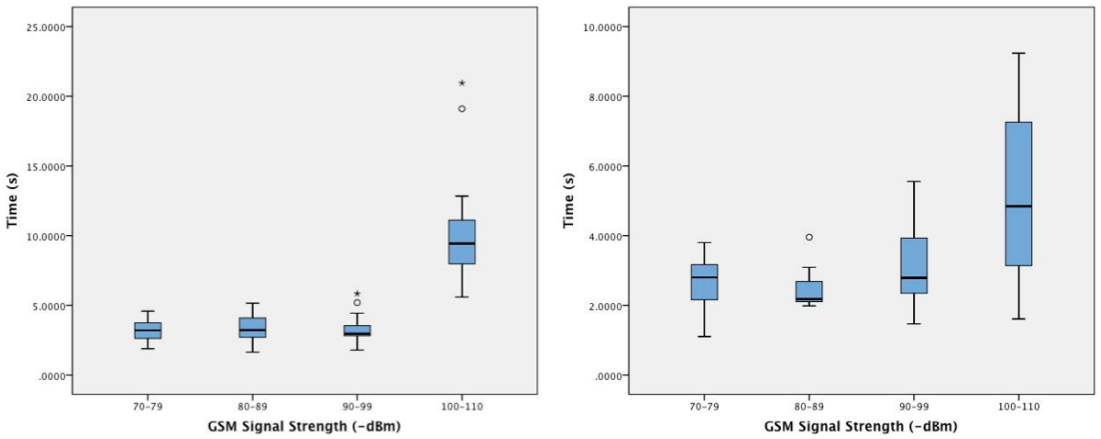


Figure 6.4: 250KB Image Offload Boxplots. Plot of offload time (in seconds) against GSM signal strength (in -dBm) for offloads to an Amazon EC2 VM (left) and a UCC cloud VM (right) over the cellular 3G network.

range. The mobile device used was the Samsung Galaxy S3, and it was located in Cork city when performing the offloads. The Amazon EC2 virtual machine is running in the Virginia, USA Amazon datacentre.

Looking at Figure 6.7, one can see that there is little difference between the median offload times for each server. The standard deviation is higher for the Amazon server; this may be expected because it is further away than the UCC server and naturally because of varying bandwidth and usage over the traversed networks. This experiment was repeated for all the file sizes, and there was little difference for all, so the boxplots for the other files are omitted. One other box plot is provided for reference, Figure 6.8, which shows the

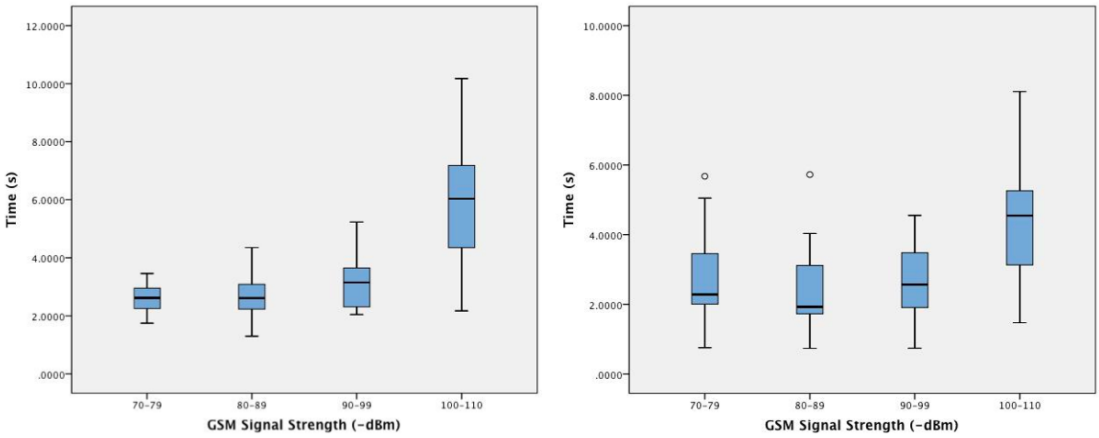


Figure 6.5: 125KB Image Offload Boxplots. Plot of offload time (in seconds) against GSM signal strength (in -dBm) for offloads to an Amazon EC2 VM (left) and a UCC cloud VM (right) over the cellular 3G network.

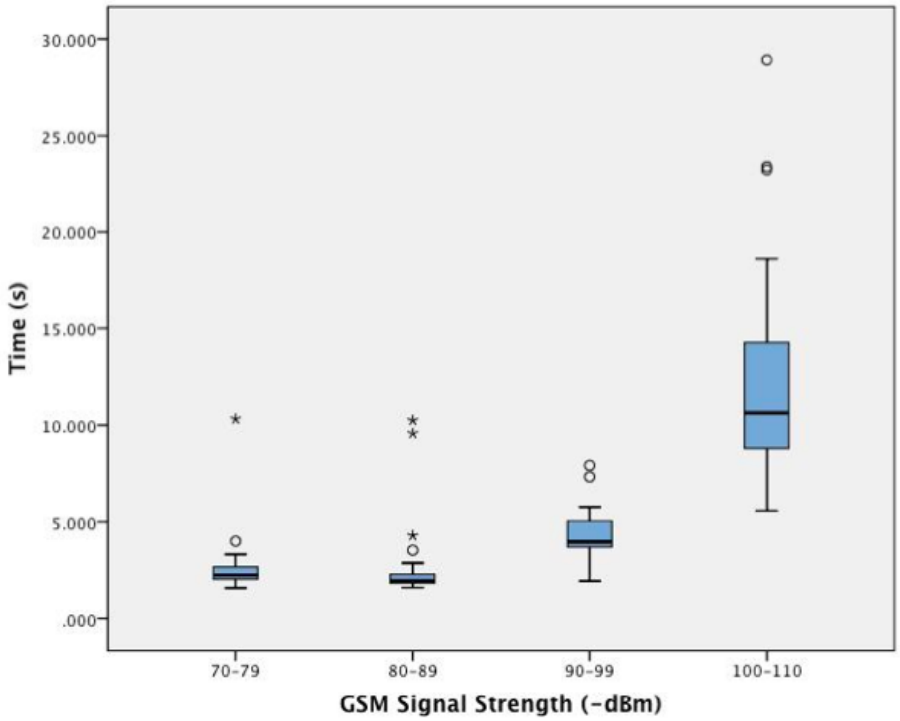


Figure 6.6: 500KB Image Nexus 5 Offload Boxplots. Plot of offload time (in seconds) against GSM signal strength (in -dBm) for offloads to a UCC cloud VM over the cellular 3G network, with the Google Nexus 5 device.

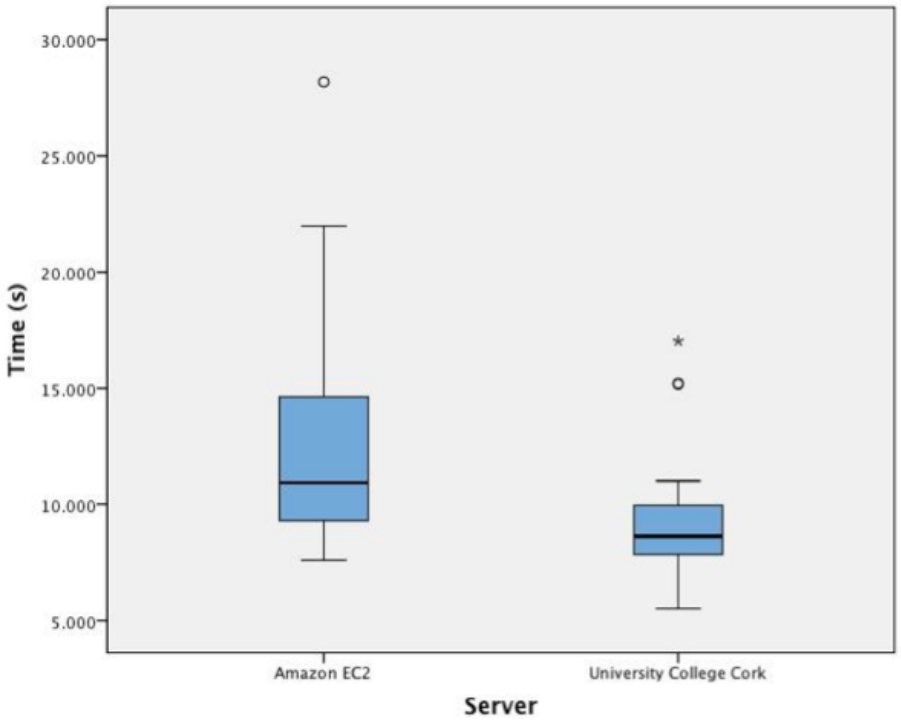


Figure 6.7: 2MB Image Offload Server Comparison Boxplots. Plot of offload time (in seconds) for offloads to both an Amazon EC2 VM and a UCC Cloud VM, in the GSM signal range of [-99, -90] dBm, over the cellular 3G network. Data taken from 25 offloads of the file for both servers in the [-99, -90 dBM range of Figure 6.1

same experiment for the 250KB file. In this case, the results for the closer UCC server show higher standard deviation and larger interquartile range than the Amazon server results, although the Amazon server data features outliers not found in the UCC server data. From this data, it is concluded that when modelling the offload performance, there is no need to consider the server distance, from a time perspective. Note that this is just the result from CAMCS and the CAMCS Client application. Consider a real-time application, such as in experiments performed in the work by Clinch et al [20], where latency caused by server distance did have an impact on results on the user experience while playing games run on Cloudlets.

6.6.3 Power Experiments

Answering research question 3 was a more difficult endeavour. For the Android OS, there is no publicly available power or energy API. One can use a graph in the device Settings to determine how much battery energy has been

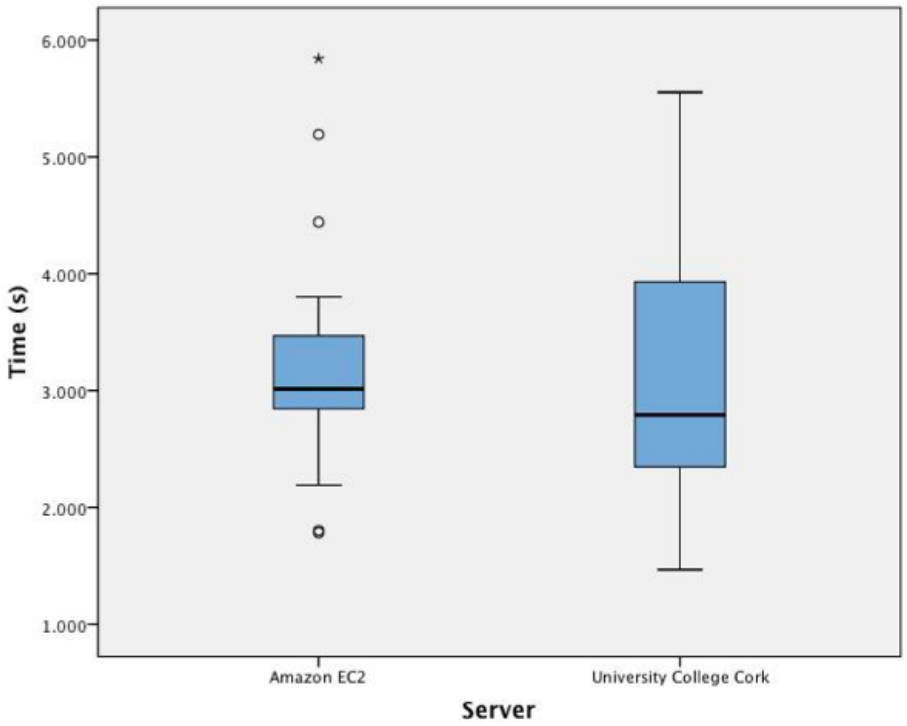


Figure 6.8: 250KB Image Offload Server Comparison Boxplots. Plot of offload time (in seconds) for offloads to both an Amazon EC2 VM and a UCC Cloud VM, in the GSM signal range of [-99, -90] dBm, over the cellular 3G network. Data taken from 25 offloads of the file for both servers in the [-99, -90] dBM range of Figure 6.4

used by the different OS components and applications in percentages, but this is not fine grained, the figures are only estimates. In the MAUI code offload paper [21], the authors used external measuring equipment placed between the battery and the device to construct a model. The implications of this for a developer are that they cannot dynamically evaluate power consumption of an application they develop.

For this research, the Trepn profiler tool [106], developed by Qualcomm, was used. Qualcomm manufacture the processors used in various mobile devices and the tool can be used to profile various aspects of the mobile device performance. A profiler application is simply installed on the mobile device, and is started. The user can then carry out whatever tasks they want to profile the performance of. The profiler is then stopped, plugged into the Eclipse IDE, and a graph printout shows the collected data. Unfortunately, the primary Samsung Galaxy S3 development device, does not profile battery power usage. This is why at this late stage the Google Nexus 5 device was introduced, which is fully compatible with the profiler. Therefore, the following experi-

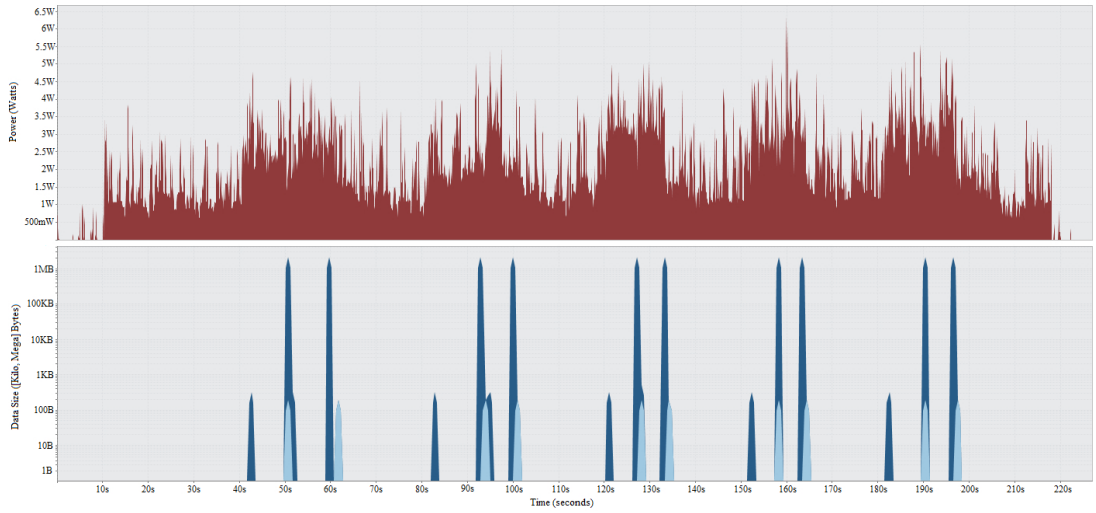


Figure 6.9: 2MB Image Offload Trepn Profiler Graphs from (-89, -80) dBm GSM Signal Range. Top graph shows battery power usage (Watts), bottom graph shows mobile data usage (Bytes), both over time (Seconds), during offload to a UCC cloud VM over the cellular data connection with the Google Nexus 5. Five offloads occurred here. Offloads occur in the periods between the tall blue spikes in the bottom graph. Therefore, spikes appear in five pairs; one pair for each offload. The first spike in a pair is the time at the start of the data transfer, and the second spike is the time at the end of the data transfer. The smaller spikes in the bottom graph result from DNS queries and handshaking. Dark blue represents data sent, light blue represents data received.

ments were carried out with the Google Nexus 5 device. For this experiment, the profiler was started. Offloads of the 2MB file were performed over the [-110, -100], [-99, -90], and [-89, -80] dBm GSM signal ranges. The results for the [-89, -80] and [-110, -100] dBm ranges (in the aforementioned order) are featured in Figures 6.9 - 6.10. For each experiment, the profiler measured the battery power before, during, and after the offloads, in Watts. For reference, Figure 6.11 shows a power draw trace gathered when no offload was taking place. Aside from occasional spikes, the power consumption remains levelled out between 1W and 1.5W for the trace with no offload taking place. This base power draw could of course vary depending on what other applications and services are running on the device at the time.

The interesting results show that there was little difference for each of the signal ranges tested. When the offloads start, there is an initial ramp up of energy used, before the offload starts. Once the offload starts, the power for each of the offloads performed in all signal ranges shows many peaks of between 5W and 7W, but all graphs level out between 3W and 3.3W. After the offload com-

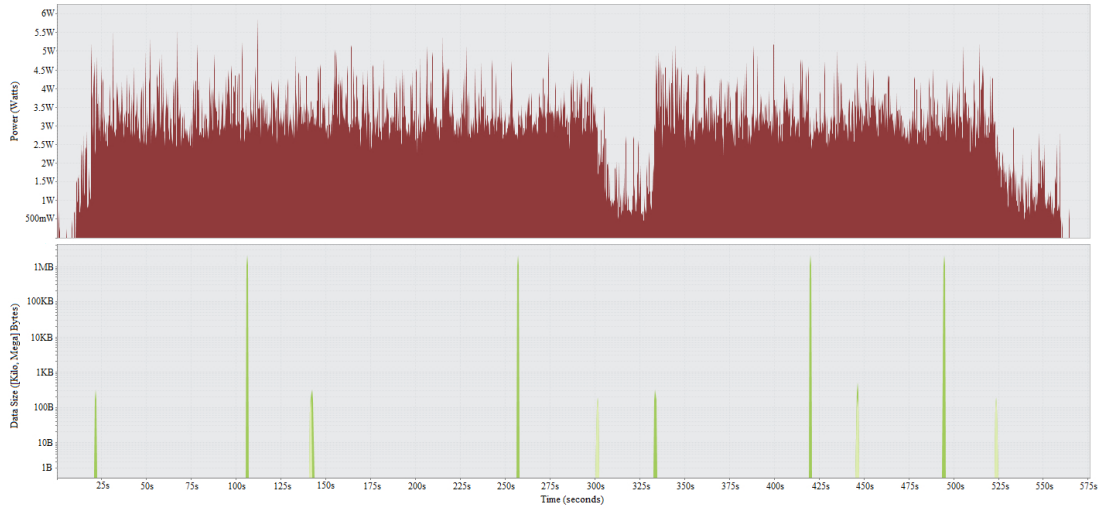


Figure 6.10: 2MB Image Offload Trepn Profiler Graphs from (-110, -100) dBm GSM Signal Range. Top graph shows battery power usage (Watts), bottom graph shows mobile data usage (Bytes), both over time (Seconds), during offload to a UCC cloud VM over the cellular data connection with the Google Nexus 5. Two offloads occurred here. Offloads occur in the periods between the tall green spikes in the bottom graph. Therefore, spikes appear in two pairs; one pair for each offload. The first spike in a pair is the time at the start of the data transfer, and the second spike is the time at the end of the data transfer. The smaller spikes in the bottom graph results from DNS queries and handshaking. Dark green represents data sent, light green represents data received.

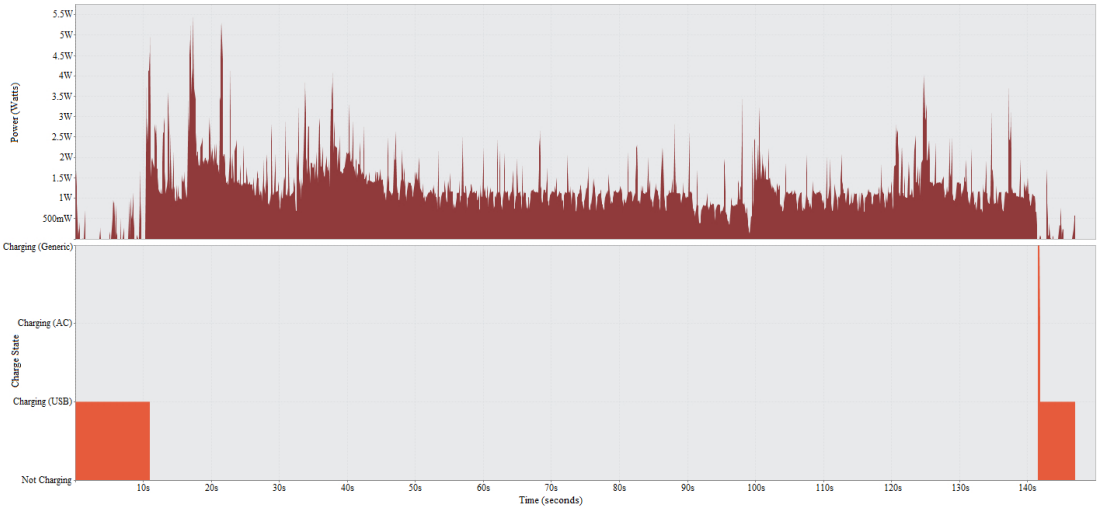


Figure 6.11: No Offload Activity Trepn Profiler Graphs. Shows battery power usage (top) and charge state (bottom) while no offload was occurring with the Google Nexus 5. Aside from the occasional spikes, power usage rests between 1W and 1.5W. It can also be seen that during the time the device is plugged into a USB charger (time periods when the yellow rectangular areas on the bottom graph are visible on the left and right), power usage drops further.

pletes, the power does not drop for a few seconds; this is the tail energy as demonstrated in the related works. The same trend was found in the [-99, -90] dBm GSM signal range, and therefore the graph is not shown.

It was anticipated that the power used would be greater as the signal strength decreased, especially for the [-110, -100] dBm range, as with the results from Subsection 6.6.1. This however does not seem to be the case. It should go without saying that because of the longer offload duration of the poorer signal strength, especially the [-110, -100] dBm range, that the transfers take more time, and as such, the power draw from the battery will occur for a longer duration of time.

In terms of energy usage, due to lack of hardware, the energy usage cannot be measured directly, however it can be estimated based on the power usage observed, and the time spent offloading. An exact value for energy usage also cannot be estimated; as shown in Figures 6.9 - 6.10, with varying power spikes during offloads, a constant value would not be appropriate. Instead, a lower-bound estimate on the offload energy is used:

$$\Omega(e) = 3W(t) \tag{6.6.1}$$

This formula is simply derived from the formula for Energy, $E = PT$, where E = energy, P = power (or work done), and T = time. In equation 6.6.1, e = energy in Joules, and t = time. 3 is derived from the minimum of 3W of power observed from Figures 6.9 - 6.10 during the offloads.

Deriving an upper-bound estimate on the energy is less practical. In the experiments performed, the highest power usage spike observed during an offload was around 7W. Defining an upper bound from 7W would by and large grossly over-estimate the energy used. Instead, a weaker upper-bound estimate on energy is defined:

$$\theta(e) = 3.3W(t) \tag{6.6.2}$$

In Figures 6.9 - 6.10, ignoring the high spikes, the maximum offload energy used commonly levels off around 3.3W, and so equation 6.6.2 is defined as

such. The derivation and terms used are the same as in equation 6.6.1.

From these results, it would appear that in terms of power draw, there is no need to include any specific power draw aspect into the model, as it does not vary considerably with stronger and weaker signal strengths. To lessen the energy utilised for offload, the speed of the offload must be maximised to reduce the time taken. From the results in Subsection 6.6.1, to achieve high speed, it is crucial to offload when the signal strength is greater than -100dBm. This result is an interesting contrast to the findings for the Bartendr work by Schulman et al [120] which found a higher power draw for lower signal strengths as described in Subsection 6.2.1. This may be caused by the older devices used in the evaluation, which were not modern smartphones, or the properties of the EVDO networks used for evaluation in the USA and India.

6.7 Modelling for Thin Client Offloading

Based on the experiments performed, and the obtained result data, a model for the mobile cloud offload decision is derived, which takes into account the resources under discussion. Subection 6.7.1 provides some practical analysis of implementing such a model. Subsection 6.7.2 will present the decision model. Subsection 6.7.3 briefly describes the implementation of the mechanism for the CAMCS Client.

6.7.1 Analysis

Considering the related works presented which provide models for energy usage at the radio level, it is difficult to incorporate these at the application level, as anybody developing a mobile cloud application may not have control over data size, radio state, or distance between the mobile device and the cell tower. When bandwidth is taken into account, regardless of bandwidth utilisation on the network, the user will want their tasks and data to be transferred regardless of the bandwidth utilisation of the network at some point. In the related works presented, the network state profiling and optimisation decision on local versus remote execution has been described. The aim of this model is to remove this overhead, which can have a detrimental impact on the user experience, and waste resources.

Table 6.1: Offload Decision Model Variables

Variable	Description
$p \in (0, 1)$	Offload Priority
s	Offload Data Size in Kilobytes
$r \in (-51, ..., -120)$	Signal Strength in dBm
$b \in (0, .., 1)$	Battery Percentage
$o \in (0, 1)$	Offload Decision Variable

The major obstacle in implementing such presented models at the application level is the limited options available to the developer. The Android API does not feature any functionality that allows the developer to inspect the power or energy state, or determine the state of the network quality without sending packets out over the network connection to evaluate RTT and bit-error rates.

Another area of difficulty with the Android API is in the details provided by the PhoneStateListener. A knowledgeable reader may question why the signal range experiments for a 3G network transfer were based upon on the GSM signal strength; 3G networks use UMTS with CDMA instead of GSM. Both the GSM and CDMA signal strength can be read from the PhoneStateListener in Android. However, implementing this functionality is dependent on the NIC manufacturer. On the Samsung Galaxy S3 device, if a developer queries the CDMA signal strength, while connected to a 3G network, the method call will return -1. This indicates it cannot read the CDMA signal strength. -1 is also returned if the developer queries the GSM bit-error rate, another method call in the Android API. The only data the Samsung Galaxy S3 device provided was the GSM signal strength. As such this was the only option to try and gauge signal quality, without adding additional network overhead. For a developer, the GSM signal strength measured in RSSI or dBm, is the only measurement that can be relied upon, at least in Europe at the time of writing. Other mobile devices were checked, and they were only able to retrieve GSM signal strength from the Android API, as with the Samsung Galaxy device.

6.7.2 Modelling the Offload Decision

Based on the experimental results in Section 6.6, and the aim of not adding any additional overhead to the mobile device, a decision model for offloading over the cellular network connection is introduced. This model uses the variables outlined in Table 6.1.

The offload decision model is then defined as:

$$o = \begin{cases} 1 & p = 1 \\ 1 & decide(r, s, b) = true \\ 0 & decide(r, s, b) = false \end{cases} \tag{6.7.1}$$

The three case stack in 6.7.1 has 3 possible outcomes. A priority variable p. If this is set to 1, then regardless of the signal strength r or battery percentage b, offload will take place immediately; hence o is set to 1. The user can specify if a task is high priority using the CAMCS Client. The second and third cases rely on a decide function, which encapsulates a set of rules to determine if offload should take place. If the device is connected to a Wi-Fi network, offload will always take place immediately, as the network is considered to be of superior quality when compared with the cellular network. However, the decision process can easily be applied to this connection as well, in terms of the battery power, or even extended further to take into account performance based on Wi-Fi signal strength. The decide function takes three parameters, r, s, and b. The algorithm pseudo code is presented in Algorithm 1.

The comments in Algorithm 1 describe how the algorithm decides if the offload should occur. Any developer, or even the user, may choose their own battery policies for offload decisions; this is just one implementation. One may even leave the battery percentage values in the cases to the user to decide as a preference on the CAMCS Client. The important decision processes in this algorithm are the cases; offload does not occur if the signal strength is weaker than -100dBm. Otherwise, the size of the data offload and the current battery situation are both considered. For cases where the data offload size is greater than 500KB, in the results, the offload time would vary depending on the size. For cases where the data offload size is less than or equal to 500KB, there was little difference in the offload times. As described in Subsection 6.6.1, for sizes below 500KB the time is mostly dominated by network overhead, rather than payload size. Where the battery drops to 10% or lower, the user more than likely wants to conserve the battery life of the device for important calls or messages, rather than having additional network activity occur.

It is worthwhile briefly mentioning a situation where the signal strength changes during the offload. If the signal strength weakens to a value less than -

Input: GSM signal strength r, data size s (in Kilobytes), battery percentage b
Output: true if offload should occur, false otherwise
if $-100 < r$ **then** ; // If signal is weaker than -100dBm

return false ; // Do not offload
else
switch s **do**
case $s > 500$; // Case: data size greater than 500KB

if $b > 0.25$ **then** ; // If battery percentage is greater than 25%

return true ; // Offload
else
return false ; // Do not offload
case $s <= 500$; // Case: data size less than or equal to 500KB

if $b >= 0.10$ **then** ; // If battery percentage is greater than 10%

return true ; // Offload
else
return false ; // Do not offload
endsw
end

Algorithm 1: Algorithm Decide

100dBm, perhaps the offload should be suspended. This is not implemented in this model. If an offload has started, then regardless of how the signal strength changes, it should continue until completion. In the related works, disconnection during offload is not discussed, except for in the MAUI framework [21]. With MAUI, if the mobile device suffers a disconnection from the cloud while code has been offloaded, after a certain timeout period, the local application will execute the offloaded code itself. This is a waste of resources in the case where offload has successfully taken place, but no result has been received. The preference taken was that starting an offload in this model, and using up time and energy as a result, should not be made to count for nothing if the signal strength deteriorates.

This algorithm must be seen as a trade-off between accuracy and the zero-overhead goal. The related works perform more extensive network profiling as has been discussed, resulting in a more accurate picture of the network con-

ditions. This is at the cost of the overhead associated with the extensive profiling. The model used here is less accurate, since no additional network profiling takes place, and because the model is based on the GSM network, rather than the CDMA network. As the GSM measure does not take into account network traffic levels, or other measures such as the signal-to-noise ratio, the model is a guideline only, and does not guarantee optimal network quality at any given time, even when the GSM signal strength is greater than -100dBm. Essentially, it is used as a prediction of the network performance. However, using this model as a guideline for offloading, does meet the user experience aim of imposing no additional overhead. Hence, it is used within the CAMCS Client as the basis for determining if queued tasks should be offloaded, or remain in the queue until a later time.

6.7.3 Model Implementation

The CAMCS Client was modified to implement the model in the previous subsection. Whenever the user enters the details of a task to be offloaded to their CPA, for example, a file to be offloaded to a cloud storage provider as will be described in Chapter 7, the task is forwarded to a TaskOffloadHandler class. This class features a custom implementation of a Queue data structure, the TaskOffloadQueue. Whenever a task is passed to the TaskOffloadHandler, unless it has been assigned priority, the task is placed into this queue, along with some additional offload data, such as the data-size. If the user has assigned priority to a task, it is offloaded immediately. The task is converted into JSON format for the transfer. Files, such as the image files in this example, are Base-64 encoded to a String for offload. Based on the number of ASCII characters in the JSON String and any Base-64 encoded files, the data size is calculated.

At this point, if the queue was previously empty, the Handler will register with the Android PhoneStateListener, to receive updates when the signal strength changes. When a signal strength change is detected (which is received as an RSSI value, which can be mapped to a dBm value), the queue is iterated, and for each task in the queue, the decide algorithm is executed. Based on the outcome of this, the task is offloaded, or the task remains in the queue. When the queue has finally been emptied of tasks, the Handler will unsubscribe from the PhoneStateListener. The user is notified via the Android notification tray when a task is being offloaded, and when the offload is complete.

6.8 Discussion

To enable a mobile cloud computing model that will work seamlessly for the user, the resources described in this chapter must be managed cost-effectively. For the end-user of the mobile device, the effectiveness of how energy and bandwidth are managed, will determine how successful the implementation of the model will be. Users want batteries that last longer, and have limited patience for delay from the network connection; only if these resources are used wisely will users uptake this model. These resources are in limited supply to begin with. The required elements for the three resources analysed in this chapter are now highlighted to define the elements of a best practice model for management.

For energy, many of the previous works that have attempted to model the energy usage of the mobile device have brought the energy usage down to three factors: power for the radio, the amount of data transferred, and the distance from the base-station. To enable a successful implementation of the mobile cloud model, these factors must be minimised. Power is supplied to the radio cannot be controlled, but the amount of data that is transferred over the connection can be controlled and minimised, and the location of cloud deployments worldwide can be planned so that they are close to the user, minimising network delay where possible (a user cannot be forced to move close to a base-station, nor can a user be expected to stay there). The other factor in this regard is how often the network connection is used, as this also has a detrimental impact on the energy usage; it must also be kept to a minimum. Only when all these factors are minimised, can a mobile cloud model be realised that effectively manages energy for the user.

For bandwidth, the amount allocated to the mobile device cannot be guaranteed or controlled, either on the fly, or in advance. Only approaches that are going to have minimal bandwidth requirements in this regard will be successful and tolerated by the end user. Systems must be developed under the assumption that only a minimal amount of bandwidth is available, and the mobile device must adapt its use of the mobile cloud appropriately to situations where a large amount is available, or where a small amount is available. If a large amount of bandwidth is available, such as on a Wi-Fi network, it should be used for the greatest advantage to the user, but only by assuming that little bandwidth is available, will all situations be tolerated by the user. It

is the same issue that applies to energy consumption; the amount of data that is transferred over the network must be minimal, and data transfer should occur sparsely. While approaches may take advantage of high bandwidth situations such as on Wi-Fi networks, and transfer large amounts of data at these times, the issue of fairness for other users connected to access points will be of concern as well.

While the end-user will likely not be concerned with what is happening on the cloud deployments at data centres, how the resources will be provisioned there, and what those resources are will be of technical importance to the implementation of the mobile cloud. Not only should the footprint on hardware requirements be minimal on the cloud servers, but also the software requirements must not be extensive either. As more consumers of mobile cloud come online, the requirements at the cloud, resulting from factors such as personalisation of applications and data, must not grow. Providing a VM to each user for example may grow at an un-manageable rate. Replicating custom and personal software for users in the cloud (in the sense that users currently have personalised apps on their phones) may be difficult to scale, considering the size and complexity of the software. To achieve effective resource utilisation, the factors outlined in equation 6.5.1 must be minimised. It was reasoned that code offloading and CAMCS do this most effectively. In addition, for CAMCS, the use of existing software and services already located in a distributed SOA fashion in various cloud deployments, that can be shared amongst many end-users, will be the appropriate deployment model in the cloud moving forward. This is especially useful, as the VM is not relied upon to carry out intense computation and subject to the resource limitations and performance demand penalties that may come with a VM.

The model for offload decision making over the cellular network in equation 6.7.1, in contrast to other works, does not impose additional profiling overhead on the device, further adding to the user experience, such as no energy or time penalties. It was observed from the experimental results that offloading time increases significantly when the GSM signal strength falls below -100dBm; above this, offload times are very similar. It was also observed that when the size of the data offloaded falls below 500KB, the median offload times are almost the same. In terms of the cloud deployment location, the results show that the actual deployment location relative to the user location had little difference in median times, but for the further away Amazon EC2 server, there was more variance. Battery power usage, aside from power spikes, was found

to be the same regardless of the signal strength during offload.

In terms of the best practices outlined in this chapter, the results from the experiments presented, now incorporated into the offload decision model, show that in order to manage energy usage effectively at the mobile device, the time for offload is the crucial factor. Minimising offload time is the goal to meet, and the best way to achieve this is to offload when the GSM signal strength is greater than -100dBm. The results also show the value of the low bandwidth approach. The smaller the data size, it goes without saying, the better performance; but for approaches that rely on transferring large volumes of data, practical application may be difficult, considering the substantial time overhead in comparison with small amounts of data.

For example, consider the previously discussed remote display approaches. Unless compressed, transferring an image of a high resolution desktop output continuously over the network, judging by the experimental results, will result in a very poor user experience, and will use available resources very inefficiently. With small amounts of data, the network overhead is the primary time contribution, and so real time applications must focus on data size and frequency of use optimisations, by making sensible use of data transferred when required. Of course, the fact that this transfer may have to take place continuously will also have a detrimental impact on the energy resources and performance. CAMCS does not require continuous data transfer, just the offload, and a result. The size of task descriptions sent to the CPA during the experiments to be presented in Chapter 8 are less than 5KB. Offloading such a small amount of data, while the device has a strong signal, will reduce offload time, and save energy.

CAMCS and the CAMCS Client can effectively manage all these resources to provide an effective model to satisfy the consumer of mobile cloud applications. If CAMCS is contrasted with the other approaches, and their implementation details that were outlined in this chapter, required data transfer for CAMCS is minimal. Continuous connections to cloud infrastructure are not required once tasks have been offloaded to the user's CPAs. In addition, the CPA is capable of carrying out work and delivering results to users without the user having to request it. Results are stored with the CPA until the mobile device is available, and so it will support disconnected operation. The decision model for offloading over the cellular network on the CAMCS Client considers the resources available, with no additional overhead. A development goal is

Table 6.2: Mobile Cloud Computing Approach Resources. Comparison of the resource requirements of each of the discussed approaches to mobile cloud computing models, along with CAMCS

	Cloudlets	**Remote Display**	**Code Offloading**	**Disconnected Operation (CPA)**
Continuous Connection	Yes	Yes	Yes	No
Energy	High due to virtual machine nature	High due to virtual machine nature	Varies depending on amount of data offloaded and frequency	Low as data transfer and frequency is small
Bandwidth	Moderate, 1-Wi-Fi hop	High, many hops	Varies depending on amount of data offloaded and frequency	Low as data transfer and frequency is small
Cloud Services	Hypervisor for user virtual machines	Hypervisor for user virtual machines	Varies (entire operating system, custom application)	Middleware application using existing software/ services
Decision Model	None, user initiated	None, user initiated	Varies, user initiated or network profiler with optimisation solver	No overhead model using existing information

that CAMCS will de deployed on multiple clouds, so that it is always close to the position of the user, and the CPA will move between these deployments; even if this showed little impact in the results, the shorter network traversals should beep the time variance low. The server side requirements only require an appropriate application container and database, and it does not require any physical resources to be allocated to each end user. Table 6.2 compares the discussed approaches with CAMCS, along the lines of how they manage the available resources discussed in this chapter.

6.9 Conclusions

In this chapter, an analysis has been performed of how mobile device resources such as energy and bandwidth, along with cloud infrastructure resources, can be managed effectively in the mobile cloud domain, and best practice approaches were modelled and presented for mobile cloud computing solutions. These were applied to existing works in the area, along with CAMCS.

A user's CPA within CAMCS works in the cloud to complete user assigned tasks, using existing cloud software and SOA services. Most importantly, it works in a disconnected fashion, which has several advantages in terms of resource management, compared with other existing approaches in the area. The network connection is not in continuous use, or required to be active as in other works. It is infrequently used, simply to send the task to the CPA from the CAMCS Client, and receive the result later. The amount of data sent over the connection, the task description, and the result, is character-based and small in size. As a result of the small data size, and infrequent use of the network connection, the approach minimises energy and bandwidth use, when compared with some of the existing approaches described in this chapter. The server side footprint at the cloud is also smaller compared to other approaches, as multiple cloud based hardware and software resources are not required for each individual user of CAMCS. By comparison, CAMCS runs within an application container, which can be deployed in multiple clouds, so that a user's CPA is nearby, minimising communication time variance.

Experiments were conducted with mobile devices to judge the performance of offloading to the cloud from the CAMCS Client, along with their results. The effects of signal strength, cloud deployment location, and power usage during offload were studied, while offloading with various different data sizes, and ranges of GSM signal strength. From these results, conclusions were drawn which were implemented into an offload decision model for the cellular network, which imposes no additional overhead on the mobile device. This model can be used as a guideline for when the cellular network conditions may be optimal for sending tasks from the CAMCS Client, to the CAMCS middleware in the cloud. The conclusions drawn from these results were discussed in the context of the CAMCS approach, and the implications for other approaches.

Applying such management to the resources is important, because each user of the mobile cloud will have many tasks to be completed, and these resources

will be required separately for each task; these resources must be used with the considerations outlined in this chapter in mind. Effectively minimising the usage of these resources, and applying the best practices that have been outlined, will be crucial in successful adoption of the mobile cloud computing model by the end user.

In the next chapter, additional application models that can be provided by CAMCS, proposed in Chapter 3, will be presented along with their implementation, and development challenges.

This chapter was based on [103]. Various modifications have been made to the text throughout the chapter, to fit with the flow of the thesis.

Chapter 7

Mobile Cloud Application Models Facilitated by the Cloud Personal Assistant

7.1 Introduction

In Chapter 3, the design of CAMCS was shown to provide support for pluggable modules. This can provide support for other types of applications to be supported by CAMCS, and some example applications and use-cases were provided. In this chapter, some of those applications and use-cases that can be facilitated by CAMCS and the CPA, are examined.

Various applications synchronise user files across each mobile device owned so that they are accessible from each of them. Changes to a file on one device can be reflected on all other devices. Dropbox [29] is one example of this model: cloud storage is provided to store the user's files, and each mobile device can retrieve the files from Dropbox using an application installed on the device. Similar services include Google Drive [28] and Microsoft SkyDrive[128]. Another example is Apple iCloud [58], which pushes content purchased from the iTunes store onto each of the user's "iDevices'", or, additionally, the user can play media files from the cloud, without storing them on the mobile device. Many users also have social networking accounts such as Facebook [35] and Twitter [139], and upload media files to these services as a form of cloud storage. One resulting benefit is that the limited storage space on the mobile device is saved. However, all of these services work in isolation. If a user has accounts

with several of these providers, all files must be synchronised and maintained separately. The user must upload files from the mobile device to each service individually using different applications, which costs time, money, and energy.

As has been seen in Chapter 2, mobile cloud computing can also be seen as a platform for demanding computations (such as Cloudlets and code offloading). Execution of computationally long-running and resource-intensive operations, such as large dataset processing and mathematical calculations, can also execute on the cloud, with results returned to the mobile device. As a result, applications do not utilise excess battery power and processing capacity of the mobile device. The application is also not at risk of interruptions, such as being killed in low-memory scenarios or accidental shutdowns.

In many enterprises such as in the office environment of a business, academic institution, or hospital, collaboration of members of different departments or teams is crucial in order to complete work made up of several different tasks. Due to time constraints of daily schedules, or in cases where team members are located in various locations worldwide, it can be difficult to come together for meetings to assign/oversee work. However, all users have their mobile devices with them to communicate back to the team. There are web-based and mobile applications available for task or project management, but these often involve users having to find time to manage and set their own work and task progress within these solutions.

In this chapter, these use-cases have been implemented as additional application models of CAMCS. This work shows how to use a CPA to enhance two of the application models described - synchronisation of files and data intensive processing. Also examined in this chapter is how the CPA can be used to complete group-based tasks. For each application model, appropriate implementation challenges, and lessons learned, are discussed.

7.2 Application Models

The three application models implemented for this work are now presented, which have been implemented as new features of CAMCS with the CPA, along with their advantages compared with existing solutions.

7.2.1 File Storage/Synchronisation

The first application model added to the CPA allows the user to send files from their mobile device to different cloud service providers. The widespread use of mobile devices as enablers for users to store and share files/content on cloud-based and social networks was the primary motivation for this feature. On the mobile thin client, the user can add their details (username/password) for different service provider accounts. File synchronisation involves several steps:

1. Service Provider Authorisation for CAMCS

   This involves selecting from a list of supported service providers, and authenticating with the selected provider, to give CAMCS access to the account.

2. Service OAuth Key Storage with CAMCS

   Once the user has authenticated on the mobile device, the authentication keys used to access the accounts on the users behalf are sent from the thin client to CAMCS in the cloud.

3. File Selection for Upload

   A user can upload a file from the mobile device by using the Android share feature, where the CAMCS Client is listed as a share option. This allows the user to select which of the provider accounts they have added to CAMCS that they intend to upload the files to. The user can select file storage providers such as Dropbox or Google Drive for any type of file. If at least one of the selected files is an image or video, it will also provide the option to upload to Facebook - Facebook only supports upload of image and video files.

4. File Upload to CPA and Service Providers

   After the user has selected the accounts to upload to, the files are sent to CAMCS in the cloud, using a RESTful web service. Once CAMCS receives the files, they are passed to the user's CPA, which will then send the files to the selected accounts - see Figure 7.1.

The advantage of such a feature is that if, for example, the user, possibly a company representative, wants to upload files such as promotional material to multiple social networks such as Facebook and Google+ to reach all possible

consumers, they no longer have to spend resources such as time, money, and energy on their mobile devices, uploading to each service provider one-by-one. Previous solutions in this regard upload files to each provider individually from the mobile device, using up the described resources during the upload to each provider. Taking advantage of this feature offered by CAMCS, users only have to upload the file once to the CPA, which takes care of sending the files to the different providers; the resources are only used once for a single upload operation. If the user has client software for the providers on their desktop PC's, laptops, or mobile devices, the files will be synchronised to them via a push operation. As a result, the current implementation does not feature a download synchronisation to the mobile device as files may be duplicated, wasting more resources. Evaluation, along with the implementation for this application will be discussed in Section 7.3.

Research in the area of mobile cloud for storage purposes mainly examines methods and protocols for secure and privacy-preserving data and file storage in the cloud. Awad et al [5] proposed an encryption scheme for storing files in the cloud from mobile devices; this scheme also permits for a confidential fuzzy-based keyword search of stored content by the user. In work by Zhou and Huang [150]. what is known as a Privacy Preserving Cipher Policy Attribute-Based Encryption method is developed to protect data gathered from sensing devices within smartphones. As part of their solution, they also develop an Attribute Based Data Storage system, as an access control mechanism to the sensing data stored in the cloud. Their solution has a focus on moving the complexity of encryption and decryption operations from the mobile device and into the cloud, for energy efficiency. Ren et al [111] developed several schemes for providing secure storage of files from mobile devices on cloud servers, where there may exist one or more distributed cloud servers, that may or may not be trusted. For the CAMCS implementation of this model, the existing security mechanisms of the third party cloud storage providers are relied upon.

7.2.2 Data Processing

One fundamental aspect of the CPA is that it can carry out work for the user asynchronously. The user can specify the details of some task to be completed on the thin client, at which point the mobile device disconnects from CAMCS,

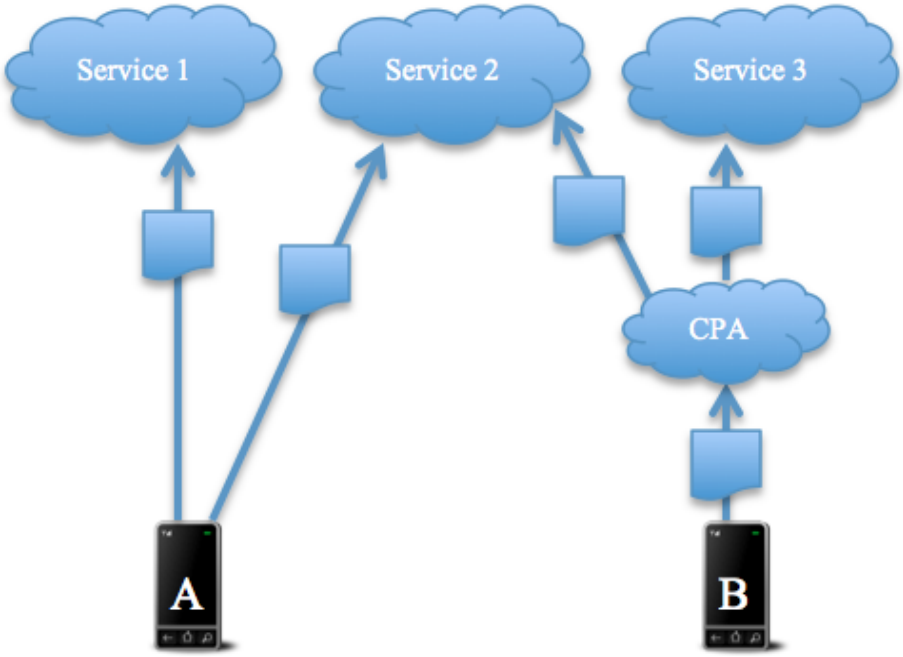


Figure 7.1: Rather than wasting resources uploading a file twice to two different services individually (device A), the user uploads the file once to the CPA (device B), which then sends the file to each of the user's service accounts.

and the work continues, with the results saved for when the user is ready to receive them. One area where this may be particularly useful is intensive data processing, especially if it is expected to take a large amount of time. As mobile devices have increased the mobility of office/lab staff, having access to such an application while on the move can be helpful, and the cloud approach can work with large and complex datasets that a mobile device may not be able to efficiently process.

In the early project work by the author of this thesis [96], a solution was implemented where the CPA would perform database queries on relational databases running on Amazon RDS. The CPA would wait for the query to be executed and save the result set for the user. This was difficult to implement, due to the nature of the different types the query could take, and the simplicity of the form-based user interface not being intuitive for the novice end-user to specify what was required. For implementing this application model with CAMCS, another direction was taken; rather than taking data from relational databases and setting up the required authentication and connections, an approach was taken whereby processing jobs would be performed on scientific data. A scientist could set a task to carry out some data processing on sets of experimental results, and get the result later.

In the absence of scientific datasets and software programs for various fields, XML datasets were used. The Computer Science and Engineering Department of the University of Washington offers freely available datasets on their website [94]. These range from data on protein sequences, to data from NASA on star systems. Many of these datasets were large in size; one dataset called Mondial, which contains information on different countries around the world, compiled from the CIA world factbook, was chosen because of its smaller size. A RESTful web service was developed as a separate application deployment from CAMCS, that could carry out statistical calculations on this data. To enable this, the Apache Commons Math library [78] was included in this service.

The flow of this work is as follows:

1. Offload of Data Processing Task to CAMCS

   The thin client is used to specify the location of the XML data by URL, and to specify the type of processing that is required from a list (in this case, statistical). The data is sent to CAMCS, which hands them over to the user's CPA, where the task data is examined.

2. Data Processing Begins at Service

   If the user has requested statistical calculations, the CPA contacts the cloud statistical service, passing it the URL to the XML dataset. At this point in time, the CPA already knows the services available (no service discovery takes place). The data processing information is passed to the calculation service. A CAMCS call-back URL that the calculation service can use to send the result back to the CPA is also passed. The service carries out the processing on the data (it calculates statistics such as the mean and mode on population data for all cities in the countries part of the dataset).

3. Data Processing Result Call-Back to CAMCS

   When finished, the service will call-back to CAMCS with the result data, which the user can fetch on their mobile device using the thin client, when they are ready. CAMCS finally marks the user's data processing task as complete - see Figure 7.2.

Another feature is that the CPA can provide real-time status updates on the progress of the data processing. The mobile thin client contains a record of the offloaded processing task, and when they open it, the CPA feeds status

updates to the thin client.

Of note is a difference between how the statistics web service was implemented compared to the database service in the early work. The statistics service is a RESTful web service. In the earlier work, the RDS service was a SOAP-based web service. One of the difficulties encountered with the SOAP-based RDS service, was that for long running quires, the Apache CXF software used at the CPA to contact the web service, would time out while waiting for the result. Apache CXF includes an asynchronous call mechanism to overcome this. However, the REST-based approach, even though it is easier to use over HTTP than SOAP, does not feature an asynchronous web service call. To avoid time outs, a call-back feature was implemented.

Advantages to this approach include the useful aspect that the web service will have libraries available to it that may not be present on the mobile device. As mentioned previously, the Apache Commons Math library was used to calculate the statistics. Other scientific libraries available include JScience [80], which were also included but were not used. It would not be trivial to calculate such statistics if done on the mobile device without these libraries.

Other advantages include the fact that the user does not need to wait for a specific piece of client software to complete the data processing, which may be prone to interruptions. The user can set the task with the CPA using the thin client and go on to do other work, or leave the office for the night and turn off the local equipment, which may have otherwise been left on and used for the processing task. The user can check in with the CPA on the go with the thin client for progress updates when required. There are difficulties and limitations in this approach that will be evaluated in Section 7.3.

Some other cloud-based solutions to data processing are available. Chen et al [17] developed a "k out of n computing" solution to perform both data processing and storage with remote services for mobile cloud, with a view to achieving energy efficiency and reliability objectives. In particular, their framework can adapt to dynamic network topology changes, such as more nodes becoming available, or nodes disconnecting, in a mobile network environment while connected to a cloud service. Huang et al [26] developed a secure data processing framework for the mobile cloud, known as MobiCloud, to provide processing on data collected from mobile devices, such as location data. They develop a proof-of-concept application called FocusDrive, to disable and enable text messaging facilities on the mobile devices of young drivers while on

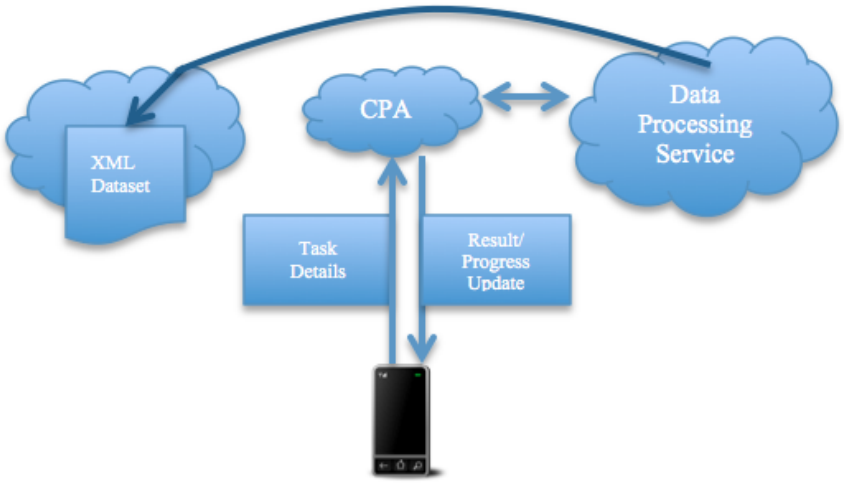


Figure 7.2: The user sends the data processing details, including the URL of the XML dataset, to the CPA. The CPA then contacts the cloud data processing service with the details, which begins the processing. The result is sent back to the CPA. The user can also receive progress updates.

the move, based on the speed of the device, the location of the device, and traffic conditions at the user's location.

7.2.3 Group-Based Collaboration

The fundamental premise of CAMCS is that users will offload tasks they require to be completed from the CAMCS Client running on the mobile device to their CPA. Another desirable goal is that CPAs should be able to communicate and work together. By doing so, a CPA can share data and discovered cloud services with the CPAs of other users. This will be beneficial for CPAs where users have similar interests, or who work on related tasks. A group of friends may have CPAs which work together to suggest activities of mutual enjoyment to their users. Shared interests that the CPAs are aware of will drive this functionality. To support collaboration, the notion of groups, has been added to CAMCS. This will allow CPAs collaborating together to complete shared tasks to the benefit of their users. Any CAMCS user can create a group, and other users are then able to join these groups. Several components are modified or introduced for this model:

1. Group Tasks and Milestones

   To support the concept of a task that can be shared among the CPAs of the group members, a new type of task was introduced, a *Group Task*. There-

fore, a distinction must now be made between an *Individual Task*, and a group task. Individual tasks correspond to the original single-user tasks. A group task is made up of many milestones. Currently, a milestone is an individual task. A milestone can be assigned to a CPA within the group, who will then be responsible for executing the milestone individually - see Figure 7.3.

2. CPA Roles

   CPAs joined to groups are assigned different roles. The CPA of the creator of a group becomes the group leader. Whenever a user joins their CPA to a group, the CPA is assigned a role. The leader defines the roles required within a group; these can correlate with the roles of the individuals who own the CPAs and their context (more information is given on this in the results Section 7.4, where an example use-case is described). Milestones also have a required role for completion. Therefore, each milestone is assigned to a CPA which meets the required role within the group.

3. Milestone Pre-Requisites and Execution

   Each milestone has a deadline for completion, along with pre-requisite milestones, which must be completed before a given milestone can begin execution. When a milestone has completed, it sends a message to the group to inform it that it has completed, and any milestones waiting on its completion, can now be assigned to a CPA within the group and begin execution.

Regarding milestone execution, this can be as simple as corresponding to an activity that the user who owns the CPA responsible for the milestone completion should carry out. Additionally, the CPA, like with individual tasks, can use cloud services to complete the milestone. In regards to a setting such as an academic institution, or a business enterprise, these services could exist in the form of internal private cloud services, that work on and with existing company infrastructures, such as databases, file storage/sharing, or workflow jobs. The advantage of such a model provided by CAMCS to CPAs, is that it allows users to have tasks scheduled for them depending on their role within the group. This will allow users the flexibility to carry on working, without having to arrange to meet in person or by video conferencing to assign tasks or check on task progress; this can be difficult depending on where in the world users are working from, or daily timetables/schedules. Many existing web-based

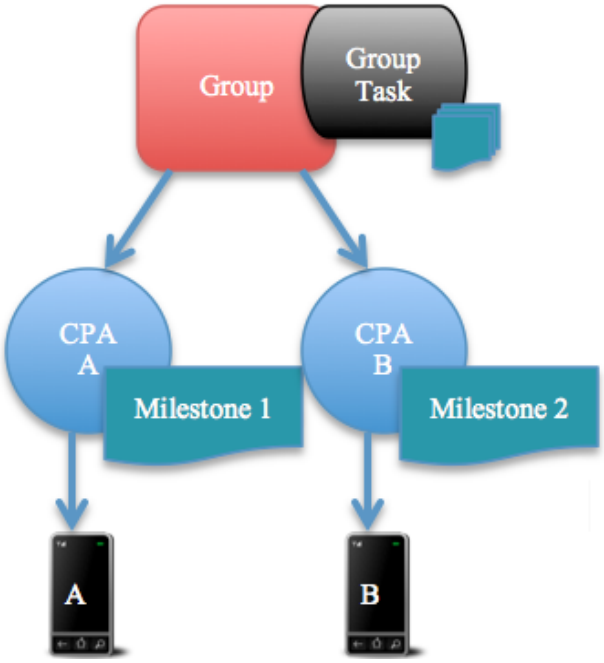


Figure 7.3: A group within the CAMCS middleware receives a group task. The group task has several milestones, two of which, have been assigned to two different CPAs. Details of each are sent to the user's mobile devices.

project management tools exist, but the CPA approach will result in task data optimised to the user's role, and will be delivered directly to the users mobile through their CPAs. It does not require user intervention, aside from initial task creation.

If required, and of benefit, the CPA can inform the user through the thin client of the progress of a milestone. Also, if the milestone requires the use of cloud services, the CPA can also interact with the user, by requesting additional information. For example, consider a group task requiring the gathering of reports on common causes of calls to a technical support group from the telephone operators. The CPA working on the milestone of an individual operator may ask their owner if it should include calls that required the operator to visit the troubled employee in the office. It might also ask the user which time-range it should gather data for, or should it exclude calls that were passed to another operator.

This will be of great benefit to companies and groups who require mobile enterprise for organising and coordinating work among employees who may be scattered in various locations worldwide, or who may be out-of-office visiting clients. Such employees will have access to their CPA and group milestones through the thin client on their mobile device.

7.3 Implementation Challenges and Evaluation

During implementation of the application models, several challenges and difficulties to the approaches that have been discussed were identified, as well as areas for improvement in API design for mobile devices. This section will describe how these challenges presented for each application model, and what steps were taken to solve them during implementation.

7.3.1 File Synchronisation

7.3.1.1 OAuth Authentication

In order to access the accounts of the different cloud service providers, the user must authorise CAMCS to access their service account by first authenticating themselves, and granting permission for the required operations. For all of the service providers used for the application models, OAuth [127] is the security access scheme employed. At development time, Facebook and Google used OAuth version 2, and Twitter and Dropbox used OAuth version 1 (by the time this paper was written, Twitter provided OAuth 2 support). In both versions of OAuth, the application requesting access to the user account with the service provider is given access credentials in the form of an access token/key/secret. With OAuth 1, a second access token/key/secret, sometimes called a "value", is also provided. When an application requires access to the user's account, they present the access token (and the value in the case of OAuth 1) with the request, and if the credentials are valid, the application is granted access. The main benefit of this approach is that the application that wishes to use the service provider on behalf of the user does not need to know and store the user's username and password for that service.

The flow of authentication and gaining an access key for most applications is as follows. The developer has to register their application with the service provider, and obtain an application key. CAMCS was registered with each of the service providers used in this work, and a key was obtained in each case. When the user wishes to allow CAMCS and their CPA access to their service provider account, in Android, they are redirected from the mobile application to the website of the service provider through a WebView, presenting the application key. The user logs in with his/her own username and password.

The user is then given a choice to grant access to the application for various operations (sometimes called "scopes"). Once the user has granted access, the mobile application is called back with the access key (and the value in the case of OAuth 1). These are then stored on the mobile device for future use.

The difficulty for CAMCS is that the mobile device does not require or use the access credentials. The CPA operating in the cloud is the entity that will be working with the service providers; therefore the CPA needs to be provided with the access credentials.

If CAMCS were a web application accessed from the desktop PC browser, the web application would receive the call-back and store the credentials. In this case, the credentials would be sent straight to the CPA. This however would not be optimal for the user experience. Asking the user to leave the mobile thin client, and open a corresponding website with a browser for CAMCS to perform the authentication would defeat the purpose of being a mobile thin client application.

To overcome this, a RESTful web service was implemented on CAMCS. When the user has authenticated with the service provider on the CAMCS Client, the access credentials are sent over HTTP (not encrypted in this implementation, as discussed in Chapter 5, but recall that the OAuth key does not expose user data) to CAMCS to be stored with their user account. The CPA can then access the credentials stored with the user's account details on CAMCS, to carry out operations with the service providers - see Figure 7.4. The access credentials typically expire after some time; CAMCS can fetch fresh credentials when this occurs, without user intervention.

7.3.1.2 Service Provider APIs

This difficulty also relates to authentication with the service providers. To implement the authentication flow, the Spring Android project [40] was used, which uses components of the Spring Social project. They simplify the work required for authentication with service providers. The developers of Spring Social have only implemented official support for Facebook and Twitter authentications using OAuth. There are several community driven projects for other providers, such as Dropbox and Google. None of these community driven projects have been ported to the mobile platform, and their implementation remains solely focused for use with Spring Social on web applications.

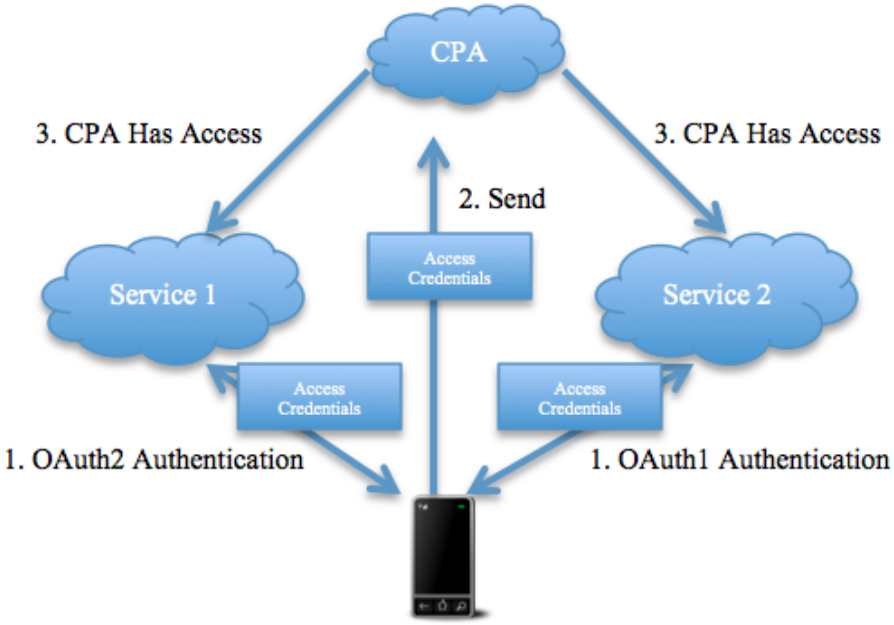


Figure 7.4: To get the access credentials to where they are needed with the CPA, the user must authenticate for each service on the mobile device (normally through a WebView). The keys are then sent to CAMCS using a RESTful web service. The CPA can then access each service on the users behalf.

These could be ported to be compatible with the Spring Android components, but this requires some development effort.

Rather than doing this, the Android APIs available from Dropbox and Google were used. This involves downloading JAR files from the different service providers, packaging the thin client with them, and using them in the code to carry out the authentication flows. By the time this was finished, CAMCS was compiled with several JAR files; those for Spring Android, Spring Social, Spring Social Facebook, Spring Social Twitter, Dropbox, and Google Play services. The file sizes of these start to build up quickly. Moreover, all of these services transfer data in JSON format, but these JAR files often contain different versions of JSON parsers, which all do the same thing, taking up even more space while doing so. One cannot set each of the APIs to use a single JSON parser of choice and remove the rest - see Figure 7.5. All of the service providers authenticate using OAuth tokens, but each provider seems to implement the authentication flow differently, rather than using some standard. Spring Social aims at resolving this, but as described, only supports Facebook and Twitter, relying on community projects for other implementations, which have not been readily ported to Spring Android.

To authenticate with Google, Google Play Services was used . This contains

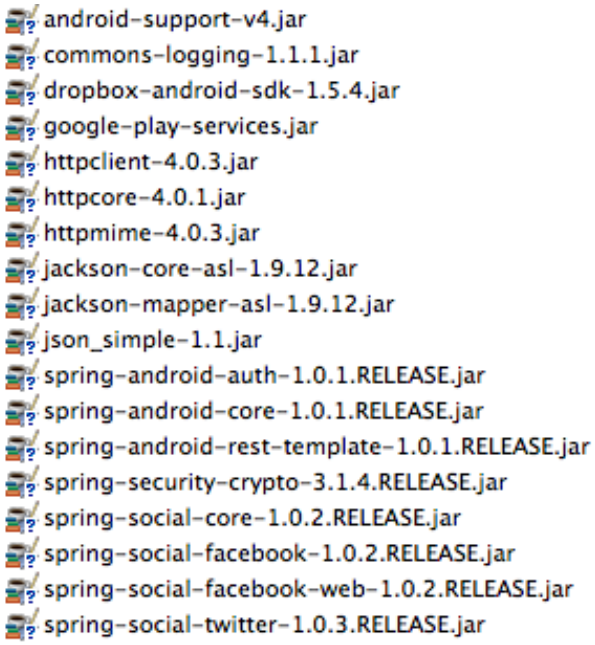


Figure 7.5: Screenshot from the Eclipse IDE of the required JAR files for the Android thin client for each service provider. Duplicated functionality can be seen; jackson JAR files are for JSON parsing, the json-simple JAR is required by another file but contains the same parsing functionality as Jackson.

an AccountManager, which is supposed to again provide common features for getting access tokens, but, like Spring, requires community built authenticator modules for the different providers. Aside from the expected need for different interfaces for the differing features of the different service providers, it would be much easier for developers, for the common task of authentication, if there was a standard API that would work for all out-of-the-box, since they all use OAuth authentication. In addition, if the user of CAMCS wanted to add another provider not already supported that uses OAuth, the CAMCS Client would need to be modified to support each new provider's different implementation of the authentication flow, so extension becomes difficult. If a standard API existed, the user could add new service provider accounts without the need to modify the mobile thin client. The different APIs take time to learn and implement, and increase the size of the applications deployed to mobile devices because of the required JAR files.

7.3.1.3 Synchronisation from Service to Device

The current implementation does not implement a download mechanism to synchronise files from the various cloud services to the mobile devices. Many

service providers already implement a push mechanism; this will automatically send a file uploaded to the provider, down to all the other devices that use a native application. On the mobile device, this would be a waste of resources if files were downloaded more than once both from the CPA and the native application.

If this were to be implemented, it would require a means for the CPA to be aware of when the user uploaded a file to the service from other sources, such as a web browser. This could be achieved by polling, but this introduces extra traffic to the service provider, which would be wasteful if no new files or updates have been added to the service provider since the last poll. A better solution would be an event notification API, which could alert interested parties, such as the CPA, when a new file has been added or of any update to existing files. This requires the service provider to implement such an API. As an example, Dropbox provides the sync API, which allows notifications to be sent after events such as new files being added occur. However, the API currently only exists for native Android and iOS implementations; the ability needed here is to inform a third party on the users behalf, in this case the CPA, so that it is aware of the file state at the service providers.

7.3.2 Data Processing

7.3.2.1 Service Extensibility

The main question facing the development of the data processing is how to expand its operation, and make it easier to invoke. As it stands, the statistics service will only work with an XML dataset that shares the same schema as that of the Mondial XML dataset it was developed against (or any specific dataset that a service is designed for use with). The statistical service developed for the Mondial dataset has to parse the XML dataset, and expects to find certain tags and attributes that can be used for calculations. While there may be other datasets representing similar data (for example, ethnic population data on European countries) that uses the same markup, it is still a fragile service.

It may be prudent if a scientist who has data to process could easily specify their own calculations that they are interested in performing to a service, so that it could readily work with different XML schemas. These could be up-

loaded to the CPA from the mobile device and be sent to the processing service. The calculations would have to specify what data to work with, and what calculations should be performed on it. Ideally, the user should be able to express the desired calculation on the thin client interface. This may be achievable with cooperation from those who develop software specifically for data processing of large formatted data. If this were not the case, as it is now, a different service would have to be developed for each different XML schema, limiting the scale of the data processing service.

7.3.2.2 Discovery of Data Processing Services

Currently, the data processing service runs in another web application, separated from CAMCS. This is because different service providers will provide their own services for processing different types of data; it is not something that the CPA can do itself at present. In this situation, services that can perform different processing on data need to be known to CAMCS. A service must be able to describe exactly what it does, what data it expects, and how it will return the results. In addition, CAMCS needs to be able to compare the dataset and instructions passed by the user, with these external services to find what service will match the request.

For this work, locations and types of services were hardcoded into CAMCS, so that it knows where to find a specific set of services that carry out specific calculations on defined datasets. Clearly, the discovery solution from Chapter 7 would be of use here, with modifications to allow the required data for the calculations (such as specifying which mathematical calculations to perform on which specific data in the XML document) to be specified as part of the service description.

7.3.3 Group-Based Collaboration

The implementation challenges faced in regards to CPAs collaborating together are mainly drawn from how CPAs can communicate with each other, how they can share data or tasks, and how the milestones are structured and assigned to the CPAs. For managing collaboration and communication between CPAs, the Collaboration Manager was introduced.

7.3.3.1 Collaboration Manager

If CPAs are to collaborate, they need to become aware of the existence of other CPAs. While any CPA could be made aware of any other CPA, possibly based on a real-life "friends" model, for the purposes of this work, this is restricted to groups. Only CPAs in the same group can communicate with each other. However, as a group may contain many CPAs, it is not practical for each CPA in a group to know about every other CPA in a group either. For this purpose, the *Collaboration Manager* was introduced.

The Collaboration Manager facilitates communication between two CPAs. Each group has an instance of a collaboration manager. A CPA knows which groups it is a member of; therefore it knows how to find the collaboration manager for a given group. If a CPA needs to discover another CPA based on a role or similar milestone, they consult the collaboration manager. However, in this implementation, each milestone is assigned to each CPA individually and they are solely responsible for executing it. As a result, a CPA does not need to consult directly with another CPA for milestone completion.

Currently, the collaboration manager is used for milestone assignment to CPAs. When the group leader starts a new group task, the collaboration manager will assign the first milestone to a CPA who has a matching role within the group. When a CPA is given a milestone, it records which group the milestone belongs to. When it has completed the task, possibly using cloud services, or their user has completed the physical work, the CPA calls back to the collaboration manager, to indicate the milestone is complete. At this time, if appropriate, the collaboration manager will assign the next milestone (see Subsection 7.3.3.2) to be completed to a CPA with a corresponding role. It should be noted, that for a group task, several milestones can be assigned to CPAs and be in a state of execution at one time; when stating that the collaboration manager assigns the next milestone, this applies to the next milestone that had to wait until the current milestone was completed. Assignment of a milestone will start execution.

7.3.3.2 Milestone Structure

Milestones are considered to be self-contained tasks, however they may have pre-requisite milestones required before they can start execution; careful con-

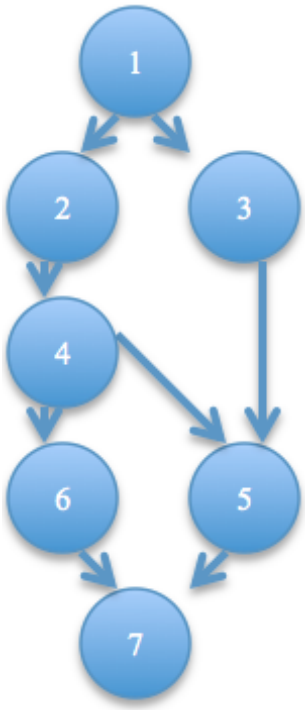


Figure 7.6: A graph G = (V, E) representing a tree of milestones for a group task, where V is the set of milestones and E is the set of milestone pre-requisites. 1 is the first milestone; it has two child milestones, 2 and 3. 2 and 3 have 1 as a pre-requisite parent milestone. All pre-requisite milestones must be completed before a milestone can start; before 5 can start, 3 and 4 must be completed. Milestones such as 2 and 3 can execute in parallel when 1 has completed.

sideration has to be given as to how milestones are structured. Milestones may also have deadlines for completion. A group task can be thought of as a graphical representation of a tree, G = (V, E), where G represents the task, V is the set of milestones that make up the task, and E is the set of pre-requisites of milestones. One milestone is designated the first milestone. Each milestone may have one or more child milestones in the graph (the first milestone must have at least one child milestone), or possibly no child milestone. Each milestone (except the first) may have multiple parent milestones in the graph - see Figure 7.6. Each parent milestone is a pre-requite milestone in the graph (represented by an edge in E), and all pre-requisites must be completed before the current milestone can be assigned to a CPA and executed.

The difficulty in implementing this in code stems from the MongoDB database embedded document structure. If all milestones have object references to each other milestone, there would be many cyclic references. As a result, each milestone, as an instance of an individual task, is given a task ID, and each milestone stores the IDs of their pre-requite milestones. Each milestone also stores

the IDs of each child milestone that should be started once it has been completed. This resembles a double-linked-list structure.

The collaboration manager uses all this information when assigning tasks to CPAs. Given a complete milestone, it will check each of the pre-requisite milestones for completion, and if all are completed, it will assign all the child milestones that should be started once the current milestone has completed.

Milestone results, as with other task results in CAMCS after being completed, are pushed to the mobile device of the user with Google Cloud Messaging. When the milestone has completed, the milestone uses the group ID to contact the collaboration manager; if it was a physical milestone that the user has completed in person, the mobile thin client is used to contact the collaboration manager of the group to inform the group of milestone completion.

7.4 Results

Experiments were performed to evaluate the timing performance of both the file synchronisation and the data processing functionality of CAMCS, which are now presented. An office-environment based group task use-case was used to demonstrate the group-based collaboration functionality.

7.4.1 File Synchronisation

To evaluate the file synchronisation performance, over five different runs, the time to upload a PNG image file of size 112KB was measured - see Table 7.1. Specifically, the following measurements were recorded: the time taken to upload the image to the CPA from the mobile device, and the time for the CPA to upload the image to Facebook and Dropbox. The mobile device used was a Samsung Galaxy S3, connected to the Vodafone Ireland operator. The mobile - CPA upload took place on a HSDPA+ cellular network connection. The CAMCS middleware was running on an Apache Tomcat version six servlet container on the cloud server. The cloud server is located within University College Cork, Ireland, and features a 1.7GHz CPU and 2GB RAM. The timing data was collected from logging statements placed in the code. The upload of the image file to Dropbox and Facebook from the CPA took place in sequential order. If threads had been utilised to do this concurrently, the total time would

have been smaller, the mobile to CPA communication time plus the maximum of the server upload times.

To compare this with the performance of uploading with the individual Android apps, the times to upload the same image with the native Facebook and Dropbox Apps with the HSDPA+ connection were measured over five runs. This timing data was obtained with a stopwatch, from the time the upload (or equivalent) button was pressed on each app, to the time when the notification that the upload was completed appeared. The timing is less accurate as a result, but the greater duration is still clear - see Table 7.2. Clearly it takes even more time, and as a result, energy and money, since the user has to upload the image twice using two different apps, whereas with the CPA the user only has to do this once.

In Table 7.1, the only time the user has to spend waiting on their mobile device to finish upload are the times for the Mobile - CPA communication in column two. Therefore, the total time for the images to reach the service providers from the mobile device in column 5 is not the total time the user has to spend waiting for upload on their mobile device. Contrast this with the total result in Table 7.2. The user has to manually upload with the applications for each individual provider, so the total time in the fourth column is the total time the user must spend waiting for uploads to complete, greater in all cases than the wait time with the CPA model.

7.4.2 Data Processing

The data processing service was deployed in the same Apache Tomcat six servlet container and cloud server as CAMCS. The XML parser used was XMLPULL [107]. For a comparison test, a small Android application was implemented, featuring a service which would carry out the same XML parsing as was performed on the server. The Android XML parser is the aforementioned XMLPULL parser that used on the server, so the comparison is fair in this regard of implementation. For the cloud service, the XPP3/MXP1 implementation [60] of the XMLPULL parser was used, as this is the same implementation found on the Android platform, as determined by that fact that they share the same package structure (the other implementations have a different package structure to the version found on Android).

Before the tests were run, Tomcat was restarted. The time was recorded with

Table 7.1: The time in seconds over 5 runs to upload a 112KB image from the mobile device to the CPA, and subsequently from the CPA to Facebook and Dropbox.

Run	Mobile - CPA (s)	Facebook (s)	Dropbox (s)	Total (s)
1	2.25	2.74	1.62	6.62
2	4.39	3.25	1.52	9.16
3	1.93	2.01	1.53	5.47
4	2.63	2.09	2.67	7.39
5	1.25	2.10	1.58	4.94

Table 7.2: The time in seconds over 5 runs to upload a 112KB image to Facebook and Android using the individual native apps.

Run	Facebook App (s)	Dropbox App (s)	Total (s)
1	15.0	4.1	20.1
2	5.1	2.6	7.7
3	6.9	3.7	10.6
4	7.0	3.3	10.3
5	6.9	5.2	12.1

logging statements in the code to fetch the XML file, the time to parse the XML, and finally, the total time over five runs - see Table 7.3. The total time includes the time for preparing the XML parser, converting the XML file to a String for the parser, and the calculation of the statistics. The majority of the time is spent on conversion of the XML to a String. The parse time decreases with each parse after the first. Another test was carried out by restarting the server again, and the same trend of decreasing parse time was repeated after an initial longer time for the first run. The larger the dataset in size, the larger the number of XML nodes that will need to be parsed, which will take up more of the limited memory if done on the mobile device. This will also take more time, and more energy from the battery.

As previously described, a small Android application was implemented to carry out the same XML parse as the cloud data processing service for comparison purposes. This ran a service thread, which executed the same Java code found on the cloud service on the same XML dataset - see Table 7.4. The results show that the XML fetch over the cellular network connection took longer than the cloud service, as one would expect due to the poorer quality connection. The XML parse consistently took around half a second, and did not show the same decreasing time trend as the cloud service. Surprisingly,

Table 7.3: The XML fetch and parse times in seconds over 5 runs for the data processing cloud service along with the total time.

Run	XML Fetch (s)	Parse (s)	Total (s)
1	0.96	1.11	4.58
2	0.38	0.67	3.77
3	0.60	0.43	4.11
4	0.38	0.08	3.37
5	0.35	0.03	2.35

Table 7.4: The XML fetch and parse times in seconds over 5 runs for the data processing Android test application along with the total time.

Run	XML Fetch (s)	Parse (s)	Total (s)
1	0.88	0.50	6.11
2	0.74	0.42	7.60
3	3.53	0.44	12.36
4	2.09	0.43	11.31
5	2.81	0.43	11.73

this means that the first two runs of the parse on the cloud server were actually slower than the mobile device. The direct cause of this is unknown, but some internal caching within the Java Virtual Machine, particularly with heap objects of the XML nodes, is suspected considering the same dataset was used with each run of the test.

Both implementations use a Java InputStream for the fetch. The bytes from the stream are then read and converted into a String for the parser input. However, when the Android client fetched the XML dataset, it also brought along formatting characters, specifically, newline characters (\n) and whitespace. This interfered with the tokeniser of the XML parser, and they had to be removed from the String (using a String replace method) before the XML string was passed to the parser. This removal operation took around four seconds each time, and is the biggest contributor to the total time on the Android device. As a result, the total time was always longer on the Android test application, even for the two runs where the parsing operation was quicker than the cloud service. This removal process did not need to be performed on the cloud service; no newline or whitespace characters were fetched in the InputStream.

Once the work is complete, the call-back is made to CAMCS, which forwards the result to the user's CPA. The CPA then uses Google Cloud Messaging to inform the user that the result is ready, and they can then view the result in the

thin client application.

With the cloud service, the mobile user does not need to upload data from the mobile device over the network connection once the data URL is specified. No energy is used up on the mobile device for the parse, and the parse is unaffected by interruptions on the device. The device is also free to complete other work.

7.4.3 Group-Based Collaboration

To evaluate the group-based task application model of CAMCS, groups based on teams and departments within a company were modelled, which can use mobile enterprise to assign and coordinate tasks and corresponding milestones within their teams. Departments may include the IT department, sales, quality assurance, or management. Each department in a company will have members who are responsible for different kinds of work. In the sales group for example, there may exist a secretary who prepares sales reports from company orders in databases, and a sales representative who takes orders from customers and places them into the system.

Consider a group task. The lead member of the sales team may require a document prepared which details the sales for the past year. This will require two milestones: (1) the sales representative must gather the required sales data from their database, and (2) the secretary must then prepare the report using this data. For this task, it could be completed in two ways. Either the sales representative could gather the data by his/herself, and then physically hand it to the secretary to type the report. Or, the collaboration manager for the group could assign the sales database task to the CPA of the sales representative, who has a role in the group as a SALES_REP. The CPA can then use a private cloud service within the company to extract the required information for the sales from the database. Once this milestone has been completed, the collaboration manager will be notified, and will assign the report preparation task to the secretary's CPA, which has the SECRETARY role. This CPA can then use a document preparation service to prepare the report with a given format and template, when provided with the data.

The collaboration manager was implemented, and successfully assigned given milestones for such group tasks. In earlier work [96], some sample services for database operations that could be deployed to the private cloud of a company

were developed and evaluated. The implementation also has other enabled features: by considering the deadline, the CPA reminds the user through the thin client of work they must complete for any approaching deadlines. A CPA, which represents a user who is currently busy with a set maximum number of tasks, will also look for another CPA within the group who has a role that supports completion of the milestone. Compared with web-based project management solutions, using CPAs to coordinate group tasks as demonstrated removes the need for manual management by a designated squad/team leader; this approach can respond to completed milestones on the fly.

7.5 Conclusions

In this chapter, CAMCS was used to implement three mobile cloud applications models that were proposed in Chapter 3; namely file synchronisation, data processing, and group-based task collaboration. The implementation of these applications, and benefits of the CPA approach with CAMCS, were described. The challenges faced while implementing these functionalities were evaluated and discussed. For the file synchronisation, these include OAuth security implementation issues and heterogeneous APIs for different service providers. For data processing, they include scalability and calculation specifications. For group-based collaboration, this includes how each CPA can be aware of each other and how tasks are structured and assigned.

Timing results were presented for file synchronisation and data processing. For the file synchronisation, the timing results showed fast performance over the cellular network. When contrasted with uploading files individually using native Android apps, the time-savings were evident. For the data processing application, the time to fetch and parse XML datasets on the server was also quick, with results comparable to or faster than the same parsing operation on the Android test application developed for comparison purposes. For the group-based collaboration, groups were modelled based on departments within a company, where department members would work together to complete group tasks. User CPAs joined these groups and were assigned tasks based on their roles within the groups, corresponding to the role of the user within the department.

It was highlighted how effective CAMCS can be as an enabler of these three

applications, compared to existing approaches. For file synchronisation, the CPA can save resources such as time, energy, and money, by quickly performing the synchronisation with different service providers; resources need not be wasted uploading files multiple times to different service providers from the mobile device. For data processing, heavy processing work can be offloaded to the CPA, so as not to use up the hardware resources of the mobile device. For group-based collaboration, the collaboration manager of the groups can assign tasks to the CPAs of users within groups, based on their role within the group. These tasks are delivered directly to the mobile device. Once the task has been created, the CPAs and the group collaboration can take-over this work asynchronously without user intervention. This will support the needs of the mobile enterprise, which need to be able to assign and coordinate task milestones to their employees wherever they are located, if arranging meetings is difficult.

The challenges highlighted in this chapter such as authentication, API design, and lack of data standards for processing, will be of crucial importance going forward as mobile cloud development increases, and mobile client software adopts the paradigm.

The next chapter will present the results of experimental evaluations performed on CAMCS running on the Amazon EC2 public cloud, as well as experiments performed on the CAMCS Client running on a mobile device. These will be analysed to determine if CAMCS meets the integrated user experience requirements of this research.

This chapter was based on [97] and [102]. Various modifications have been made to the text throughout the chapter, to fit with the flow of the thesis.

Chapter 8

Experimental Evaluation

8.1 Introduction

The purpose of the CAMCS middleware is to deliver cloud-based services to mobile devices, generally in a disconnected, asynchronous fashion. The argument is that by taking this approach to mobile cloud computing, the practical difficulties of other solutions, such as high energy, high bandwidth, and constant connection requirements, can be overcome, therefore removing the factors that would result in a detrimental user experience. For this chapter, an evaluation of the performance of CAMCS has been undertaken, to understand if this solution is capable of achieving these objectives.

The performance of CAMCS, while deployed on a public cloud, has been studied in several scenarios, and the results of this study will be presented. Additionally, the CAMCS Client, running on Android-based mobile devices, is also studied and presented. A user study of CAMCS was also undertaken, where volunteering participants used the CAMCS Client, running on his/her own Android mobile device, to perform tasks with CAMCS for several days. Each participant completed a survey at the end of the study; the responses are presented and analysed. With these evaluations completed, this chapter will focus on determining if CAMCS meets the requirements that were outlined as required for an integrated user experience of mobile cloud applications. The analysis will also determine how CAMCS conforms with the models outlined in Chapter 6, to verify the resource requirements when contrasted with other mobile cloud computing solutions.

Table 8.1: Relevant properties for Spring Boot and Apache Tomcat. The relevance of the maxThreads and acceptCount properties from Tomcat will present themselves during evaluations.

Property	Value
Spring Boot Version	1.2.4.RELEASE
Java Version	7
Apache Tomcat Version	7
Servlet Version	3.0
Tomcat maxThreads	200
Tomcat acceptCount	100

8.2 Evaluating CAMCS on a Public Cloud: Setup and Methodology

For the purposes of all the experiments presented in this section, the result data has been gathered while CAMCS was deployed on the Amazon AWS public cloud, using the free tier (however in some cases, the experiments have used resources beyond the limits of the free tier).

8.2.1 Amazon AWS Experimental Setup

CAMCS has been implemented as a Spring Boot application; it compiles to one Java JAR file. The benefit of Spring Boot, is that it contains an embedded Apache Tomcat servlet container. As such, the JAR file can be executed as a normal Java application, anywhere running a Java virtual machine (JVM), and does not require an existing Tomcat installation. Some relevant properties of Spring Boot, and the underlying Apache Tomcat Servlet container are presented in Table 8.1.

Amazon AWS provisions virtual machines for users with the EC2 service. These are launched using an Amazon Machine Image (AMI), which is essentially a template for the underlying operating system, along with any software desired for runtime purposes. The AMI is launched onto an EC2 VM instance. For all experiments, an Ubuntu Linux 14.04 (Hardy) image was used for the base of the AMI to be launched on the EC2 instances. For the experiments, the CAMCS JAR file was uploaded onto an existing virtual machine running this AMI.

For runtime purposes, CAMCS uses the Docker platform [27]. Docker is designed to ease deployment for applications, with a "run everywhere" goal. The motivation, is that rather than a developer having to individually setup and install all software required by the application to be deployed on each machine or virtual machine, Docker hides away these concerns. This is achieved by the *Container* system. Docker containers are essentially a self-contained operating system platform to run an application. A developer can create a custom Docker container by writing a "Dockerfile". The Dockerfile is the template for the container, and the developer specifies what software should be setup to run in the container, and what runtime actions should take place. Dockerfiles start with an "image" upon which to run the container; this image is created from a base "image", such as an existing Ubuntu Linux image (very similar to Amazon AWS AMIs). Then, the developer can add in any extra applications required, such as a database to support the main application, and specify what commands should execute when the container starts up. This Dockerfile is processed into a new, custom image. The image is then uploaded to Dockerhub, a public repository where thousands of Docker image are stored and shared. These images can be pulled into an existing Docker installation on a developer's own local machine or virtual machine. At this point, these images are run as new containers, and are available immediately. The developer does not need to install a local database and run it to support the application as was typical in days past; this is all handled by the Docker container system.

Docker was installed onto the original EC2 instance. The CAMCS JAR file was added to a new Docker image, based on the existing "java7" Docker image (an official image from Oracle). The image was then created and pushed to Dockerhub, where it was then pulled down onto the EC2 instance's own Docker installation, and launched as a container immediately. From this setup, a custom Amazon AMI was created. This custom AMI can then be used by EC2 to instantiate a new EC2 VM instance and run CAMCS in a Docker container, whenever horizontal scaling takes place by EC2.

Three other initial Amazon EC2 instances were used. One each for the service registry queried by CAMCS during discovery, and another for a dummy web services application, providing some custom services created for the purpose of evaluation. Finally, the third EC2 instance ran a MongoDB instance. This is the backing database for CAMCS. This was not, as described in the Docker scenarios previously, run within the CAMCS Docker container. This was because of the scaling requirement. If ten separate EC2 instances were created

Table 8.2: AWS t2.micro Instance Specifications, taken from AWS website. This instance is suited for general-purpose applications with low requirements, that may receive an occasional burst in traffic, which is handled by the Turbo capacity.

Feature	Value
vCPU	1 (High Frequency Intel Xeon Processors, Burstable with Turbo up to 3.3GHz)
Memory	1GB
Network Performance	Low - Moderate
EBS-Optimised Available	No
Category	General-Purpose

by auto-scaling, then there would be ten separate databases queried by each of the ten running instances of CAMCS; they all need to share one source of information, and so one separate instance was required.

Unless otherwise stated, all EC2 instances used are of the free-tier eligible type, t2.micro. This is categorised as a *general-purpose* instance for low demand web applications that can receive the occasional burst in traffic. The specifications for this instance are presented in Table 8.2.

8.2.2 Services Experimental Setup

The EC2 instance running the dummy services application ran a Spring Boot JAR application, but not using a Docker container. The services created for evaluation were, as has also been highlighted in previous chapters, the tourism places-of-interest (POI) service, the Twitter traffic service, a weather service, and the Google calendar service. Another service was created specifically for evaluating CAMCS with compute intensive applications, an N-Queens game. The N-Queens game is a classic example of a compute-intensive problem, and has also been used in other work in the mobile cloud computing literature, as described in the literature review in Chapter 2, to evaluate middlewares. In this implementation, N-Queens was created as a RESTful web service, and it takes in two URL parameters; a min and a max, integers representing the minimum and maximum values of a random number N generated at call time, to solve the N-Queens problem for. In initial evaluation, the max value was restricted to 13. Any value greater than 13 took a long time to solve, and anything over 16 ran on a local laptop for periods longer than 30 minutes without

finishing. The Java code is taken from the Computer Science Department at Princeton University [122], and the authors state that the code will solve recursively for any value of N <= 25.

A service description using the work from Chapter 5, was inserted into the registry application running on its own EC2 instance at runtime, so that CAMCS would be able to discover it for user tasks.

8.2.3 CAMCS Experimental Setup

The aim of the evaluations was to gauge the performance of the middleware running on cloud infrastructure, in terms of metrics such as response time, throughput, and latency, where hundreds or thousands of CPAs would be running within CAMCS at any given time. For the experiments carried out, Apache JMeter [61], a Java application designed to "'load test functional behaviour and measure performance", was used, to simulate users interacting with their CPAs. To emulate anticipated behaviour, each of these users would send a request to CAMCS, with a task to solve the N-Queens puzzle for some random number; for each task request sent, the min and max parameters were set at 3 and 8 respectively. The performance metrics gathered from JMeter, and the experiments performed in this chapter, are consistent with evaluations that have taken place for other work on mobile cloud middleware, such as in the thesis work by Qian Wang [143] and Huber Flores [38].

It is important to understand that the evaluations performed, while informative, differ slightly from normal expected CAMCS operation. Bearing in mind the sequence of events from Chapter 5 in terms of the sequence of events of task description, discovery, the choice of service by the user along with parameter input, followed by service consumption, and the receiving of a result HTML page, this cannot be replicated by the performance test all at once. Primarily, this is because these operations all occur in a disconnected, asynchronous fashion. The user is disconnected from their CPA between each step, while the CPA performs the work asynchronously; the user receives notifications at each step. Furthermore, the CPA cannot continue with service execution (for a new task), without receiving data parameters and/or context permissions from the user on the CAMCS Client.

As a result, for testing purposes, the concept of an *Anonymous CPA* is introduced. An anonymous CPA does not belong to any one user, but will complete

work for any user when a request is received by CAMCS on-the-fly. Additionally, all operations carried out by an anonymous CPA, for testing, are synchronous, blocking operations. No task description, service selection, or parameter input steps take place. A new RESTful service was added to CAMCS for these tests. This service takes in the name of a service that the CPA should discover from the registry (namely, the name of the N-Queens game service). This data is sent by JMeter for each sample during the test to the "/camcs/-task/put/anon/task/3/8/false/false" endpoint:

{"taskName":"testTask","serviceId":"Games","operationName":"N-Queens Cloud",
"taskData":{"dataMap":{},"contextPermissions":null},"service":null}

3 and 8 represent the URL parameters for the min and max values respectively for the N-Queens game. "Games" is the name of the Service within its Service Record in the registry, and "N-Queens Cloud" is the name of the Service Operation provided by the Games service.

To understand the performance of the architecture as a whole, the performance of the registry running on it's EC2 instance with service discovery must be known. Additionally, the performance of the N-Queens game running on its EC2 instance, must also be known. For example, evaluation results could measure the entire process from the CPA receiving a task and undertaking discovery, up to the point where the N-Queens game service is contacted, and a result is received by the CPA.

For the purposes of testing just CAMCS itself in isolation from discovery and service execution, boolean switches were introduced as parameters to this CAMCS testing web service. These can be seen in the URL path above; the first "false" is a boolean indicating if CAMCS should perform discovery during the processing of the request, and the second "false" boolean indicates if CAMCS should actually consume the N-Queens game service and wait for the response. True/false values are possible for both indicating which course of action the test request should take. If discovery is false, the CPA will use a cached copy of the Service Record (stored by CAMCS) from the registry; the registry will not be contacted for that request. If consume is false, then the CPA will run all tasks that the Task Executor described in Chapter 5 does, except that the N-Queens game service will not be contacted, and a result JSON String will be generated instead, which says that the "Work for the N-Queens task has been completed".

Once again, these operations will occur in synchronous fashion; once the initial request is received to the CAMCS testing endpoint described previously, the CPA will start discovery by contacting the registry (if the parameter is true, otherwise cached copy is used), and as soon as this is completed, the CPA will then move straight onto the consumption part of the sequence with the Task Executable. The result (real result or generated success string depending on the value of the consume parameter) will be returned back to JMeter immediately.

It should be clear from this subsection, that before testing the end-to-end CAMCS architecture, some evaluation of the dummy web service application running the N-Queens game, and the service registry application, both on their own EC2 instances, should take place first with JMeter.

All AWS experiments with JMeter in this chapter were performed over a Wi-Fi connection (unless otherwise stated), to the Eircom ISP in Ireland, with a measured download speed of 1.8Mbps, and an upload speed of 0.40Mbps.

8.3 CAMCS Performance Evaluations on Amazon EC2

8.3.1 N-Queens Game Application on EC2

JMeter was used to load test the dummy services EC2 instance with the N-Queens game service. JMeter was setup to use 1000 threads (users), which would repeatedly send a HTTP request to the service, each with N = 10. A ramp up period of 60-seconds was set, and the Apache HTTP Client version 4 was used. The 1000 threads repeatedly sent sample requests for a period of 10-minutes.

The JMeter summary report data can be seen in Table 8.3. A response time graph can be seen in Figure 8.1. The average response time is seen to be approximately 5.1 seconds over 112156 requests, with a median of 4.3 seconds, and a throughput of 184 requests completed per second over the 10 minute

Table 8.3: Metrics from JMeter while 1000 threads simulated users requesting to solve the N-Queens problem for N = 10.

No. of Samples	Average (ms)	Median (ms)	90% CI (ms)	95% CI (ms)	Throughput (requests/s)
112156	5161	4375	7159	9902	184.0

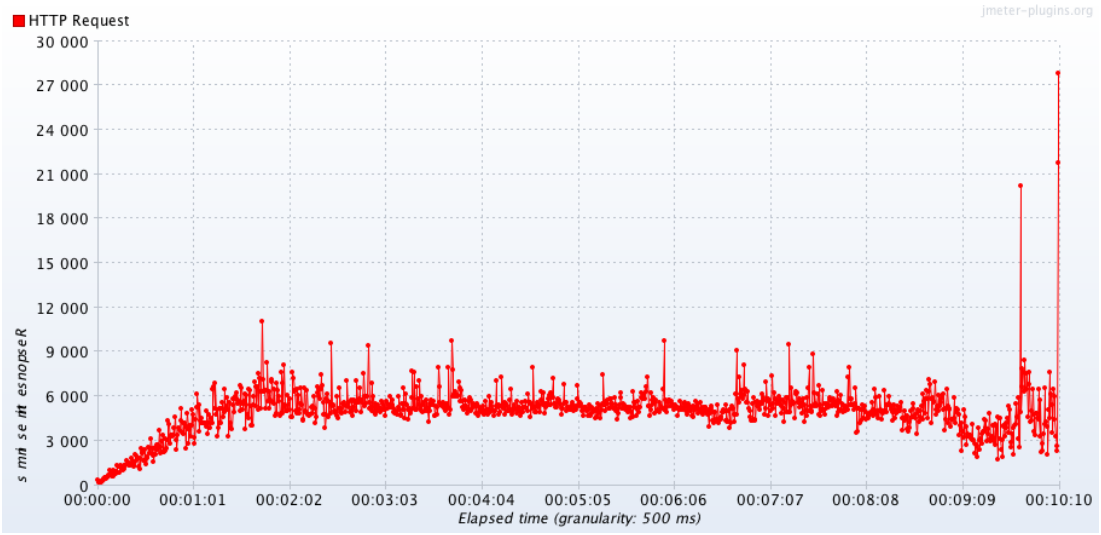


Figure 8.1: Response time graph from JMeter while 1000 threads simulated users requesting to solve the N-Queens problem for N = 10.

duration of the test. An error rate was also measured but is not shown below; the error rate can result from stale threads that have not received a response in over a minute. This reached 0.01% over the course of the test.

Looking at the EC2 instance, a graph of the CPU utilisation can be seen in Figure 8.2. As one would expect, the N-Queens application is a compute intensive application and CPU bound, reaching 100% CPU Utilisation for N = 10.

Moving forward with the evaluation, as described earlier, test requests confine their maximum value of N to 8, rather than 10 or greater.

8.3.2 Service Registry Application on EC2

As with the test of the N-Queens Game Service, an evaluation also took place with the same options and settings as with that test, again running for 10 minutes. With this test, JMeter sent repeating HTTP requests straight to the registry, querying specifically for the N-Queens Game Service record. Of course, this application can expect overhead from having to query the backend MongoDB database. Unlike the CAMCS EC2 deployment using a MongoDB database on a separate instance, the MongoDB instance for the registry runs in another Docker container on the same EC2 instance; they are linked together by the Docker container linking system. With each search request, using the query keyword "puzzle", the registry retrieves all Service records from

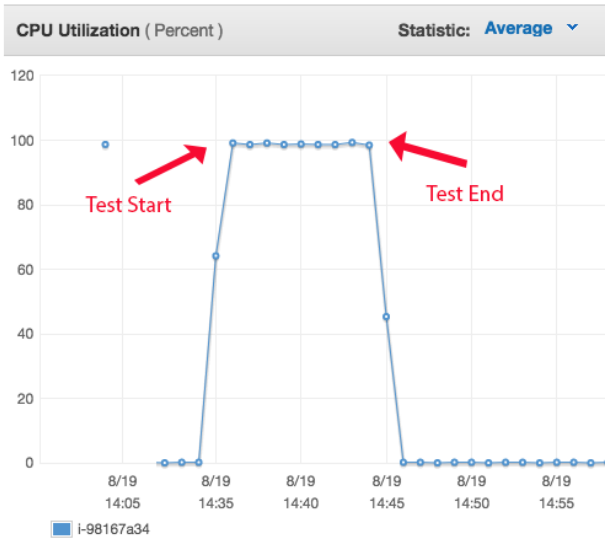


Figure 8.2: AWS CloudWatch CPU Utilisation Graph for Services EC2 Instance with 1000 threads simulated users requesting to solve the N-Queens problem for N = 10.

the MongoDB database, with a "find all" operation. The Java code itself iterates over the records for the matching puzzle service and returns it. Bear in mind, there are less than 10 records in the database for the sample services. If the database contained hundreds or thousands of Service records, optimisations would be needed, and larger overhead and resulting response time could be expected. In the test, the registry does perform the part-of-speech tagging (POS) described in Chapter 5, even on the sole search token used.

The JMeter summary report data can be seen in Table 8.4. A response time graph can be seen in Figure 8.3. The average response time is 14.3 seconds over 39742 requests, and from the response time graph, can be seen to be variable. Also shown is a median of 6.3 seconds, and a throughput of 66.1 requests completed per second over the 10 minute duration of the test. The error rate recorded over the course of the test was 1.85%. Compared with the compute intensive N-Queens puzzle, the throughput is lower, and with the longer average response time, it appears that the discovery will be a greater bottleneck in the evaluation of the CAMCS architecture, than the N-Queens puzzle.

When considering the high average response time for querying a database

Table 8.4: Metrics from JMeter while 1000 threads simulated users searching the registry for the N-Queens Game, using search keyword "puzzle".

No. of Samples	Average (ms)	Median (ms)	90% CI (ms)	95% CI (ms)	Throughput (requests/s)
39742	14356	6313	19642	33636	66.1

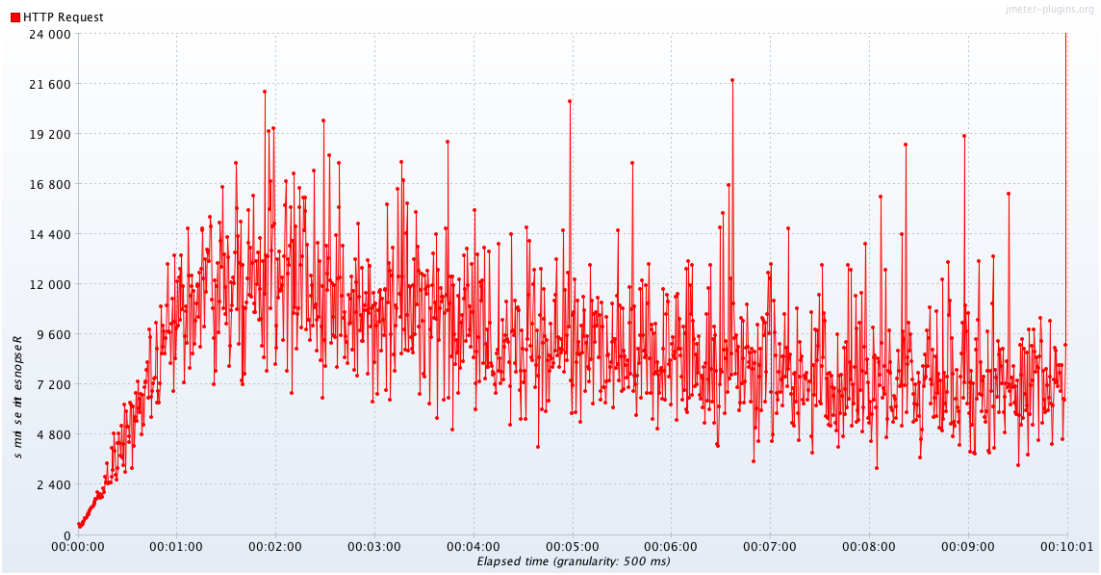


Figure 8.3: Response time graph from JMeter while 1000 threads simulated users searching the registry for the N-Queens Game, using search keyword "puzzle".

with only ten records, it should be noted that the EC2 instance is not optimised for MongoDB. The developers behind MongoDB recommend that the EC2 instance has "EBS Optimisation" enabled. EBS (Elastic Block Storage) is the root storage volume associated with the instance; it is not a form of persistent volume, and would be deleted upon instance termination. EBS Optimisation adds a channel for greater throughput for requests through to the EBS volume. The optimisation was not enabled because the option does not fall within the free tier. This, combined with the free micro type instance can explain the poor throughput and response times here.

Looking at the EC2 instance, the CPU utilisation graph from CloudWatch is shown in Figure 8.4. For the 10-minute duration of the test, the CPU usage spikes at 34.33%, but evens out to between 7% and 9%. This, combined with the poor throughput and response times noted as a result of the lack of EBS-optimisation show that the registry is not compute intensive or CPU-bound, but requires high throughput and memory performance.

In the coming sections, where the architecture is tested from the CAMCS EC2 instance, some tests will include service discovery and service consumption occurring on the respective EC2 instances, and others will not. This will be stated where appropriate. The number of simulated users chosen for the tests, 500, and 1000, were chosen randomly. Supporting more users, as the results



Figure 8.4: AWS CloudWatch CPU Utilisation Graph for Service Registry EC2 Instance with 1000 threads simulated users searching the registry for the N-Queens Game, using search keyword "puzzle".

will show (in one experiment, 10000 users was chosen), requires more powerful EC2 VMs, that are outside of the limits of the free tier of usage that was available for use, based on financial constraints on the research.

8.3.3 CAMCS - 500 Users with No Scaling

Two evaluations are presented for this case; one where CAMCS did not perform service discovery and service consumption (the two boolean parameters in the request path were set to false), and one where both operations are performed (the path parameters were set to true). Comparing each of these cases will show the timing overhead as a result of performing service discovery and consumption, as well as the time for CAMCS to do its own processing. For each case, JMeter was set to run 500 threads (simulating 500 users concurrently), sending HTTP requests to CAMCS, for an anonymous CPA to complete a task of solving the N-Queens problem for a random value of N between 3 and 8. JMeter was also set with a 60-second ramp-up period, and a uniform random timer was set to be applied before each request was sent, resulting in a random delay of a maximum of 1 second, with a constant period of 100ms.

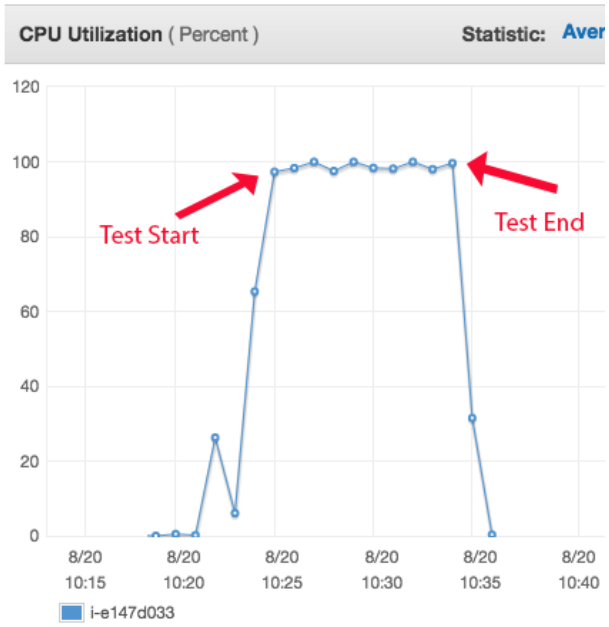


Figure 8.5: AWS CloudWatch CPU Utilisation Graph for the CAMCS EC2 Instance with 500 threads simulating users requesting anonymous CPAs to complete a task solving the N-Queens puzzle for random N between 3 and 8. No scaling took place.

JMeter metrics are presented in each case, along with CPU utilisation graphs for the EC2 instance, from AWS CloudWatch. No auto-scaling took place; requests were sent directly to the EC2 instance, rather than a load balancer.

The first test case was where service discovery and service consumption were not performed. These results show just the time time taken by the work of CAMCS itself. The first row of Table 8.5 presents metrics from by JMeter at the end of the test, having run for 10 minutes duration. The average response time for CAMCS handling 35691 requests from 500 users concurrently over the 10 minutes, was 8 seconds, and the 95% confidence interval is 9.9 seconds. A maximum throughput of 54.7 requests per second was achieved. The CPU utilisation, as seen in Figure 8.5, was 100% for the duration of the test, showing the work by CAMCS to be compute-intensive, and CPU bound.

Table 8.5: Metrics from JMeter while 500 threads simulated users requesting anonymous CPAs to complete a task solving the N-Queens puzzle for random N between 3 and 8. No scaling took place. Approximately 600ms response time overhead is present when discovery and consumption is enabled.

Discover/Consume	No. of Samples	Average (ms)	Median (ms)	90% CI (ms)	95% CI (ms)	Throughput (requests/s)
No	35691	8056	8244	9529	9932	54.7
Yes	34659	8631	8940	10234	10626	51.3

Table 8.6: Memory usage figures from the EC2 instance showing the memory usage at various points. The total memory assigned to this t2.micro instance type is 1GB.

Time	Memory Used (MB)
Before CAMCS Docker Launch	172 (17%)
After CAMCS Docker Launch	256 (25%)
During 500 User Test	434 (42%)

The second test case was where both service discovery and service consumption were both performed. The second row of Table 8.5 shows the results from JMeter for this test, again having run over 10 minutes. The average response time for CAMCS handling 34659 requests from 500 users concurrently over the 10 minutes, was 8.6ms, and the 95% confidence interval is 10.6ms. A maximum throughput of 51.3 requests per second was achieved. This shows that, as one would expect, discovery and service consumption adds overhead to the overall processing time, in this case approximately 600ms over all requests. This would change depending on optimisations on the registry (how quick it can retrieve a record if there are hundreds or thousands in the database), and the work done by the service also depends on the individual service or request (for example, in this case if larger values of N were used for the N-Queens solving task). The CPU utilisation also reached 100% for this test, but the graph has been omitted for space.

For a complete picture, CPU utilisation graphs for the registry and services EC2 instances are presented in Figures 8.6 and 8.7 respectively, gathered at the end of the test where service discovery and service consumption were enabled. These should be contrasted with the same metrics presented as results for the individual tests of these instances in the previous subsection in Figures 8.4 and 8.2. A big difference can clearly be seen. Compared with the 100% utilisation in both cases in the previous section, the CPU utilisations average off at approximately 5-7% and 3-5% for the registry and services tiers respectively, after initial spikes. The difference can likely be explained by the additional time required for the requests to reach each instance, because of the time taken by the work of CAMCS.

From these experiments, baseline performance figures of CAMCS for 500 concurrent users are taken to be approximately 8 seconds response time, with a request throughput of approximately 55 requests per second, for this low-resource, general purpose, t2.micro EC2 instance. With this result in mind,

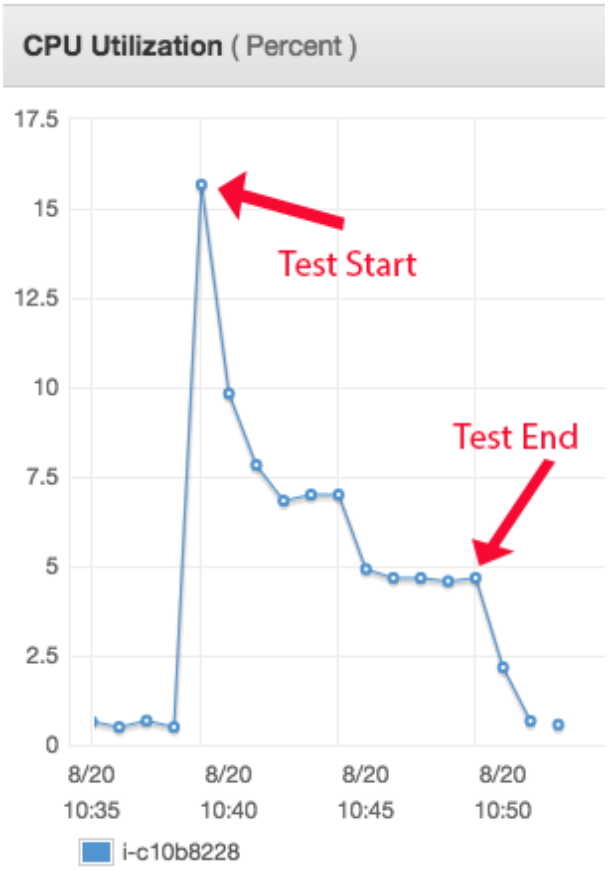


Figure 8.6: AWS CloudWatch CPU Utilisation Graph for the Registry EC2 Instance during CAMCS Test with 500 users and no scaling. An initial spike is present of approximately 16%, but this decreases during the run. Comparing with Figure 8.4, the utilisation is far lower when accessed through CAMCS.

and knowing that the CAMCS work is CPU-bound, cases with horizontal and vertical scaling are now considered, to improve on these metrics.

8.3.4 CAMCS - 500 Users with Horizontal Auto-Scaling

Three evaluations are presented for this case, and are presented in the following subsections. In all cases, this is the first set of experiments where the potential of cloud computing is used to benefit the performance of CAMCS, and the user experience.

Amazon AWS provides Auto-Scaling, whereby setting alarms, EC2 can power up new instances to cope with high demand, and then power them down again when demand has lowered. To achieve this, AWS needs three elements; a *Launch Configuration*, an *Auto-Scaling Group*, and an *Elastic Load Balancer*.

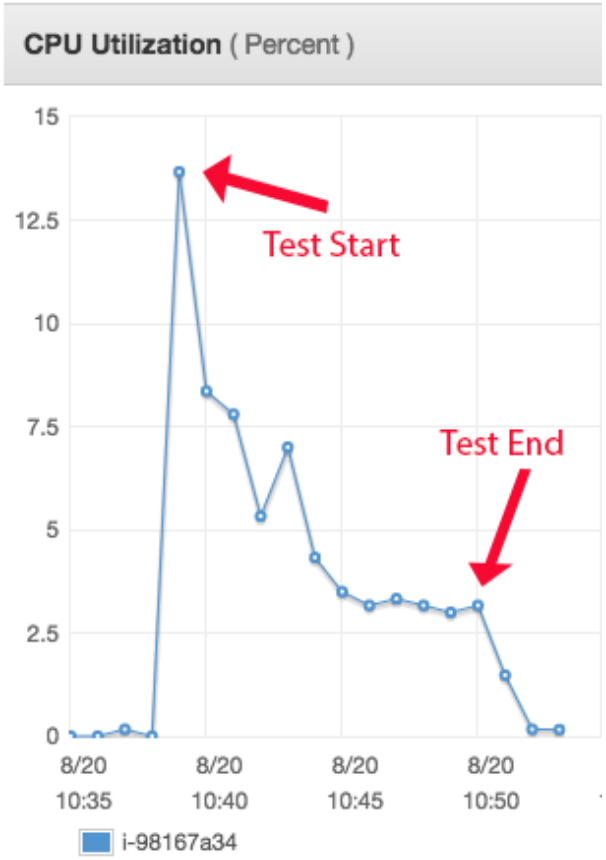


Figure 8.7: AWS CloudWatch CPU Utilisation Graph for the Services EC2 Instance during CAMCS Test with 500 users and no scaling. An initial spike is present of approximately 13%, but this decreases during the run. Comparing with Figure 8.2, the utilisation is far lower when accessed through CAMCS.

Launch Configuration - To be used with an auto-scaling group. The configuration is used to configure how new EC2 instances should be launched, such as specifying the AMI to use, and what kind of instance the AMI should be deployed on. For this experiment, the custom AMI running the CAMCS middleware with Docker was used, and *user data* was specified to instruct the instance to startup the CAMCS Docker container at launch time. The free tier standard t2.micro instance was selected as the instance type.

Auto-Scaling Group - An auto-scaling group consists of zero to many EC2 instances. It is configured with CloudWatch *alarms*. When an alarm is activated, a *policy* is used to determine what actions the group should take. The policies used in each of the following experiments will be detailed where appropriate.

Elastic Load Balancer - The elastic load balancer is associated with the auto-

scaling group, and has a public DNS address (A-record). Any traffic sent to this address (which all JMeter traffic was for these experiments), would be evenly distributed among all the EC2 instances in the auto-scaling group.

The auto-scaling group has a minimum capacity, a maximum capacity, a current capacity, and a desired capacity, where each capacity measure equates to a number of EC2 instances. For these experiments, an initial desired and minimum capacity was set to 1, and maximum was 3. The desired capacity changes over the course of the experiment running time, based on what alarms are being fired.

The load balancer uses a default grace period of 5 minutes for a new EC2 instance to startup and become healthy, before it begins forwarding traffic to that instance. Healthiness is determined by passing 10 consecutive health checks by default; a health check is just a successful TCP connection to an open port, 8080, which Docker had exposed for CAMCS connections. If the health checks are passed, the Load Balancer flags the instance as "InService". Failing a health check leaves the instance running, but it is then flagged as "OutOfService". *Connection Draining* of 5 minutes is also enabled; this allows traffic currently routed to an instance flagged for termination (due to the capacity decrease alarm and policy) 5 minutes to complete work, before the load balancer terminates that instance. Any traffic still connected to the instance after the 5 minutes is forcibly closed.

The use of more VMs to service traffic demand is known as horizontal scaling. If the one initial EC2 instance has maximum CPU utilisation, resulting in higher average response times and latency, running further instances with CAMCS should bring the average response time and latency down, along with the CPU utilisation on each instance.

8.3.4.1 First Evaluation

For the first evaluation using auto-scaling, the following two policies were used; one to increase group capacity by launching 1 new instance each time the average CPU utilisation of an instance was greater than or equal to 70% percent for a continuous period of 5 minutes (the first alarm). The second policy was to terminate 1 EC2 instance when the average CPU utilisation dropped to 50% or lower for a continuous period of 5 minutes (the second alarm). Ser-

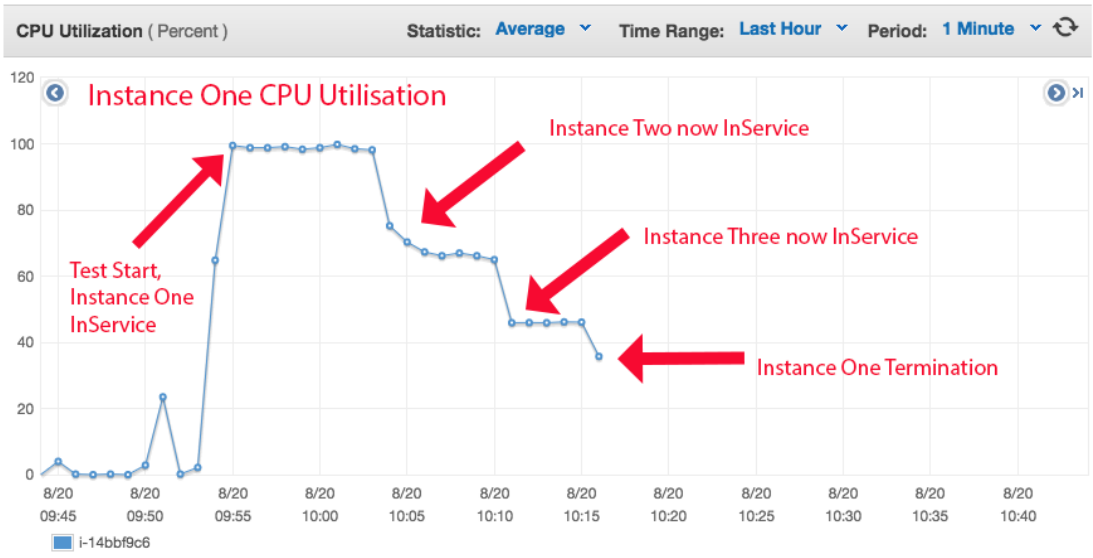


Figure 8.8: AWS CloudWatch Monitoring CPU utilisation for CAMCS EC2 Instance One, with 500 users, horizontal auto-scaling, taken at the end of the first evaluation. Red arrows indicate events during the test.

vice discovery and consumption were set both set to false. JMeter was started with 500 users and a ramp-up period of 60 seconds.

JMeter was started with 500 users and a ramp-up period of 60 seconds. During these tests, four separate EC2 instances were used; JMeter metrics captured at various times during the test as the four instances started and terminated, are shown in Table 8.7. The CPU utilisation graphs for each instance, captured at the end of the experiment are shown in Figures 8.8, 8.9, 8.10, and 8.11 respectively.

Having started JMeter running the test, the CPU utilisation of instance one jumped immediately to 100%. This action triggered the scale-up policy after the first five minutes, and a second instance was automatically launched. At the ten-minute point, the grace period for the warm up of instance two

Table 8.7: Metrics from JMeter while 500 threads simulated users requesting anonymous CPAs to complete a task solving the N-Queens puzzle for random N between 3 and 8. This is the first evaluation of auto-scaling. Shows metrics captured at various times during the test, showing the differences as instances were started and terminated.

Instances InService	No. of Samples	Average (ms)	Median (ms)	90% CI (ms)	95% CI (ms)	Throughput (requests/s)
One	25244	7895	8202	9478	9848	54.4
One & Two	84380	6067	3425	9493	12214	71.1
Two & Three	124161	5795	1816	9510	14783	75.3
None, End	226617	5512	1410	9396	17147	80.8

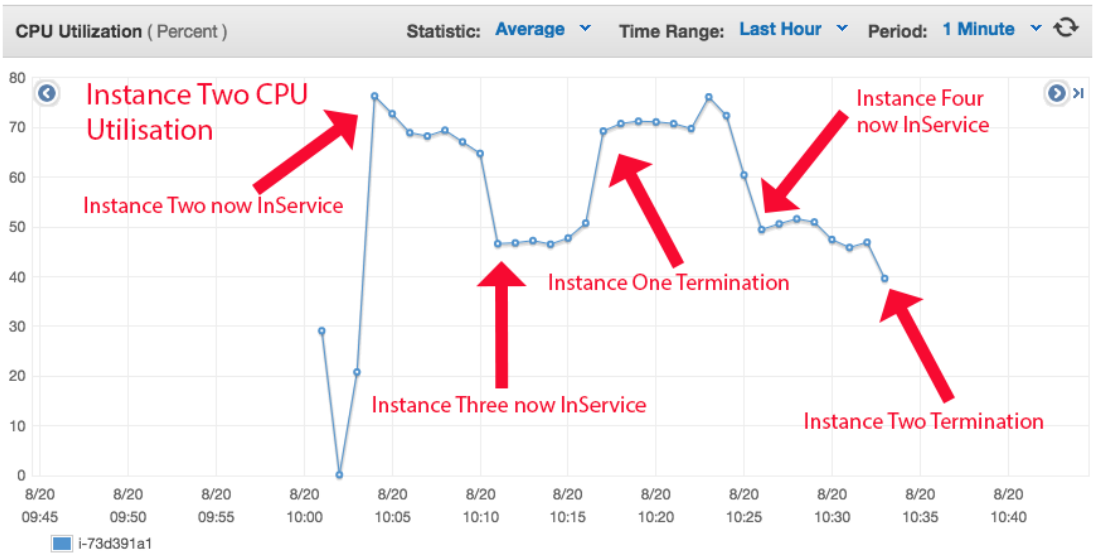


Figure 8.9: AWS CloudWatch Monitoring CPU utilisation for CAMCS EC2 Instance Two, with 500 users, horizontal auto-scaling, taken at the end of the first evaluation. Red arrows indicate events during the test.

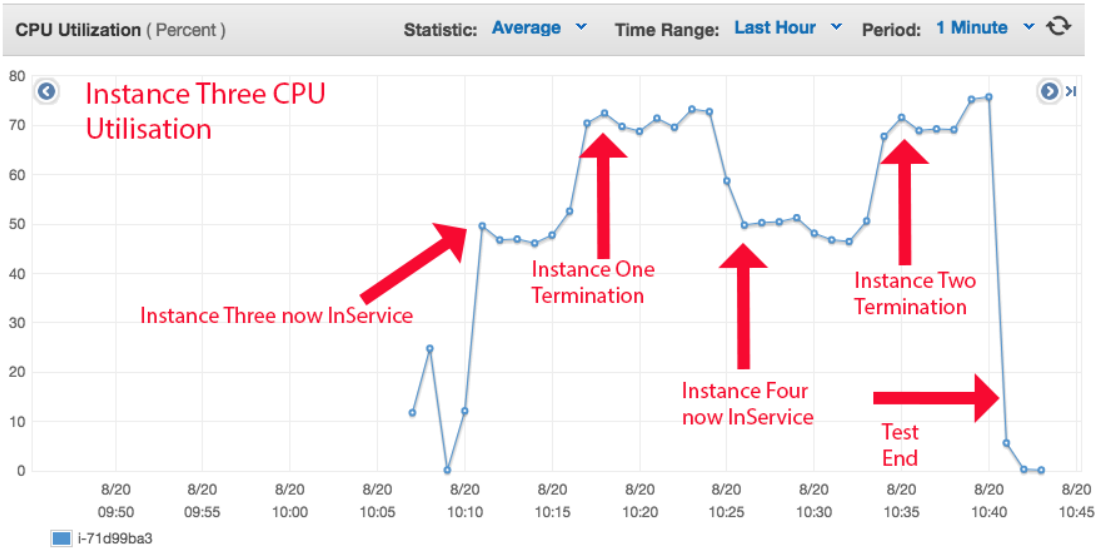


Figure 8.10: AWS CloudWatch Monitoring CPU utilisation for CAMCS EC2 Instance Three, with 500 users, horizontal auto-scaling, taken at the end of the first evaluation. Red arrows indicate events during the test.

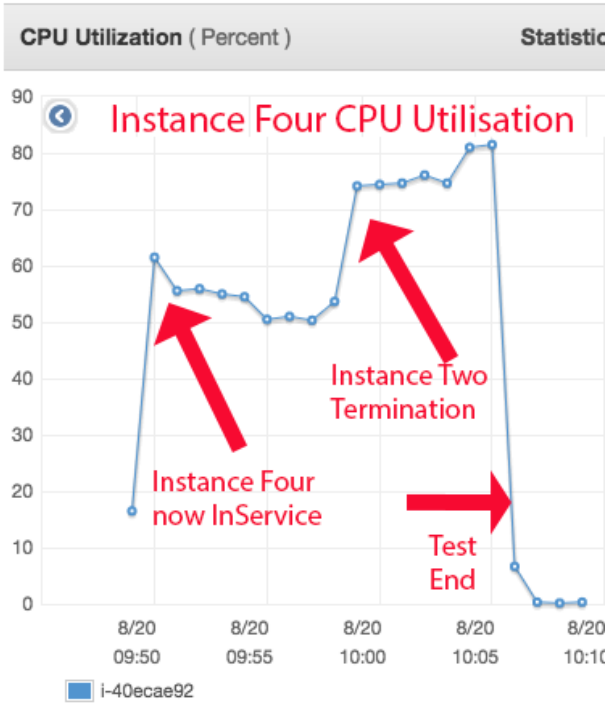


Figure 8.11: AWS CloudWatch Monitoring CPU utilisation for CAMCS EC2 Instance Four, with 500 users, horizontal auto-scaling, taken at the end of the first evaluation. Red arrows indicate events during the test.

elapsed, and the load balancer began forwarding traffic to the new instance. Metrics were recorded by JMeter just before the load balancer flagged the second instance as InService and began traffic forwarding to it. This shows an average of 7.9 seconds, and a throughput of 54.4 requests per second.

The CPU utilisation of instance one remained at 100% for the duration of the time it took instance two to startup (about 3-4 minutes), before dropping to approximately 77% after 5 minutes. However, this resulted in the scale-up alarm firing a second time for instance one, which prompts auto-scaling to now launch a third instance. While instance three is starting up, the CPU utilisation on instance one drops to approximately 65%, while the CPU utilisation on instance two varies between 67-70%, as the load balancer is now using both instances.

Instance three is flagged as InService by the load balancer at the 15 minute time point of the test, and the load balancer begins forwarding traffic to it. JMeter metrics were captured just before the third instance came into service. Compared with the metrics taken from JMeter before instance two became InService, it can be seen that the result of having two instances running CAMCS has reduced the average response time to 6 seconds, and the throughput has

increased to 71.1 requests per second.

With three instances now in service, the CPU utilisation drops to approximately 45% on instance one, and 47-48% on instance two. Instance three CPU utilisation settles at 45-46%. The three instances are now under-utilised. The result of each instance's average CPU utilisation falling below 50% for a continuous five minutes, fires the scale-down alarm, and AWS auto-scaling terminates instance one at the 20-minute time point, bringing it OutOfService on the load balancer. This forces the CPU utilisation to increase to approximately 70-72% on instance two, and 70-74% on instance three. This once again fires off the scale-up alarm, and a fourth instance is now launched. This becomes InService at the 30-minute time point.

JMeter metrics were captured just before the fourth instance came into service. Average response time has continued to drop to 5.7 seconds, and throughput has continued to increase to 75.3 requests per second, despite the fact that only two instances are in service. This can be explained by the CPU bursting capacity of the t1.micro instance. This is governed by a daily allowance of CPU credits, and as shown in Figure 8.12, using instance three as an example, the credit balance continued to drop for all instances during the test. Once the credit balance reaches zero, the CPU bursting cannot take place until the next day.

Once instance four comes into service, CPU utilisation on instances two and three drops back again to approximately between 47-52%. Instance four CPU utilisation settles into use at between 50-55%. The scale-down alarm is fired once again, and auto-scaling terminates instance two close to the 40-minute time point. As expected, this increases the CPU utilisation on instances three and four to between approximately 70-75%, and 75-80% respectively. This would have fired the scale-up alarm again, but the test is ended after 45-minutes, before this can happen. The JMeter metrics captured at the end of the test shows average response time has reduced to 5.5 seconds, and throughput has increased to 80.8 requests per second.

Figure 8.13 shows the number of HTTP 200 OK responses from CloudWatch monitoring on the load balancer for the duration of the test. With solely instance one InService at the start of the test, this is approximately 3300. When the instance two becomes InService, this increases to between 5000 and 5500 for the remainder of the test, showing the benefits of scaling. Figure 8.14 shows the average latency graph from CloudWatch, for the load balancer. With just

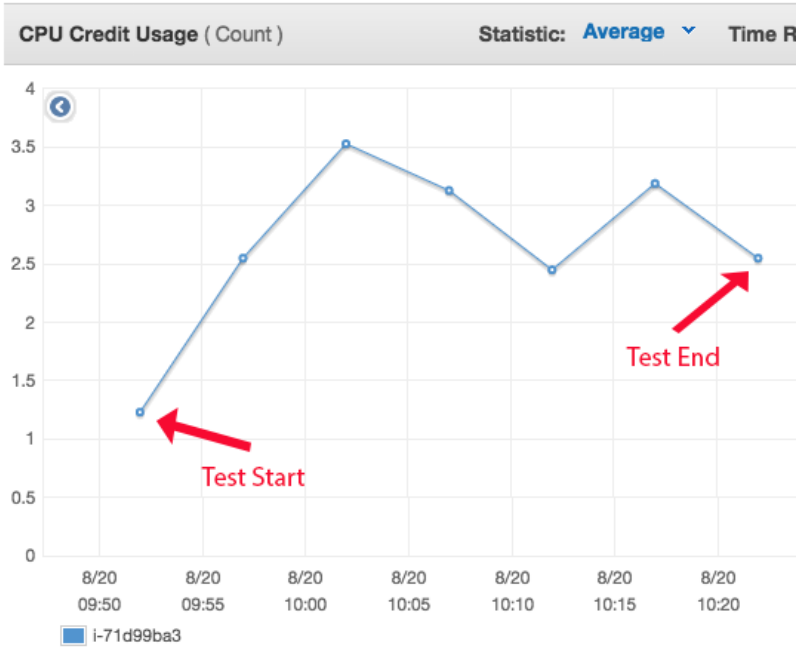


Figure 8.12: AWS CloudWatch Monitoring CPU credit balance captured from Instance Three during the first evaluation with horizontal auto-scaling. This shows the credit balance reducing as the test proceeds; this indicates that the capacity turbo boost to 3.3GHz of the vCPU was used during the test. Once this value reaches zero, the boost capacity can no longer be used. Credits are re-charged on a 24-hour basis.

instance one in operation at the start of the test, this is approximately 8 seconds. When instance two becomes InServices, this dramatically decreases to 50 milliseconds.

The results of this first evaluation show that horizontal scaling is beneficial to the performance of CAMCS in terms of response time and throughput, but having three instances in operation for this evaluation, does not produce a large increase in performance over two instances. For the second evaluation following, the auto-scaling policies are changed, based on the results from this evaluation.

8.3.4.2 Second Evaluation - Change of Scaling Policy

For the second evaluation, a modification was made to the policy for scaling up; increase group capacity by launching 1 new instance each time the average CPU utilisation of an instance was greater than or equal to 85% percent for a continuous period of 5 minutes (the first alarm). The second policy for scaling down was the same as that used in the first evaluation. The same JMeter set-

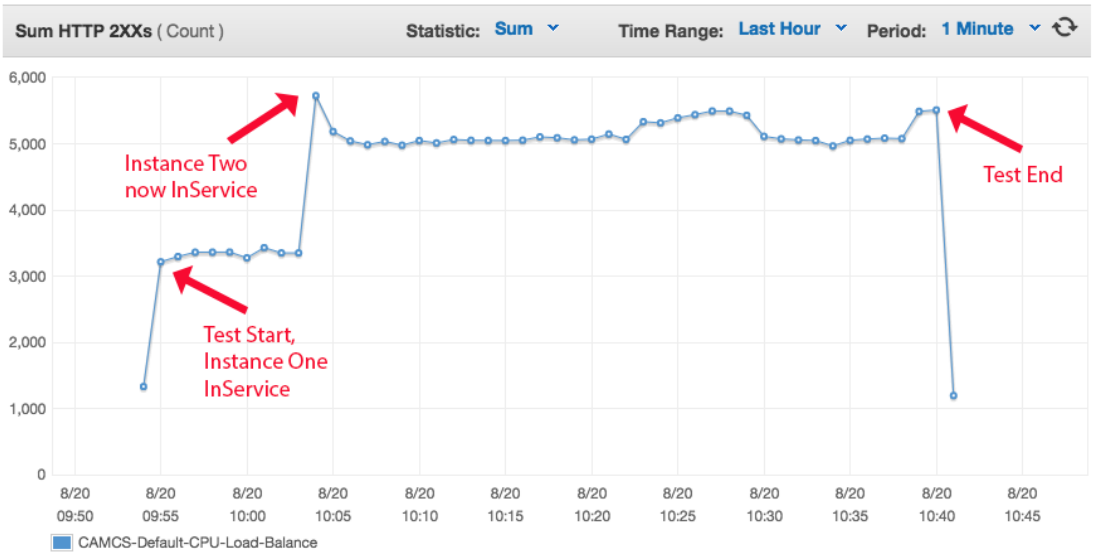


Figure 8.13: AWS CloudWatch Load Balancer monitoring for the number of successful HTTP 200 OK requests recorded each minute, during the first evaluation with 500 simulated users, and horizontal auto-scaling. The use of two instances improves the number of requests to between 5000 and 5500, but the use of three instances at once does not show any considerable improvement.

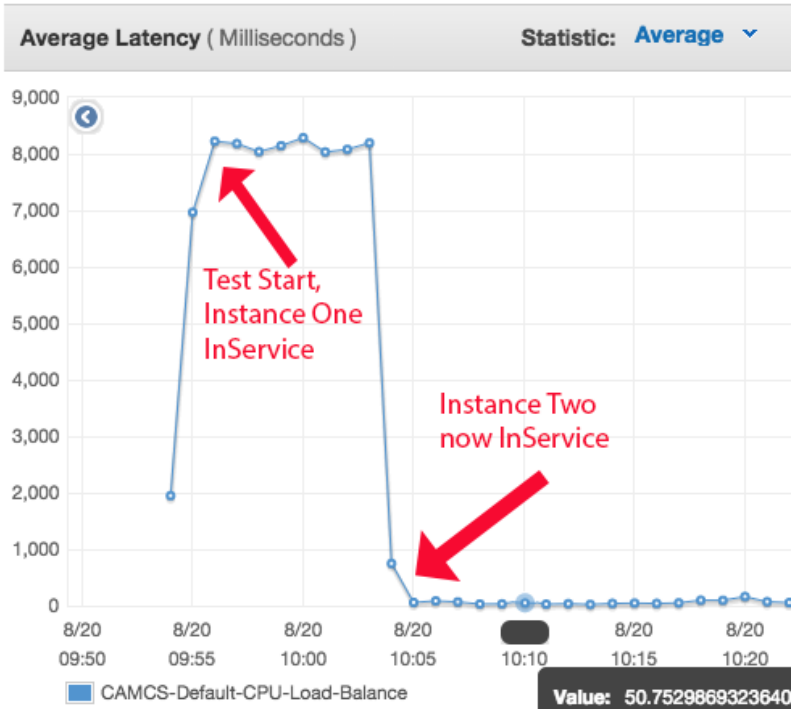


Figure 8.14: AWS CloudWatch Load Balancer monitoring for average latency recorded each minute, during the first evaluation with 500 simulated users, and horizontal auto-scaling. The use of two instances reduces the latency from 8 seconds to 50 milliseconds, but the use of three instances does not show any considerable improvement.

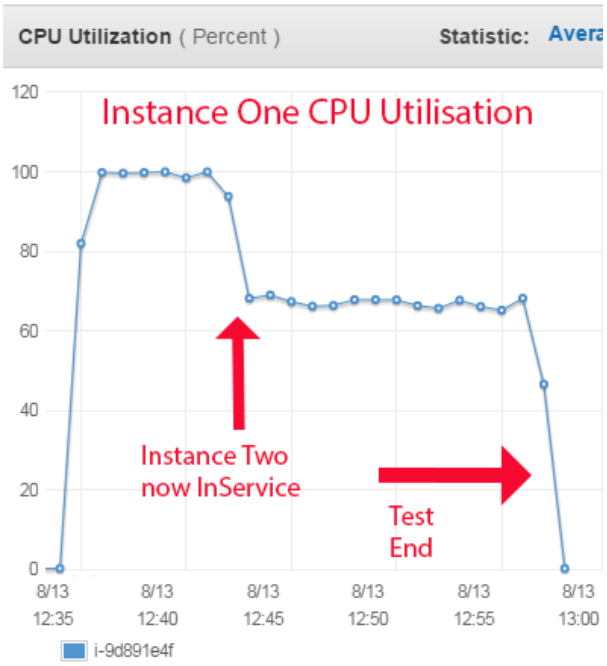


Figure 8.15: AWS CloudWatch CPU Utilisation Graph for EC2 Instance One, taken at the end of the second evaluation, showing the initial average CPU utilisation of 100% followed by a decreased to approximately 68 - 70% after the second instance became InService.

tings were used as in the first evaluation. Service discovery and consumption are still disabled.

Figure 8.15 shows the CPU utilisation of the initial EC2 instance at the end of the test. CPU utilisation immediately reached 100% within the first minute of starting. As it remained above 85% for a continuous period of 5 minutes, at the 5-minute mark, the auto-scaling group launches a second instance within the group. The 5-minute grace period is applied before traffic is routed to this instance. The average CPU utilisation for the second instance is shown in Figure 8.16, captured at the end of the test. When instance two is flagged InService by the load balancer, the CPU utilisation on instance one drops to about 78%, and this is where it remains for the duration of the experiment. The CPU utilisation of instance two varies between 70-80% for the duration of the experiment after it is launched. At this point, as neither instance rises above CPU utilisation of 85% again, the third instance that is permitted under the auto-scaling group, is not required, and never launches.

Table 8.8 shows a sample of the output from JMeter that was captured while the experiment was running, just before instance two became InService, and at the end of the test when both instances were InService. Notably, when just

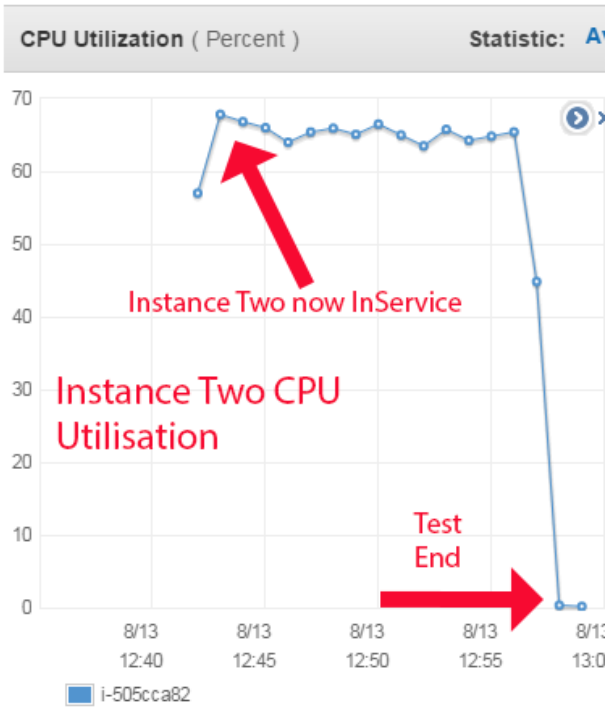


Figure 8.16: AWS CloudWatch CPU Utilisation Graph for EC2 Instance Two, taken at the end of the second evaluation, showing the average CPU utilisation of approximately 65%.

instance one was InService, the average response time at that point over 23745 requests was 7.3 seconds, with a throughout of 57.9 requests per second, similar to the first evaluation. The metrics show that at the end of the test when both instances were in service, that for over 98267 requests made, the average response time has dropped to 5.4 seconds, and the throughput has increased to 77.8 requests per second, very similar to the figures recorded at the end of the first evaluation.

It is very important to understand that in normal operation, no mobile user will be waiting in a blocked state for 5-8 seconds with the CAMCS Client, waiting for a task to complete. The experiments are conducted in a synchronous, blocking fashion, while the CAMCS model is asynchronous, and non-blocking. Once the mobile user has sent the required parameters for a task to his/her CPA, the task will then begin execution. When the result is ready, the user is notified; tapping the notification to view the result HTML page, the user will wait only for a few seconds while loading the result page, similar to loading any other webpage on his/her device browser. This will be even more evident for previously completed tasks that are re-run, and for automatic tasks. These two task models are expected to be the most common means of users

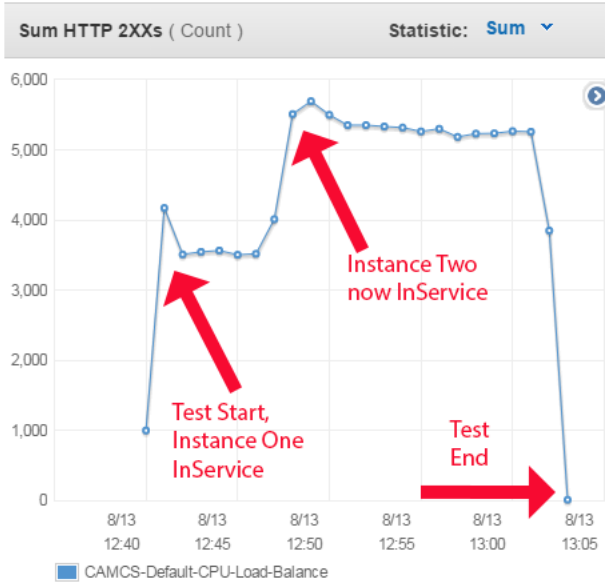


Figure 8.17: AWS CloudWatch Load Balancer monitoring for the number of successful HTTP 200 OK requests recorded every minute, both before, and after the second instance becomes InService. This is with 500 users, with service discovery and consumption not taking place.

completing tasks with CPAs, rather than creating new tasks frequently.

Figure 8.17 from the CloudWatch monitoring on the load balancer shows how the number of successful requests (HTTP 200 OK) that it handled increased when the second instance came online. Figure 8.18 shows from CloudWatch monitoring on the load balancer, the average latency for the duration of the test. Before the second instance became InService, this was approximately 7.8 seconds. After the second instance became InService, this dropped to approximately 0.27ms (which, not visible in the figure, was retrieved from mousing over the 1-minute interval points in the graph). These graphs show almost identical behaviour as with the first evaluation for two instances using the modified scale-up policy.

The results of the second evaluation show that, for this test with with 500 users,

Table 8.8: Metrics from JMeter while 500 threads simulated users requesting to complete an N-Queens puzzle for N randomly generated between 3 and 8. Service Discovery and consumption did not take place. This is the second evaluation of auto-scaling. Shows metrics captured at various times during the test, showing the differences as instances were started and terminated.

Instances InService	No. of Samples	Average (ms)	Median (ms)	90% CI (ms)	95% CI (ms)	Throughput (requests/s)
One	23745	7327	7705	8953	9330	57.9
One & Two	98267	5463	1716	8898	12716	77.8

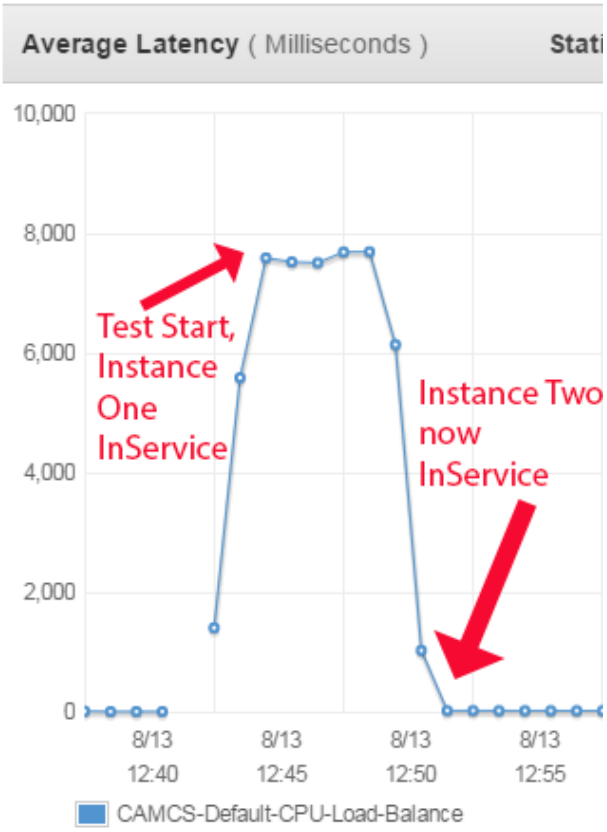


Figure 8.18: AWS CloudWatch Load Balancer monitoring for average latency recorded each minute, both before, and after the second instance becomes In-Service. This is with 500 users, with service discovery and consumption not taking place.

the use of two cloud instances delivers equivalent performance to using three instances, in terms of average response time and throughput. The modified scale-up policy, with 85% as the baseline CPU utilisation metric, shows the importance of testing and observing auto-scaling behaviour, and adjusting the policies for more effective resource utilisation; the CPUs of each instance are both utilised 70% or more, rather than being under-utilised as in the previous evaluation.

8.3.5 CAMCS - 1000 Users with Horizontal Auto-Scaling

In this subsection, the results of an experiment to increase the number of users are presented, from 500 simulated users to 1000. The same settings were used for JMeter as in the experiments with 500 users and horizontal scaling. Service discovery and consumption did not take place.

After an initial test using the same scaling policies as with the first evaluation of 500 users and horizontal auto-scaling, the same CPU utilisation patterns for three instances launched during the course of the test were found (graphs omitted for space). Three instances were launched because the second instance did not become flagged as InService by the load balancer until over 5 minutes after the instance created, forcing the scale-up policy to fire a second time. With three instances, under-utilisation occurred, resulting in one instance being shut down, increasing utilisation on the remaining two, forcing a third instance to start a second time. JMeter metrics did not show any significant improvement with the use of three instances against two.

For the second test, the auto-scaling group settings were changed to allow a maximum instance count of 2. This would give the second instance a longer time to start up, without a third instance being launched, resulting in better CPU utilisation. Figures 8.19 and 8.20 show the CPU utilisation for both instances captured at the end of the test, showing that after the initial 100% CPU utilisation on the first instance, both instances achieved utilisation of approximately 75% for the duration of the test when they were both InService.

Table 8.9 shows the JMeter metrics captured at various points during the test. With 1000 users, before the second instance became InService, with 34650 sample requests, average response time was 16.3 seconds, with throughput at 55.4 requests per second. The response time, as expected, is twice that of the response time for 500 users, although with comparable throughput. The other metrics captured at the end of the test show with two instances in service, average response time dropped to 11.3 seconds over 146444 sample requests made during the test, and the throughput increased to 78.1 requests per second. It is important to again emphasise, that in normal operation, no mobile user will be waiting in a blocked state for between 11 and 18 seconds with the CAMCS Client, waiting for a task to complete; the model is asynchronous.

The metrics also show a decrease in the median response time from 17.1 sec-

Table 8.9: Metrics from JMeter while 1000 threads simulated users requesting anonymous CPAs to complete a task solving the N-Queens puzzle for random N between 3 and 8. Shows metrics captured at various times during the test, showing the difference between just before the second instance became InService, to just before the test was terminated with the two instances InService.

Instances InService	No. of Samples	Average (ms)	Median (ms)	90% CI (ms)	95% CI (ms)	Throughput (requests/s)
One	34650	16319	17177	18442	18849	55.4
One & Two	146444	11366	2698	18350	30001	78.1

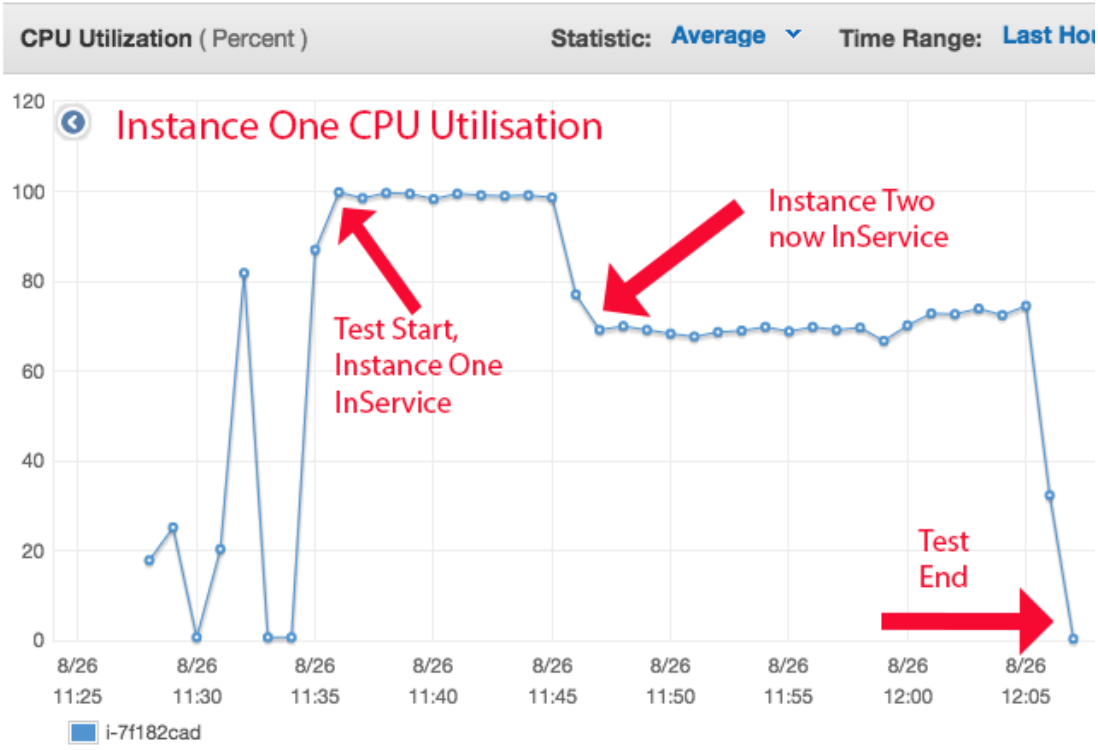


Figure 8.19: AWS CloudWatch CPU Utilisation Graph for EC2 Instance One, with 1000 users and horizontal auto-scaling, captured at the end of the experiment, showing the initial average CPU utilisation of 100% followed by a decrease to approximately 70% after the second instance became InService.

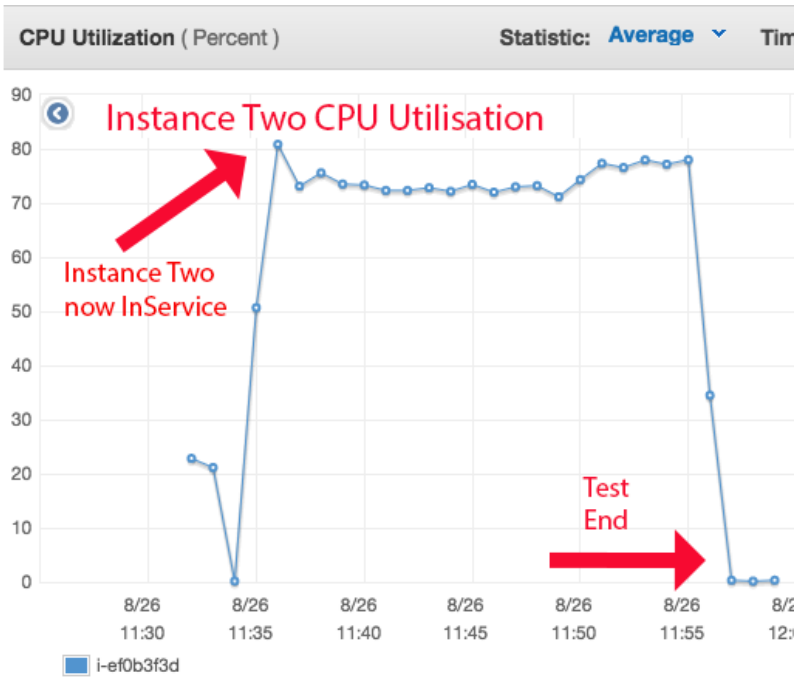


Figure 8.20: AWS CloudWatch CPU Utilisation Graph for EC2 instance two, with 1000 users and horizontal auto-scaling captured at the end of the experiment, showing the average CPU utilisation of 72-77%.

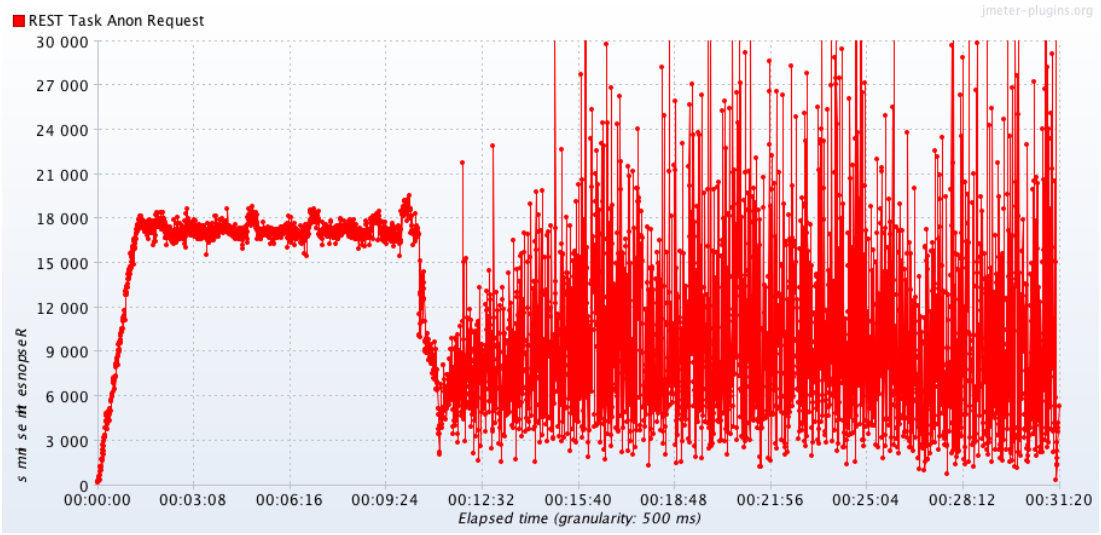


Figure 8.21: Response time graph from JMeter while 1000 threads simulated users requesting anonymous CPAs to complete a task solving the N-Queens puzzle for random N between 3 and 8. The drop in average response time from 16-17 seconds to 9-11 seconds can be seen. The highly variable response times after the Load Balancer flags the second instance as InService can be seen.

onds to 2.7 seconds, while the 95% confidence interval increased from 18.8 seconds to 30 seconds. This trend was also present in the tests for horizontal auto-scaling with 500 users. These figures can be explained by examining Figure 8.21, a JMeter response time graph. With one instance, the graph shows an average response time of between 17 and 18 seconds, by the time instance two becomes in service. Once instance two comes into service at the 10-minute point, the response times become far more erratic. This is a result of the load balancer. JMeter showed a request error rate of 2% at the end of the test. This came from requests that ended with HTTP errors such as "connection reset", "connected timed out", or "the target server failed to respond". Note that because of the default JMeter settings, any-failed requests were not re-tried. With no load-balancer in use, running the same test with 1000 users on only one EC2 server, these errors do not appear, and the error rate remains at 0%. It is unclear what causes these messages; overhead would almost certainly be introduced as a decision is made on which instance should take the request. Although the load-balancer documentation stresses that more running instances forces the load-balancer to work faster.

In line with the previous evaluations, Figures 8.22 and 8.23 show the metric graphs from CloudWatch for the number of HTTP 200 OK requests, and load-balancer latency during the course of the test respectively. Similar results

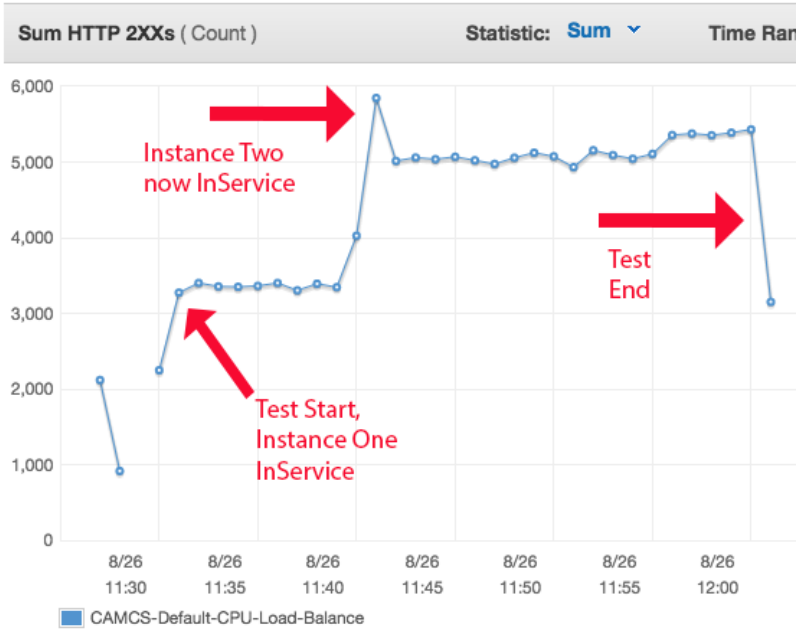


Figure 8.22: AWS CloudWatch Load Balancer monitoring for the number of successful HTTP 200 OK requests recorded every minute, both before, and after the second instance becomes InService. This is with 1000 users.

are discovered with the use of two instances compared to one, more requests are handled, and latency reduces. The results from this evaluation show that CAMCS can continue to handle a high volume of growing users, although longer response times again show the value of using the appropriate number of instances, fine-tuned scaling policies, and the right instance type for the compute intensive operations (which was not the case here with the t2.micro instance used).

8.3.6 CAMCS - 500 Users with Vertical Scaling

So far, it has been shown that horizontal scaling by increasing the number of EC2 instances running CAMCS is beneficial in terms of reduced response times, and higher throughput. However, the results have also shown that the work by CAMCS is compute intensive; the free-tier eligible t2.micro general-purpose instance is not suitable for this kind of demand. Ideally, CAMCS should be deployed on an AWS instance that is categorised as being *compute-optimised*, with EBS optimisation as well. For this experiment, an AWS instance is used that meets this criteria; a c4.xlarge instance is utilised to repeat the test with 500 simulated users with no horizontal scaling. Service discovery and consumption are also disabled. The specifications for the c4.xlarge instance

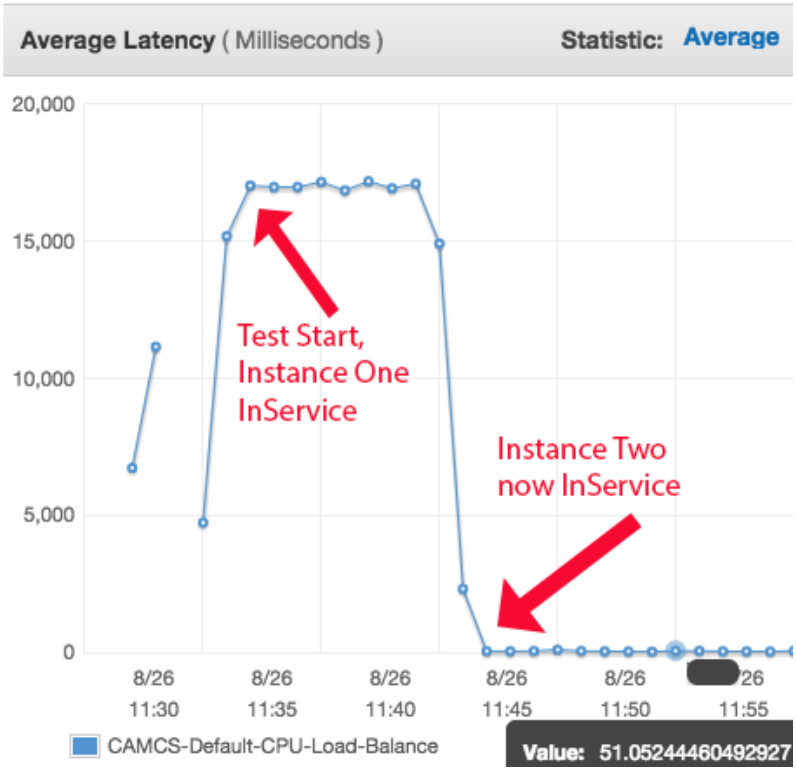


Figure 8.23: AWS CloudWatch Load Balancer monitoring for average latency recorded each minute, both before, and after the second instance becomes InService. This is with 1000 users.

can be found in Table 8.10.

The JMeter metrics are presented in Table 8.11, and the CPU utilisation is presented in Figure 8.24. The metrics gathered from one instance of this type are comparable to those obtained with horizontal-scaling the t2.micro general-purpose instances, with an average response time of 5.3 seconds over the 74985 requests made, during the 15-minute duration of the test. CPU utilisation can be seen in Figure 8.24. This shows that on this instance type, the CPU is under-loaded, peaking at approximately 32%. This can be compared with using one t2.micro instance, where the utilisation reaches 100%.

These results show that vertical scaling can also be effective at reducing response times and the throughput of CAMCS, and that a compute and EBS optimised instance is the desirable instance type. However, the pricing of this type of instance is high, and may not be cost effective to scale horizontally, given the under-utilisation of the CPU of the sole instance.

Table 8.10: AWS c4.xlarge Instance Specifications, taken from AWS website. This instance is optimised for compute-intensive operations.

Feature	Value
vCPU	4 (High frequency Intel Xeon E5-2666 v3 (Haswell) processors optimised specifically for EC2)
Memory	7.5GB
Dedicated EBS Throughput	200Mbps
Network Performance	High
EBS-Optimised Available	Yes
Category	Compute-Optimised

Table 8.11: Metrics from JMeter while 500 threads simulated users requesting to complete an N-Queens puzzle for N randomly generated between 3 and 8. Service Discovery and consumption do not take place. This demonstrates vertical scaling on one large, compute and EBS optimised EC2 instance type, c4.xlarge. Results are comparable to those obtained horizontal-scaling the general-purpose t2.micro instances.

No. of Samples	Average (ms)	Median (ms)	90% CI (ms)	95% CI (ms)	Throughput (requests/s)
74985	5385	10138	19979	13150	75.9

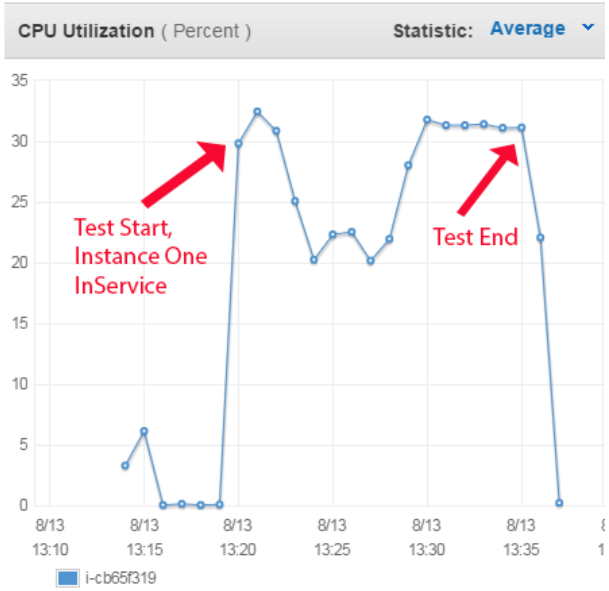


Figure 8.24: AWS CloudWatch CPU Utilisation Graph for CAMCS running on one "compute and EBS optimised" c4.xlarge instance, showing an under-utilised CPU, peaking at 32%, for 500 simulated users, with service discovery and consumption disabled.

8.3.7 CAMCS - 10000 Users with Horizontal Auto-Scaling

For a final test, to evaluate how CAMCS would scale under a high-load, a test was created where JMeter would simulate 10000 users sending requests to CAMCS with a task to solve the N-Queens problem. The JMeter test was setup to use a 5-minute ramp-up period for this experiment. The free tier t2.micro instances were specified for use.

A new auto-scaling group was used. This group has a minimum and desired EC2 instance count of 3, and a maximum instance count of 10.

Unfortunately, because of the default limits of the Tomcat servlet container shown in Table 8.1, the test quickly overwhelmed CAMCS on the three instances. JMeter was only able to simulate approximately 2500 users, before the error rate recorded during the test reached upwards of 70% failed requests. The test also proved to be too demanding for the default t2.micro instances, also becoming overwhelmed. At the time the test was cancelled, the CPU utilisation of the three EC2 instances dropped to below 20%. Increasing the thread-pool size of Tomcat is not recommended, as the average response time for each individual request would be expected to grow as a result of context-switching in the CPU for each request, decreasing performance further.

This evaluation was not continued. However, the experiment further demonstrated that the only feasible way to cope with this burst of high-demand traffic, in respect of this many users, is to use appropriate instances, such as the compute-optimised and EBS-optimised instances, as the capacity of Tomcat by itself, cannot be so easily increased. Future work can address this, when more resources are available for running CAMCS on suitable compute-infrastructure with the required capacity for such demand. Additionally, code profiling can be applied to the CAMCS code-base, to identify overheads present in the code, such as expensive objects and memory leaks; such problems can be corrected for optimisation. The CAMCS version used in these experiments, was not optimised with such a technique.

8.4 Mobile Device Performance Evaluations with the CAMCS Client

Unlike other mobile cloud computing solutions, the aim of this thesis project, in light of the user experience requirements outlined in Chapter 3, is for most of the work to take place within CAMCS running on cloud infrastructure, leaving the mobile device with little responsibility. Evaluation on the mobile device takes place with the CAMCS Client application. These experiments take place on a Google Nexus 5 device, running Android 4.4.4 KitKat. Unlike the experiments for CAMCS on the cloud, where changes were made to make the operations (such as discovery and service consumption) synchronous and blocking, all of the operations on the CAMCS Client are maintained asynchronous for this evaluation, as this is how the mobile user would interact with CAMCS.

8.4.1 Initial Installation Resource Usage

Figure 8.25 shows the Android application details for the CAMCS Client, just after it has been installed. The application APK installation takes up 13.5MB of the device storage. 4.3MB of this comes from the JAR files bundled with the application for the different required libraries, which were the subject of Chapter 7. A memory consumption graph from the Android Studio IDE, just after the application is launched for the first time, shows a memory consumption of 16.98MB out of 17.29MB allocated (omitted for space).

8.4.2 Resource Usage For Task Work

To evaluate an example of using the CAMCS Client to perform task work, a new task was created for fetching calendar data from a calendar, with the intent of using the Google Calendar service described in Chapter 5. This test was performed twice; once on a Wi-Fi connection, and once on a 3G cellular connection.

Fig 8.26 shows the battery power consumption and data transfer graphs, captured with the Trepn Profiler [106], for the complete work associated with completing the task, including initial task creation, service selection from discovered services, data parameter input, and result retrieval, over the same Wi-Fi

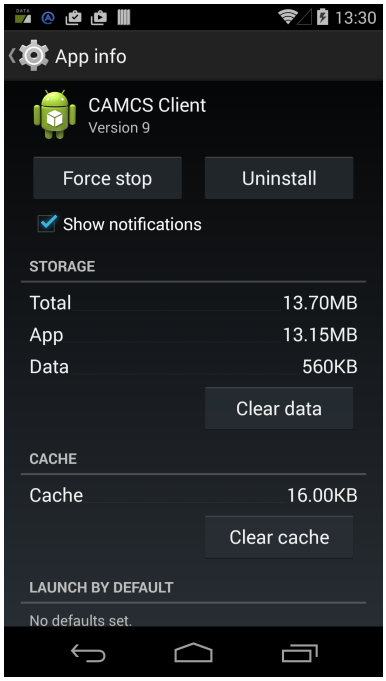


Figure 8.25: Readout from the Android OS application Settings menu on the Nexus 5 device for the CAMCS Client application. The APK storage required is only 13.5MB.

network used for the AWS experiments. Fig 8.27 shows a screen capture from the Trepn Profiler Android application on the Nexus 5 mobile device, which lists statistics recorded for various metrics during the capture.

Almost identical to the results presented from the evaluation in Chapter 6, a base-line power consumption of 1.5W is seen for the duration of the work. Spikes were present each time network activity occurred, for the most part up towards the 3W point, with others spiking above this to 5W. The network activity graph at the bottom is marked to show the events corresponding to the different actions completed during the test. Correlating this with the metrics present in the Trepn Profiler Android application, a total of 4.59KB of data was sent to the CPA, while 377.70KB was received by the client, from the CPA. By far, most of the 377KB of data received is from the result HTML page being fetched and displayed. Hovering over the Trepn Profiler graph in the Eclipse IDE, it shows that the size of the result HTML file for this individual's Google calendar, was 372.37KB.

Fig 8.29 shows the battery power consumption and data transfer graphs, captured with the Trepn Profiler, for the same test, but performed over the cellular 3G network (Vodafone Ireland as the MNO). The results are for the most part identical to those recorded during the test over the Wi-Fi network. In some

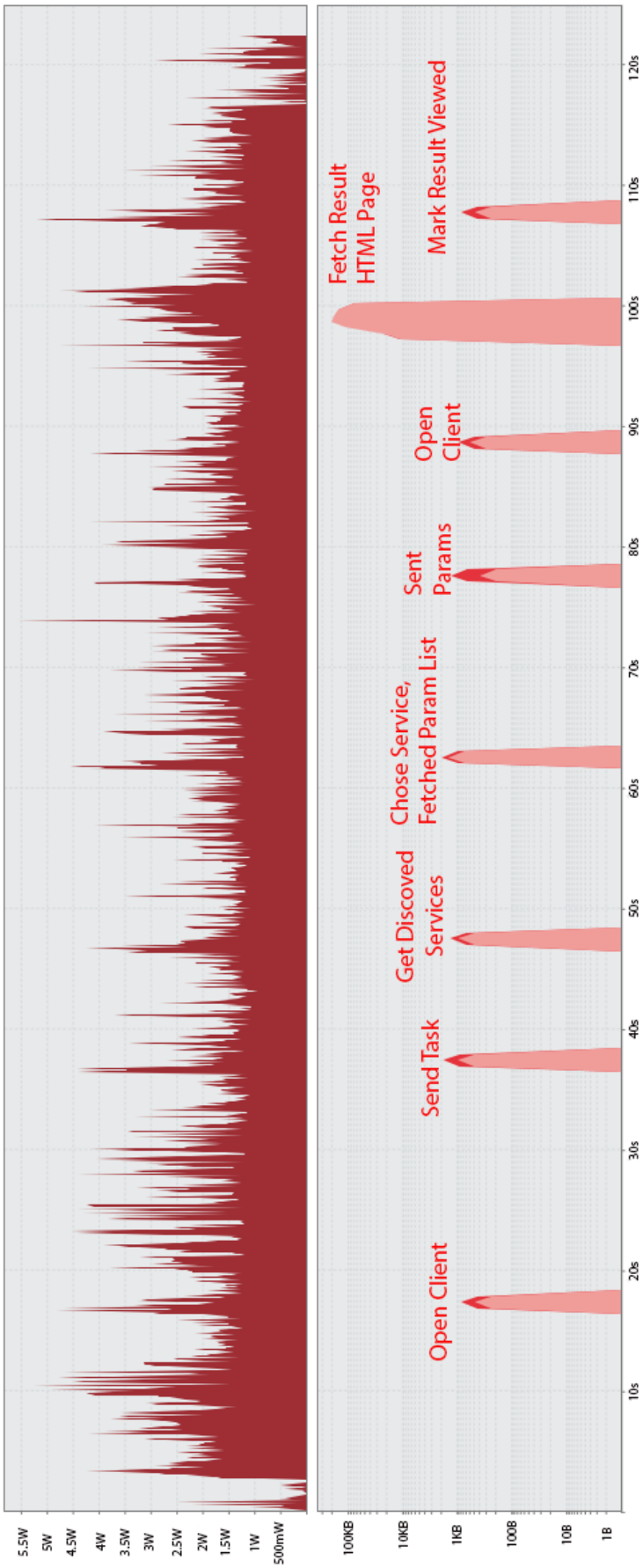


Figure 8.26: Trepn Profiler graphs for battery power usage and data transfer recorded on the CAMCS Client, during the course of interactions required for completing a Google Calendar fetch task, over a Wi-Fi network. Marks in the data transfer graph show the points at which network activity from user interactions occurred. Power usage spiked around network activity to 3W, and data transfer (send/receive) was less than 400KB in total.

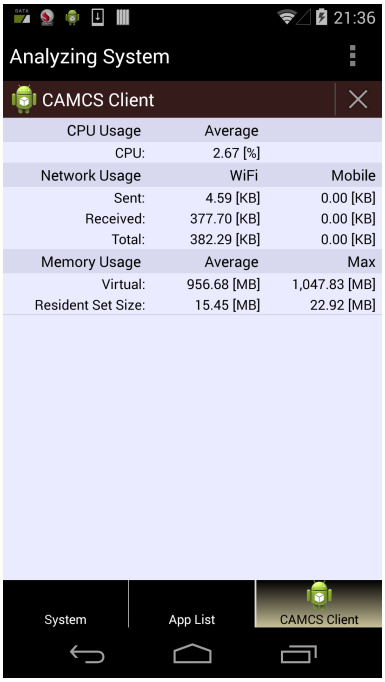


Figure 8.27: Screen capture of the readout from the Trepn Profiler Android application on the Nexus 5, displaying metrics recorded for the CAMCS Client during the calendar task test over a Wi-Fi network. The results should be correlated with the graphs in Figure 8.26.

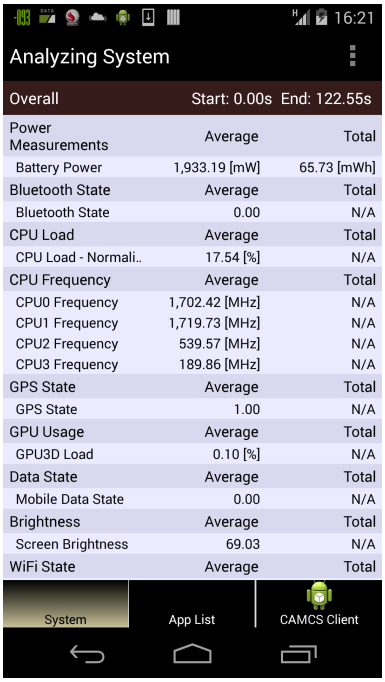


Figure 8.28: Screen capture of the readout from the Trepn Profiler Android application on the Nexus 5, displaying metrics recorded for the OS during the calendar task test over a Wi-Fi network.

cases, such as when opening the client at the start of the test (which downloads the list of current tasks), and fetching the parameter list for the chosen service, it took longer to receive the data over the network (the gaps in the graph are from these steps). The longer wait time increases the energy used from the battery, for the same power consumption levels. Fig 8.30 shows a screen capture from the Trepn Profiler Android application on the Nexus 5 mobile device, which lists the same statistics found for the test over the Wi-Fi network.

The other metrics from the statistics gathered from the TrepnProfiler Android application on the device, show that the peak memory usage for the CAMCS Client during the interaction was 20MB over Wi-Fi, and 24.7MB for the cellular network (unlikely to be due to the network), and CPU utilisation averaged at 2.67% for the Wi-Fi network, and 2.21% over the cellular network. Figures 8.28 and 8.31 show the OS metrics captured by the Trepn Profiler during the Wi-Fi and cellular network tests respectively; these figures demonstrate the higher battery power consumption during the test on the 3G cellular network, as would be expected given the longer wait times present in the cellular network data transfer graph.

These results highlight the low data transfer requirements of the CAMCS Client, and re-enforce the objective that the client does not perform any compute-intensive work. The results also demonstrate that given the longer latency over the cellular 3G network, the client does not contribute any significant resource usage when compared with the Wi-Fi network results, and aside from the longer waiting time for some steps, the client does not suffer as a result.

8.5 User Study

Before analysing the results of these evaluations to determine if they meet with the requirements outlined in Chapter 3, the results of a user study of CAMCS are now presented. Having user feedback from real mobile users shows if the CAMCS approach is a successful one in delivering mobile services, in such a way that meets the integrated user experience goal.

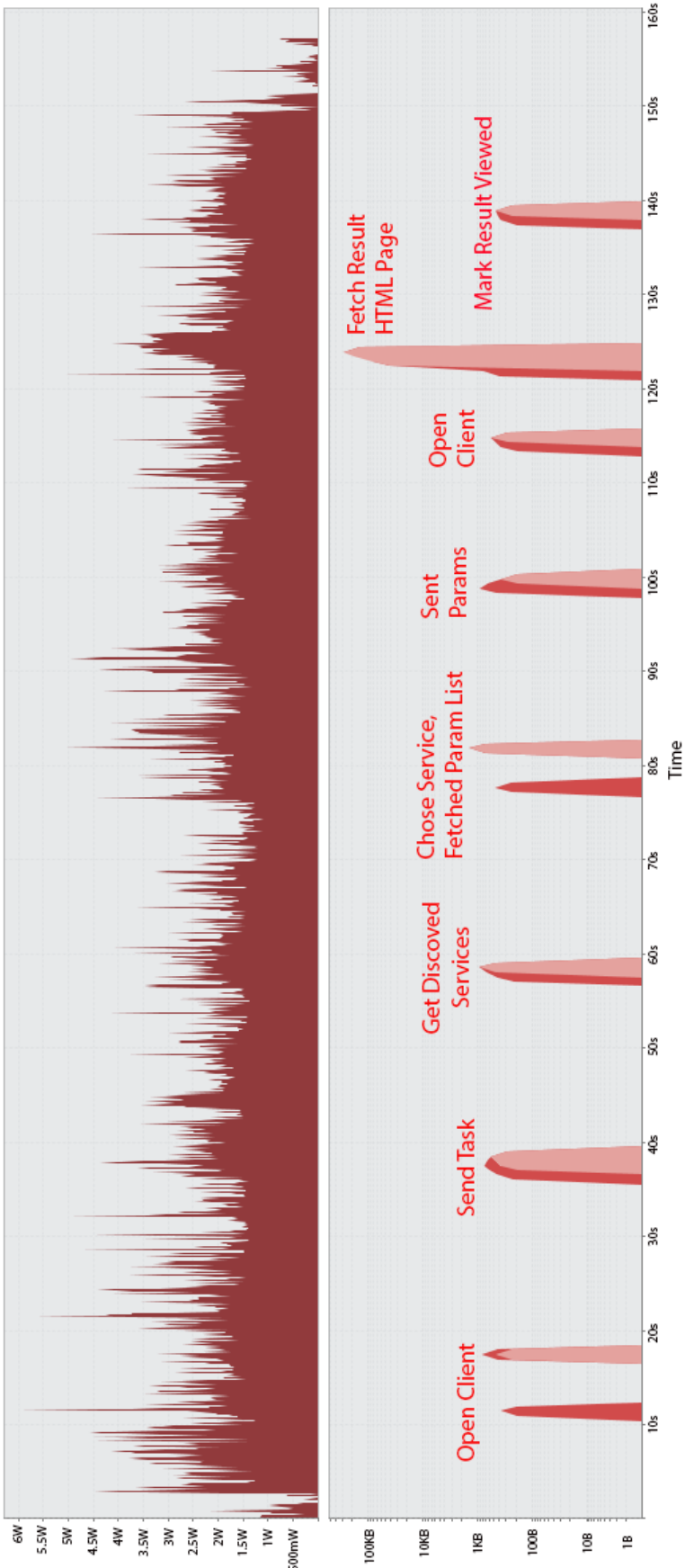


Figure 8.29: Trepn Profiler graphs for battery power usage and data transfer recorded on the CAMCS Client, during the course of interactions required for completing a Google Calendar fetch task, over a cellular 3G network. Power usage and data usage are found to be essentially the same as with the Wi-Fi evaluation, but in some cases, longer response times were observed (which also increases energy usage).

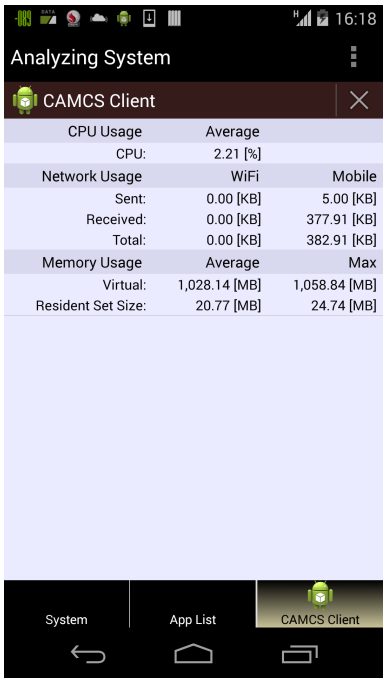


Figure 8.30: Screen capture of the readout from the Trepn Profiler Android application on the Nexus 5, displaying metrics recorded during the calendar task test over a cellular 3G network. The results should be correlated with the graphs in Figure 8.29.

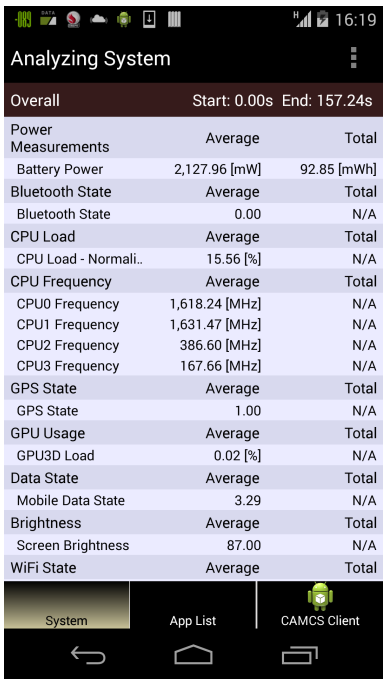


Figure 8.31: Screen capture of the readout from the Trepn Profiler Android application on the Nexus 5, displaying metrics recorded for the OS during the calendar task test over a cellular 3G network. The average battery consumption on the OS during the test was higher than with the same test on the Wi-Fi network, 2128mW against 1933mW.

8.5.1 Study Setup

10 volunteers were recruited in total among final year BSc and MSc students, as well as other PhD candidates in the Department of Computer Science, in University College Cork. This group of volunteers was selected because of their familiarity with mobile devices and applications. The benefits and limitations of the CAMCS solution for working with cloud-based applications and services, when compared with existing application models available to mobile devices, are immediately evident to such users. Users who are less familiar or comfortable with mobile devices may not be able to understand or appreciate the advantages or disadvantages of the CAMCS approach. This does not exclude the possibility of future work where such a study can be conducted again, and expanded to more groups of users with different ability levels.

Each volunteer was provided with the CAMCS Client to download and install on their Android-based mobile device. Once installed, each participant was instructed to register with CAMCS, receiving their own CPAs in the process. For a period of at least 4 days, up until 1-week, the participants were free to use the CAMCS Client to assign task work to their CPAs. The users were able to work with any of the five services that were online during the study. Two weather services were available, one from Geonames.org, and Yahoo! Weather. The Google Calendar service presented in Chapter 5 was also available. In addition, to demonstrate the context awareness features, the Foursquare Tourism service from Chapter 3 was also available to be manually searched and used for tasks. This required each participant, if they wished, to turn on Location Monitoring in the CAMCS Client settings; each user also had to give consent to their location data being shared with Foursquare on the task parameter entry Activity, also shown in Chapter 5. Finally, a Lufthansa Flights Status service, built from the Lufthansa Open API [4], was also available for users during the study, and each participant was also instructed on how to use it to check the status of a flight. This service was used to demonstrate the Service Flow and sessions capabilities, for complex multi-flow-step services, also described in Chapter 5. Due to Facebook API usage restrictions, participants could not use the Facebook integration to provide "likes", work, and education context data, for service ranking.

Each participant was also instructed to setup a task, such as a local weather task, to run automatically at their CPA, at various times of day during the study, so they could experience the automatic task execution model. Finally,

the task re-run feature, with and without task parameter editing, was also highlighted to the participants.

A second evaluation study was also offered, allowing computer science students to develop their own services. This required participants to create their own web service, and use the provided user-oriented service description language to describe it (A Java JAR file containing files was provided for this). Participants in this study were also required to create a result HTML template for their service. However, only one participant took part in this study, and did not complete it due to personal reasons. Therefore, the results of this evaluation are omitted.

At the end of the study, each participant was requested to complete a CAMCS user survey anonymously, that was provided on SurveyMonkey.com, describing their experiences, and perceptions of the usefulness of CAMCS, and his/her CPA.

8.5.2 Study Survey Results

The survey consisted of 17 questions, which can be seen in Appendix A. They were spread across 5 different topics: Experiences with Mobile Computing, Usage of CAMCS and the CPA, CAMCS Context Awareness Support, and User Experience Perceptions. Most questions were based on the Likert scale, and the remaining questions required participants to type in a text-based answer. Of 10 volunteers, 9 participants returned responses.

Looking at Experience of Mobile Computing section of the survey, 7 participants were either "aware" or "very aware" of mobile cloud computing, with 2 unfamiliar. 8 users responded that they were either "aware" or "very aware" of how mobile applications installed on their mobile device make use of the resources such as battery power, storage space, memory, and network data. Participants were also asked to give examples of resource-intensive applications that they would be interested in running on their mobile devices, that they currently cannot run. This question provided interesting answers; 6 users either couldn't think of anything, or did not respond. This is consistent with the observation made in the work by Flores et al [37], that there are few application use-cases that truly required the infrastructure of the cloud. The 3 other responses ranged from intensive video games and video applications, to programming environment tools, with one response simply complaining about

the memory consumption of many applications on mobile devices.

Moving to the CAMCS and CPA Usage section of the survey, all 9 responses rated the CAMCS Client as "easy" or "very easy" to use. 7 responses rated the CPA as a "useful" tool for completing work on a day-to-day basis, with 1 response of "very useful", and 1 "undecided". The survey also asked if more services were available for the CPA, how useful did they feel it would be on a day-to-day basis; this showed an improvement in the responses, with 4 "very useful", and 5 "useful" responses. The participants were also asked how likely would they be to use CAMCS (or a similar product) if it could work with services that could be useful on a day-to-day basis; there were 2 "very likely" responses, 5 "likely", with 2 "undecided". Finally, participants were asked if they found the user-oriented descriptions of services and service parameters, as presented in Chapter 5, to be informative and user-friendly. 4 participants rated them as being "informative", 4 "undecided", and 1 "uninformative". As an aside to the results for this section of the survey, when considering popular services used during the study, CAMCS logs showed that the Yahoo! Weather Service, and the Google Calendar service, were utilised more often that the other services by the volunteers. This demonstrates that services providing updated information throughout the day, possibly on events or data that is subject to frequent change, are likely to be most valuable to mobile users.

The third part of the survey looked at how the users felt about the context awareness features. The survey asked if the storage of personal context data with the CPA in the cloud would be a privacy concern for them. 6 participants were either "very concerned", "concerned", or "a little concerned". 3 participants were either "a little concerned" or "unconcerned". Similarly, 6 responses showed that participants were "very concerned", "concerned", or "a little concerned", about their privacy in the CPA sharing their context data with third party service providers, 2 were "a little unconcerned".

The remaining questions in the context-awareness support part of the survey were only to be answered by those who turned on Location Monitoring, and tried using the Foursquare Tourism service, of which 4 out of the nine participants did. However, some users who did not, answered these questions as "undecided", so these results are excluded. All 4 respondents found the consent model for sharing data with third party services to be "re-assuring" in regards to data privacy. All 4 responses also found the capability of sharing context-data with service providers for task/result personalisation to be either

"very useful" or "useful".

The final part of the survey asked participants about their user-experience perceptions of CAMCS and the CPA. Only 1 participant noticed adverse effects on their mobile device from the installation of the CAMCS Client (slow/unresponsive system, high battery power drain, frequent network activity), and commented that "Phone ran a small percentage slower than normal and battery drained faster. Not major effects. Otherwise, no". In the final question, it was explained to the participants that CAMCS aims to deliver an integrated user experience of mobile cloud applications, and what an integrated user experience was. The question asked did the participants feel that CAMCS achieves this goal. 3 responses stated that "CAMCS strongly achieves this goal", 4 responses for "CAMCS achieves this goal", with 2 "undecided".

8.6 Analysis

This section will analyse the results gathered from the evaluations that were performed on both the AWS cloud servers, and the mobile device, to determine if the CAMCS middleware, and the CAMCS Client for mobile devices, meets the resource utilisation models and user experience requirements, outlined in this thesis.

8.6.1 Mobile Cloud Resource Usage Evaluation

Chapter 6 analysed and modelled the resource requirements of CAMCS, in terms of energy, bandwidth required by the mobile device, and and the physical requirements at the cloud server. Based on the results presented in this chapter, this section briefly evaluates against those models and arguments, to see it they hold true for CAMCS in its complete version.

8.6.1.1 Energy Usage

Chapter 6 showed that in terms of energy usage at the mobile device, related work determined that the important factors to be considered were the energy to power the antenna, the size of the data sent/received, and the distance between the mobile device and the base-station. The analysis was based on work

by Heinzelman et al [50], Equations 6.2.1 and 6.2.2. The chapter argues that the approach taken to mobile cloud computing in this thesis does not contain additional energy costs as with other approaches, such as VM-synthesis costs or profiling costs. The costs would come from an the offloading of the task description, an acknowledgement, and finally, the pushing of the result to the mobile device.

The work on service discovery and consumption added two more steps; receiving the lists of discovered services, and parameter input. The mobile device energy usage results in the previous section show that for the calendar task, less than 400KB of data was used in total (send and receive). Once again, the offload decision model always chooses to offload, because the size of the data sent to the CPA is so small that offload time, and resulting energy usage as a product of power and time, is small. It can also be seen, the because the CAMCS Client does no compute-intensive work, the energy cost association, for the most part, is associated with bandwidth utilisation over the mobile network.

8.6.1.2 Bandwidth Usage

Chapter 6 argued that because there are no certainties surrounding bandwidth allocation in mobile cellular networks, the only approaches to mobile cloud computing that would be successful, and accepted by ordinary mobile users, are those that would assume low bandwidth available. Unlike other approaches, which require continuous, high-volume data transfer, these results and experiments have shown how little data transfer takes place with CAMCS.

8.6.1.3 Cloud Resource Usage

Chapter 6 compared the resource usage at the cloud-server to the requirements for CAMCS. One core aspect of the argument was that each user is not assigned their own VM, as is the case in some of the works in the literature review, and that the VM running CAMCS in the cloud, is essentially distributing complex workload, in terms of service execution, to other servers in the cloud. A simple model was proposed as a guideline to analyse the costs associated with the cloud based resources, Equation 6.5.1, which identified CPU utilisation,

network input/output, memory usage, and storage, as the primary costs associated with a cloud server.

The evaluations performed on the AWS EC2 servers in this chapter show that the VM running CAMCS is not free of workload, with 100% CPU utilisation on the t2.micro instances for the 500 users tested. The experiments on the compute-optimised c4.xlarge instance showed the CPU was under-utilised. This again shows the importance of selecting the right instance type for the compute work required by CAMCS. Fortunately, the model holds; work on completing the tasks is not performed in the same instance as CAMCS. Bear in mind the CPU utilisation graph for the EC2 instance running the N-Queens game as a service, which reached 100% CPU utilisation on the t2.micro tier when tested directly by JMeter. Performance of the CAMCS VM would worsen significantly based on this result, if it was running the services that completed tasks for a large number of users.

The other arguments hold true. Network input/costs have been shown in the cloud evaluations to be small for a large number of users. Memory cost directly depends on the number of users using CAMCS at once, but as highlighted, was only 200MB at start-time. Storage costs on the server additionally now come from storing user task results and context data, HTML files and XML files respectively, as opposed to customised VM images or applications. Task result storage can also be distributed to other cloud storage providers such as Dropbox if the user prefers, which will reduce the CAMCS cloud server storage cost. Context storage may grow, mainly the historical context, which can be time-period limited. Current context will always be small in comparison, as only fresh context data is ever stored with it, as older context is moved to the history.

8.6.2 Meeting User Experience Requirements

To conclude the evaluation of CAMCS and the associated CAMCS Client application for mobile devices, this section will recall the requirements for an integrated user experience detailed in Chapter 3. Based on the research work in the thesis, and the results in this chapter, an evaluation is undertaken to determine if CAMCS meets these requirements.

1. *The approach has to address the latency between the device and the cloud infrastructure*

Considering various networks, especially mobile cellular networks, latency is ever present, and must be tolerated. A successful mobile cloud computing approach, will not suffer from high latency, and will not be noticeable to the mobile user. To accomplish this, one crucial factor must be considered; first, minimisation of data transfer requirements. By not heavily utilising the network, and by not requiring a continuous connection to cloud infrastructure, latency becomes a smaller issue.

In this thesis work, a disconnected, asynchronous approach to mobile cloud is taken. Between each step of a user completing a task, the mobile device is free to disconnect from the cloud. No continuous connection is required, and all task progress is stored with the user's CPA in the cloud. Furthermore, as shown in the CAMCS Client evaluation section of this chapter, the text-based task descriptions (queries), as well as service selection, and parameter inputs, are very small in size. By only transferring small amounts of data, the effects of latency are not noticeable to the end user.

This can further be understood by noting the results from Chapter 6, where experiments were performed to understand the difference between offload times from the mobile device to cloud data centers, both in University College Cork were this thesis work was undertaken, and the Amazon AWS Data Centre in Virginia, USA. For the 500KB data offloads that were performed, the box-plots of the offload times to each deployment showed little difference, aside from greater variation in the Amazon results. Also consider that the mobile results show that the data transferred for the calendar task in the CAMCS Client evaluation section in this chapter, was less than 400KB, with only 4KB being sent to the user's CPA, rather than anywhere close to 500KB or more.

For non-real time applications, and applications with small data transfer requirements, the effects of latency are not significant, and CAMCS meets this requirement because of its disconnected, low data transfer design.

2. *The approach has to minimise bandwidth utilisation*

   As with addressing the latency requirement above, the small amount of data transferred between the mobile device and CAMCS in the course of completing a task (as shown by the results in this chapter), CAMCS meets this requirement because of its low data transfer requirements.

3. *Device workload overhead must be minimised*

   For the most part the work of the CAMCS Client simply involves collecting data or actions from the mobile user to send to their CPA within CAMCS, or presenting output from the CPA. These operations include taking in parameter data for tasks, and presenting task results. In the CAMCS Client evaluation section, for the whole course of the interactions required with the CAMCS Client for completing the calendar task, the average CPU utilisation was only 2.67%, and the memory allocation for the client application only peaked at 20MB. Energy usage above the base-line only occurred during network send/receive operations, which were not long running.

   The only operations performed by the CAMCS Client that may be considered costly include the collection and sending of context data, and the network state monitoring for the offload decision maker. The mobile user is free to set the frequency with which context updates occur if they are concerned about the energy cost, or turn them off altogether. In terms of monitoring the mobile network state, recall that no profiling occurs as in other works; the decision model only uses information already available to it from the mobile device, namely the GSM signal strength, and as such it is described in this thesis work as a zero-overhead model.

   Based on these aspects, CAMCS does not impose any significant workload overhead on the device.

4. *The approach must gracefully handle mobility aspects such as disconnection*

   CAMCS with its disconnected and asynchronous task execution model, in terms of task description, discovery, service selection, service execution, to finally receiving the task results, has been guided by the principle of disconnected operation from the beginning. Context data for the user can also be inferred from historical context stored with CAMCS on the cloud, should no new fresh context be available because of disconnection. Other mobility factors such as handoff are also of no concern, as all progress for completing tasks is safely stored with CAMCS in the cloud.

   As the mobile device is not responsible for any aspect of the work involved with work to complete a task aside from user input, CAMCS meets the mobility requirements.

5. *Provisioning for context awareness must play a central role*

   CAMCS, as hinted in the name, can take advantage of user context data for both service discovery and task execution with those services. Context can be received from the mobile device and other sources, such as social networks. As part of the service description format outlined in Chapter 5, provisions have been made for Context Parameters, so that services can describe what kind of context they can work with. In the absence of fresh context, new context can be inferred from historical context data. Context Profiles allow the CPA to carry out different work at different times, depending on the active profiles. Additionally, the capability is also included, because of the Context Definitions, for developers to create their own Contexts from the Simple and Complex Contexts that already exist within CAMCS, and their own Context Profiles based on these contexts.

   Context data is therefore readily available for use with CAMCS, therefore meeting this requirement.

6. *The solution must uniquely cater for the mobile user, rather than the desktop user*

   CAMCS is a solution uniquely catered to use on mobile devices, not just in terms of its support for mobility concerns such as disconnection, unlike other solutions where desktop VMs are used, CAMCS takes advantage of unique mobile features. These include the data provided by the sensors for the context awareness requirement above, which would not be possible with a traditional desktop operating system running on a VM. The resource requirements of CAMCS on the mobile device also respect the limited supply of energy and bandwidth available to the mobile device. Task results, based on HTML web pages, which can be displayed on smart mobile devices using Android WebView (or equivalent) is also catered to mobile devices, while being platform neutral in its use of HTML and other web technologies.

   By respecting the mobility properties of mobile devices, and by using their distinct features described, CAMCS is specifically a mobile device solution to the mobile cloud computing paradigm, rather than an approach that adapts desktop technologies to mobile devices.

7. *The thin client on the mobile must provide an adaptive UI and services*

The adaptive nature of the UI is primarily provided by the offloading decision maker described in Chapter 6, which is designed to respect the current resource status of the mobile device. The factors taken into account by the decision maker are the network signal strength, the size of the data to be offloaded, the battery status, and task priority. The algorithm, as already explained is zero-overhead in it's own right. Depending on the factors outlined, the model will either offload the task immediately, or queue it until the network signal strength improves.

Ironically, because the amount of data transferred per-task is so low, the decision maker essentially always offloads immediately. In the results in Chapter 6, only when the data size was greater than 500KB was any noticeable increase visible in offload times.

In terms of the other costly operations associated with the CAMCS Client, namely the context data collection from the sensors, this can be switched off or have update frequency reduced, as already discussed. The CAMCS Client can cater it's own operation to the current situation, but rarely does any action need to take place.

8. *A standards-based solution must be used*

   Essentially, CAMCS does not use any standards solution for working with services, as the service description format created in Chapter 5 is used. However, CAMCS does work with existing service technologies, such as RESTful services, which, for the most part, are not generally described in any form of description language apart from API documentation. WSDL files that are used to describe SOAP-based services, can easily be translated into the service description format created for this work, as all the elements of the service description format have an equivalent markup in WSDL.

   Considering these facts, CAMCS is considered to be backwards compatible with all existing standards. Nothing prevents existing services from being used, aside from developers taking the time to create a service description for their service, for insertion into the registry.

8.7 Conclusions

This chapter has presented the results of experiments performed to evaluate the CAMCS middleware running on the Amazon AWS cloud, and the CAMCS thin client application running on Android-based mobile devices. The aim of these experiments was to understand the performance of both the CAMCS middleware and the CAMCS thin client, and determine if the resource models presented in Chapter 6, in terms of energy, bandwidth, and cloud-server resource usage, correlate with the finished development state. More importantly, the evaluations aimed to determine if CAMCS conforms to the requirements outlined for an integrated user experience in Chapter 3 of this thesis.

The cloud test of CAMCS on Amazon AWS investigated the performance in terms of how cloud resources were used, such as CPU, memory, and network I/O, with hundreds of simulated users, with the Apache JMeter load testing software. Other result metrics from JMeter, important for the user experience, such as response times and request throughput, were also analysed, in terms of no scaling, horizontal auto-scaling, and vertical scaling, of Amazon EC2 instances. Results showed that CAMCS can effectively scale to handle high demand, and can perform well for the mobile users, when running on the right type of cloud instance (compute optimised), with the correct scaling policies for effective resource utilisation.

The evaluations on the CAMCS Client on the Android mobile device were performed with the Trepn profiler. Experiments were performed to understand resource usage during the normal use of completing tasks, with both Wi-Fi and cellular 3G networks, as well as the resource usage of the application installation. The results showed very low resource utilisation in terms of CPU, memory, and network usage.

The results of a user study that was performed were also presented. A survey taken by each participant at the end of the study showed that CAMCS and the CPA were perceived well, as a useful tool. All participants agreed that CAMCS delivers on the goal of providing an integrated user experience of mobile cloud applications.

Finally, an analysis showed that the CAMCS middleware, and the CAMCS Client on mobile devices, match most of the usage metrics defined in the resource usage models, and met with all of the integrated user experience requirements defined for mobile cloud computing solutions.

The final chapter will summarise the research presented in this thesis, along with highlighting the benefits and limitations of this work. Areas of future work will also be presented.

Some of Sections 8.3, 8.4, and 8.6 will appear in [101].

Chapter 9

Conclusions and Future Work

This chapter will summarise the research work presented in this thesis, and present the conclusions of the research. Finally, based on the results and limitations of the work, areas for future work based on the research undertaken will be presented.

9.1 Summary of Research Problem and Solution

Mobile cloud computing is a paradigm which promises to overcome the limitations present in the capabilities of mobile devices, based on their physical constraints. Several approaches that seek to implement this paradigm were presented in the literature review. However, many of these approaches place demanding requirements on the mobile device, such as the need for frequent high-volume data transfer. This can be costly in terms of energy and bandwidth usage; energy use drains the device battery, and the bandwidth can be costly depending on the data plan from the mobile network operator. Mobility issues also play an important role; continuous high-quality connections are required to the cloud infrastructure from the mobile device. If the signal is lost, the solution becomes ineffective.

Based on these problems faced by the existing approaches, it can be said that they deliver a poor user experience. The literature review shows that these other works do not consider the user experience as a factor in their implementation. Delivering a positive, integrated user experience of mobile cloud applications and services is crucial to the success of this paradigm. An inte-

grated user experience of mobile cloud applications and services, is defined as a model that:

- Provides seamless interaction between the mobile user and the cloud.
- Enables rich new functionality and models of interaction between the mobile user and the cloud.
- Respects the limited resources available to the mobile device.

Based on the analysis in the literature review, a mobile cloud middleware approach was adopted to deliver a solution that would conform to this integrated user experience. The middleware, known as Context Aware Mobile Cloud Services (CAMCS), delivers cloud-based services to mobile users, in a disconnected, asynchronous fashion, while respecting the resources available on the mobile device.

The primary component of CAMCS that mobile users interact with, is the Cloud Personal Assistant (CPA). Each user of CAMCS is assigned their own CPA upon registration. A mobile user's CPA can be thought of as a trusted, third party representative of the user, that works to complete tasks assigned. The mobile user communicates and interacts with his/her CPA through a thin client application installed on the mobile device, the CAMCS Client.

9.2 Conclusions

CAMCS was designed as a mobile cloud computing middleware, which completes work for the mobile user in an asynchronous, disconnected fashion. The aim for implementation was to deliver an integrated user experience of mobile cloud applications. In the literature review, other mobile cloud computing approaches were found not to respect the user experience. In Chapter 3, based on an analysis of the related work in the literature review, several requirements were derived that future approaches to the mobile cloud computing paradigm should implement, in order to provide an integrated user experience. These requirements take into account the unique nature of mobile devices, in terms of their limited resources (energy, bandwidth), their mobility (portability, disconnected nature), and their advantages compared to desktop computers (e.g. sensors for context awareness). The design and system architecture have been presented, along with detailed descriptions of the compo-

nents and features that make up CAMCS. An evaluation was also presented detailing how CAMCS is expected to meet the user experience requirements outlined. At the design stage, CAMCS was expected to be a disconnected solution, with low data-transfer requirements, and that would make use of user-context data from mobile sensors. The evaluation determined that because of these factors, CAMCS would not suffer from the same energy, bandwidth, and latency problems as other approaches had. Finally, several application scenarios and models were presented as additional features that could be provided by CAMCS. Ultimately, these were implemented in other chapters.

The use of context awareness is intended to personalise service execution and results to the user's situation. The Context Processor component of CAMCS was incorporated to deliver this requirement. In Chapter 4, the implementation of the Context Processor component of CAMCS was presented. The literature review presented the many challenges and considerations that were present in the decision on how the Context Processor would be implemented, in terms of context representation, inference, and storage. Based on this analysis, the use of Ontologies was selected as the technology of choice. In particular, the SPICE mobile ontology was utilised to represent user context data. The OWL API was used for writing and storing current and historical user context into XML files, and for its inference engine support for querying and reasoning. The structure of context within the processor was also presented, along with the considerations included for developer extension. Additionally, Context Profiles were introduced. These allow a CPA to undertake different kinds of work, depending on what context profiles are active. The evaluation involved the creation of some cloud services that made use of context data collected by CAMCS and provided to user CPAs; these included a Foursquare-based tourism/places of interest service, making use of location context, and an experimental Twitter traffic service, which made use of location and activity context, to pull tweets from Twitter that may be relevant to traffic conditions while the user is travelling to work by car. The aim of the work was to show how user context data could be collected by CAMCS from the CAMCS Client, stored by the Context Processor, and then be provided to CPAs to personalise cloud service execution to the user's situation.

CAMCS completes user work, created as tasks, using existing cloud-based services which utilise existing SOA technologies. In Chapter 5, a solution was presented for solving the problem of mobile users being able to discover and consume cloud-based services from their mobile device. Traditional SOA web

service standards, such as XML-based WSDL files are not effective for describing services to mobile users without a technical background. Therefore, a custom, user-oriented mobile cloud service description format was developed and presented. This structure allows cloud-based services to be described in a user-friendly way to mobile users, allowing them to take part in the service discovery process. Existing service markups, such as WSDL, can be easily converted to the description format. Support is provided for services that complete work in a single call, or services that require several calls/steps to complete, that make up a Service Flow. These service descriptions are stored in a custom service registry, which is queried by the CPA when a user creates a task. The user can choose from the discovered services on their mobile device, and provide the appropriate input parameters. The mobile user sees no technical markup describing the service, just user-friendly names and descriptions of services, their operations, and parameters. The description format also has support for context parameters, allowing a service which makes use of user-context data to describe what data it should receive and for what purpose. Service sessions were also introduced to the CPA to support maintaining data between steps, for tasks where the user selected a Service Flow for task execution. Optimisations were also shown that were included as part of the processes, including part-of-speech (POS) tagging to remove useless query tokens, and the ranking of discovered services based on user's context data ("liked" pages on Facebook, showing user interests), based on the Normalised Google Distance for semantic similarity calculations. A complete explanation was provided to show how a mobile user can use the discovery and consumption solution to find services to complete tasks. Accordingly, the task model used by CPAs was also presented, showing the implementation and the work it performs. The chapter also introduced result templates, which allow developers flexibility to format service results to their liking or company branding. This is a HTML based solution, so that platform and device neutrality is achieved.

As CAMCS was designed to provide an integrated user experience, understanding how the scare resources available to the mobile device would be used, was a crucial concern. In Chapter 6, an analysis was performed to understand how mobile devices make use of energy and bandwidth resources available to them, both in little supply. Having studied works addressing these problems, models and guidelines were presented that took into account the unique mobile cloud considerations, such as the continuous connections and data transfer requirements of some of the approaches presented in the literature review. The

existing approaches, as well as that taken by CAMCS, were analysed against these models, to show that CAMCS can achieve effective usage of the minimal resources available, compared to other approaches. An analysis was also presented of the requirements encountered by each approach (including CAMCS) at the cloud server infrastructure, in terms of resource usage (CPU processing, storage, memory, and network I/O required). As CAMCS makes use of services distributed on various servers in the cloud, and does not rely on customised mobile OS runtimes on the cloud server (amongst other requirements), CAMCS can be considered a low resource application. Experiments were also presented to evaluate mobile-cloud offload performance on cellular networks, in terms of signal strength against offload data size, data center location, and power requirements against signal strength. For the small data offload sizes used by CAMCS, offload times were fast while GSM signal strength was greater than -100dBm. Data center location was shown to have little difference for small data sizes, and power draw only peaked in the experiments for most cases at 3.3 Watts, against a background consumption of 1.5 Watts. The results were used to create a zero-overhead offload decision algorithm for the CAMCS Client. It is a tradeoff compared with other mobile cloud approaches that use profiling; there is no profiling time or energy overhead, as the model uses data already available to the device, but it may be less accurate. However, because of the small data sizes and infrequent use, this is not a concern. CAMCS was shown to be use resources sparingly and effectively.

The architecture of CAMCS supports pluggable modules for additional functionality, and several use-cases and scenarios were proposed in Chapter 3. In Chapter 7, CAMCS and the CPA were expanded to provide support for three additional application models proposed; file synchronisation, XML data processing, and collaborative group work. The implementations of each model were presented, along with some of the difficulties and challenges encountered while implementing each model, which can be used to guide further software development for mobile cloud applications. Results showed that file synchronisation with the CPA can be an effective time and energy saver for saving files across multiple cloud services at once, rather than synchronising to each service provider separately. XML data processing results showed the time advantages offered by the cloud for completing this work, compared to local processing on a mobile device. Use-case scenarios were presented for the group-based collaborative CPAs, such as multiple employees working together in an office, using their CPAs to coordinate and complete work on a

common task.

To understand if CAMCS and the CPA approach to mobile cloud computing can be effective, an extensive evaluation was performed in Chapter 8, with both the middleware running on Amazon EC2, and the mobile client running on a Google Nexus 5 device. Experimental evaluations took place using the JMeter load tester to evaluate how CAMCS performed on Amazon EC2 with up to 1000 users, who were requesting a demanding compute intensive N-Queens puzzle be solved for random values of N between 3 and 8. Overhead from service discovery and service execution was also evaluated individually, to gauge the performance of CAMCS by itself. Evaluations took place with no scaling, horizontal scaling (increasing/decreasing the number of running VMs with CAMCS), and vertical scaling (switching to a VM with higher-specification hardware resources). Results showed that CAMCS can be scaled up effectively, both horizontally and vertically, to cope with demand. The use of both types of scaling reduced average response times and increased request throughput. With 500 users constantly loading the server, with horizontal auto-scaling enabled, average response time of CAMCS was reduced from 8 seconds to 5 seconds, with throughput increased from approximately 45 requests per second to 70. With 1000 users constantly sending requests and auto-scaling enabled, average response time was reduced from approximately 17 seconds to 11 seconds, with comparable throughput increase to that with 500 users.

Continuing the evaluation with performance on the mobile device, CPU utilisation, data transfer, and power consumption during the steps of completing tasks, were found to be small; 2.67% (on network Wi-Fi) and 2.21%(on 3G cellular network) average CPU utilisation, 377.7KB data transfer for competing a Calendar task (with 372.3KB coming from the result HTML file alone), and power consumption corresponding to the experiments in Chapter 6 for both cellular 3G and Wi-Fi networks. Next, a user study was undertaken with volunteers, who installed the CAMCS Client on their mobile devices, and used CAMCS for several days to complete tasks. The different task execution models and features were experimented with. Each participant completed a survey at the end of the study. All participants found CAMCS to be a useful tool for completing tasks, and agreed that CAMCS delivers on the goal of providing an integrated user experience of mobile cloud applications. An analysis was performed to confirm that CAMCS conformed to the resource usage models in Chapter 6. Finally, the requirements for an integrated user experience pre-

sented in Chapter 3 were re-visited. Based on the results of the experiments, CAMCS was evaluated against each requirement, to confirm that they were met. This analysis concluded that CAMCS delivers an integrated user experience, as defined by these requirements.

9.2.1 Benefits

The benefits of the work presented in this thesis are now briefly summarised from the conclusions presented.

- Based on the results of the evaluations, CAMCS can deliver the integrated user experience requirements outlined. The disconnected approach taken requires no continuous connection to the cloud, and there will be no loss of work should disconnection occur. The CAMCS Client was shown to exhibit low data-transfer requirements, and therefore, low energy usage. This should be contrasted with other approaches which do not meet these requirements.
- CAMCS can personalise the user experience by use of context data gathered from the mobile device. CAMCS can provide context information to cloud services, which can influence the output of a service to the user's situation. Other works have only shown how context data can be collected, processed, and stored, typically within a middleware solution. Support is also provided for developer extension.
- A generic mobile cloud service description, discovery, and consumption solution, designed for use with the CPA, was presented, allowing end-users to find the mobile cloud services they need to complete their work, in a user-friendly way, hiding away all technical details. This is compatible with all cloud-based services, and the solution can even be adapted for other projects outside the scope of CAMCS. Other works have required end-users to know the technical details of a service, or can only work with a specific set of limited services. Both of these points do not follow an integrated user experience.
- The study of mobile cloud resources modelled the low-resource usage that the CAMCS approach can bring when compared with other approaches to the mobile cloud paradigm, specifically in terms of energy usage, bandwidth utilisation, and cloud server resource usage.

- CAMCS is extendable to other functionality, as was shown with the additional application models presented. This can be adapted to provide more services directly through CAMCS, utilising the disconnected approach for the user experience aims.

9.2.2 Limitations

The CAMCS approach, and work presented in this thesis, presents some limitations that should be addressed; some of these are also used for proposals for future work.

- By its nature, CAMCS can only deliver non-interactive applications; that is, applications that require a user to provide constant input based on a continually arriving output, such as a game, are not feasible with the CPA, due to its disconnected approach. However, a disconnected approach to playing a game would essentially defeat the purpose of playing it.
- The premise of mobile cloud computing has been that demanding applications, such as the typical use-case of augmented reality, are executed in the cloud, instead of on the mobile device, yet the services presented in this work do not feature demanding, compute-intensive work. However, as described in Chapter 2, work by Flores et al [37] made the interesting observation that aside from cases such as augmented reality, few mobile applications really require the compute-intensive capabilities offered by the cloud.
- Applications with real-time requirements also cannot be guaranteed for on-time delivery to the mobile user through a CPA, again because of its disconnected nature.
- CAMCS is still limited in the user input it can receive for service parameters. This means that in its current form, CPAs can only take in simple data, and work with services that work with this data. Many services in the cloud require direct user-authentication for performing work for a user. The work can be extended to support more data input types, such as files from a mobile device, and to support more complicated interactions with services.

- Existing services in the cloud cannot work with the CPA in their existing RESTful/SOAP forms. This has limited the potential of the CAMCS solution in this work. This is because without using the customised discovery and consumption processes through CPAs, specific types of required data input parameters could not be collected, as they are often described technically, or not described at all. Additionally, responses from services also contain technical data that are of no meaning to ordinary end-users. There are no common structures, interfaces, or input/output standards to services offering similar functionality that the CAMCS Client could be programmed with in advance; all services expose completely unique APIs. These problems drove the creation of the description, discovery, and consumption work in this thesis. While tools can be developed to automatically describe services in the description format required by this work, developers would have to make time to manually undertake this work. As quick as this process may be, they may find it off-putting.

9.3 Future Work

Results and analysis from the evaluations that have been performed have shown that CAMCS can deliver an integrated user experience of mobile cloud applications. However, the work still presents limitations and challenges that can be addressed in future work with further research and evaluation.

- *Expanding Compatibility for Existing Services*

  In order to evaluate CAMCS, some experimental services were developed for proof-of-concept demonstrations. CAMCS is capable of working with all existing cloud-based services that utilise existing SOA technologies, namely SOAP-based services and RESTful services. These existing services need to be described using the service description format described in Chapter 5, and placed into the service registry, in order for them to be found by the CPA. This should be a simple process for a developer working on a few services, but porting entire directories of services to the format was a limitation of this work, due to the time that would have been involved. Unfortunately, because of the way existing services are described (with WSDL), or possibly not described (as with RESTful services), they are not immediately suitable for use with the CPAs user-

oriented discovery process. In future work, tools can be developed to take existing WSDL files, or other API documentation as input, and produce as output, a service description for the service, that uses the service description format in this work. The registry can be expanded with a web based interface as well, to allow developers to create descriptions in a user-friendly environment. These descriptions could be inserted into the registry automatically by the web-based interface.

- *Real Time Services and Task Models*

  The CAMCS solution was described previously as having the limitation of being suitable only for non-interactive applications, where synchronous, continuous back and forth interaction is not required between the mobile user, and a task being executed. Future work can deliver a real-time model. For example, services can feed task results with a continuing stream of data, such as updated weather, news headlines, or live updates from a feed service or website. This can enable synchronous tasks, where the user can provide direct feedback to the service being used for the task, through his/her CPA. An enabling technology for this can be the Mobile Edge Computing standard by the ETSI, described in the Literature Review in Chapter 2. Services running on computing nodes located at the network edge, typically one-hop away from the mobile device for low latency communication, can drive such a real-time task model.

- *Service API Modifications*

  It is important to consider that service APIs can change or be removed completely, which the current implementation does not account for. This presents a problem for tasks that are re-run by mobile users. As CPAs store the descriptions of services chosen by mobile users to complete tasks, they would not know that the API has changed until they try and contact the service, only to receive an error response. This could be addressed by adding a service description validity period, where the service provider agrees to keep the service as specified by the API available, at least until that time. If that time passes, the CPA would have to re-perform service discovery, and fetch an updated description. This could also present a problem for new tasks, where the user chooses a stale service during discovery; the description could be out-dated or the service it describes could have been removed. The registry should be able to pro-

vide extended API options for descriptions to be removed and updated.

- *Service Registry and Service Description Extensions*

  Further optimisations to the service registry, in addition to the extended API operations discussed, should be considered. This work already presented optimisations including Part of Speech (POS) Tagging, and service ranking based on context. Possible future work includes adding service feedback and reviews by mobile users. This data could be provided by the registry, and used in the ranking process and presented to other mobile users during service discovery, to give a better idea of how each user felt about the service having used it.

  Some steps of the service discovery and execution could become more dynamic, in regards to Service Operations that are of the Service Flow type. Even though the service description format provides the ability to have multi-step services that make up a Service Flow, supported by sessions, the steps could become more flexible. In the current implementation, services can send a choice of options for selection steps back to the CPA, based on inputs from previous steps, it would be ideal if the data entry steps could also dynamically change during the course of task execution, based on options chosen by the user in selection steps. Currently all data entry steps are static, and must be defined at description time.

- *Open APIs*

  It as highlighted in Chapter 5 that a significant limitation was found to be that many service provider API's were limited in access to developers with developer keys, with a criteria of a maximum number of requests allowed over a 24-hour period. Each mobile user will have no development key; the short term solution was to hardcode the CAMCS developer key into the service descriptions as a parameter, invisible to the mobile user. For the CPA approach to be successful, APIs must become more open. There is the concern that service providers would not be happy to support their infrastructures receiving calls from thousands of user CPAs per day, straining their own server resources and associated costs. An agreement would have to be made, or approaches taken in other related works in the literature review, such as physically moving copies of services from the service provider, to the client (in this case, the VM running CAMCS) would need to be explored.

- *Extended Security Features*

  Additional authentication and authorisation mechanisms can be added onto CAMCS. To prevent bot registrations, users can be put through a user account activation process, by means of email address verification. Additionally, the CPA model for encryption of user task data with public and private keys can also be expanded, to also encrypt context data stored in the current and historical context ontology XML files.

- *Tasks Feeding Other Tasks*

  Similar to the data-sharing middleware COSMOS proposed by Sankaranarayanan et al [116], and the web service mash-up middleware by Wang and Deters [144], the task model used by CPAs could be expanded, so that the result of one task, could become an input to another task. This may require expanding service descriptions to introduce some standard format to task results, in order to provide meta-data which could mark up and identify specific parts of a task result that could be considered as a meaningful service input.

- *Expand Context Data Sources*

  This work only uses a limited number of sources for user context. For Android-based mobile devices, Google Play Services only offers Location and Activity contexts updates, as a standard API. Other sources from the mobile device can be used, such as the microphone, and light sensor. Now with the prevailing use of smart watches connected to smart mobile devices, many health-related metrics can be pulled from these devices as well. Some devices even have temperature sensors. The problem is that all these extra sources of context cannot be guaranteed to be available on all devices, but the CAMCS Client can be modified to make use of these sources if they are available.

- *Expanding Collaborative Group-Based Work with CPAs*

  An initial introduction and implementation was introduced in Chapter 7 that showed how CPAs could be organised into groups, for example, in a company office settings, to engage in collaborative work to complete common tasks; many CPAs working together with services, completing milestones of larger jobs. This can be implemented and evaluated further; a communication language or protocols would need to be devised

for CPAs in order for them to coordinate, assign responsibilities, and exchange data.

- *Federated CPAs*

  Aside from the group collaborative works, CPAs can become part of federations, which can share information and data between CPAs within a given federation. For example, mobile users who are friends and know each other personally, may wish to share files or tasks between their CPAs. Federations can enable a "friend" relationship between CPAs, which can then allow trusted sharing. This would also require a formal protocol and communication language/syntax between CPAs.

- *Porting to Other Mobile Platforms*

  The CAMCS Client is only currently available for the Android OS mobile platform. This can easily be extended to other platforms, such as Windows Phone, and Apple iOS. Considering that the aim of CAMCS is to move work away from the mobile device and into the cloud, this can be achieved without significant development time. Furthermore, CAMCS has already taken steps to be platform neutral, such as with the HTML-based task results, which can be displayed on existing mobile devices.

- *Dynamic Mobile Client for Pluggable Modules*

  Currently, if a pluggable module, such as the application models presented in Chapter 7, are implemented on CAMCS, the CAMCS Client would need to be updated manually to support the new functionality. The dynamic form rendering based on service descriptions for data parameter input, could be adopted here to provide dynamic mobile client user interfaces for each pluggable module into CAMCS.

- *Direct Performance Evaluations with Related Works*

  Many of the solutions presented in the related work that aim to implement the mobile cloud computing paradigm, do not feature publicly available software or programming frameworks, that would allow a direct comparison user-study between, for example, Cloudlets, and CAMCS. One exception to this is the Sapphire platform by Zhang et al [149], which features support for building applications with code offloading. If more software was available, a CAMCS version of an application based on SOA service technologies, could be developed alongside a

Cloudlet or code-offload version of the same application, and compared directly by the mobile user.

References

[1] Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: a mobile context-aware tour guide. *Wirel. Netw.*, 3(5):421–433, 1997.

[2] Eyhab Al-Masri and Qusay H. Mahmoud. Mobieureka: an approach for enhancing the discovery of mobile web services. *Personal Ubiquitous Comput.*, 14(7):609–620, 2010.

[3] F. AlShahwan and M. Faisal. Mobile cloud computing for providing complex mobile web services. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, pages 77–84, 2014.

[4] Lufthansa Open API. `https://developer.lufthansa.com`. Accessed: 2015-09-26.

[5] A. Awad, A. Matthews, and B. Lee. Secure cloud storage and search scheme for mobile devices. In *Mediterranean Electrotechnical Conference (MELECON), 2014 17th IEEE*, pages 144–150, 2014.

[6] Amazon AWS. Amazon ec2 api documentation.

[7] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 280–293, 1644927, 2009. ACM.

[8] Jakob E Bardram. *The Java Context Awareness Framework (JCAF)–A Service Infrastructure and Programming Framework for Context-Aware Applications*, pages 98–115. Springer, 2005.

[9] Nor Shahniza Kamal Bashah, Natalia Kryvinska, and Do van Thanh. Service discovery in ubiquitous mobile computing environment. In *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services*, pages 763–767. ACM, 2010.

[10] Aaron Beach, Mike Gartrell, Xinyu Xing, Richard Han, Qin Lv, Shivakant Mishra, and Karim Seada. Fusing mobile, sensor, and social data to fully enable context-aware computing. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, pages 60–65. ACM, 2010.

[11] Anton Beloglazov. *Energy-efficient management of virtual machines in data centers for cloud computing*. PhD thesis, Department of Computing and Information Systems, The University of Melbourne, 2013.

[12] IBM Bluemix. `http://www.ibm.com/cloud-computing/bluemix`. Accessed: 2016-01-05.

[13] Steven Bohez, Elias De Coninck, Tim Verbelen, Pieter Simoens, and Bart Dhoedt. Enabling component-based mobile cloud computing with the aiolos middleware. In *Proceedings of the 13th Workshop on Adaptive and Reflective Middleware*, pages 1–6, 2677019, 2014. ACM.

[14] Cristian Borcea, Xiaoning Ding, Narain Gehani, Reza Curtmola, Mohammad A. Khan, and Hillol Debnath. Avatar: Mobile distributed computing in the cloud. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2015 3rd IEEE International Conference on*, pages 151–156, 2015.

[15] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[16] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical report, Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[17] Chen Chien-An, Won Myounggyu, R. Stoleru, and G. G. Xie. Energy-efficient fault-tolerant data storage and processing in mobile cloud. *Cloud Computing, IEEE Transactions on*, 3(1):28–41, 2015.

[18] Jason H. Christensen. Using restful web-services and cloud computing to create next generation mobile applications. In *Proceedings of the 24th*

ACM SIGPLAN conference companion on Object oriented programming systems languages and applications, pages 627–634, 1639958, 2009. ACM.

[19] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314, 1966473, 2011. ACM.

[20] S. Clinch, J. Harkes, A. Friday, N. Davies, and M. Satyanarayanan. How close is close enough? understanding the role of cloudlets in supporting display appropriation by mobile users. In *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on*, pages 122–127, 2012.

[21] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62, 1814441, 2010. ACM.

[22] W3C Standards: Linked Data. `http://www.w3.org/standards/semanticweb/data`. Accessed: 2016-01-05.

[23] Sohini De and Suddhasil De. Uncoupling in services of mobile cloud computing using tuple space model: design and formal specifications. In *Proceedings of the first international workshop on Mobile cloud computing & networking*, pages 27–32, 2492355, 2013. ACM.

[24] Lien Deboosere, Bert Vankeirsbilck, Pieter Simoens, Filip De Turck, Bart Dhoedt, and Piet Demeester. Cloud-based desktop services for thin clients. *Internet Computing, IEEE*, 16(6):60–67, 2012.

[25] Twitter Developer. `https://dev.twitter.com`. Accessed: 2015-09-10.

[26] Huang Dijiang, Zhou Zhibin, Xu Le, Xing Tianyi, and Zhong Yunji. Secure data processing framework for mobile cloud computing. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 614–618, 2011.

[27] Docker. `https://www.docker.com`. Accessed: 2015-09-26.

[28] Google Drive. `https://drive.google.com/drive`. Accessed: 2015-09-09.

[29] Dropbox. `http://dropbox.com`. Accessed: 2015-09-09.

[30] Khalid Elgazzar, Hossam S. Hassanein, and Patrick Martin. Daas: Cloud-based mobile web service discovery. *Pervasive and Mobile Computing*, 13, 2013.

[31] Khalid Elgazzar, Hossam S. Hassanein, and Patrick Martin. Daas: Cloud-based mobile web service discovery. *Pervasive and Mobile Computing*, 13(0):67–84, 2014.

[32] Khalid Elgazzar, Patrick Martin, and Hossam Hassanein. *A Framework for Efficient Web Services Provisioning in Mobile Environments*, volume 95 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, chapter 16, pages 246–262. Springer Berlin Heidelberg, 2012.

[33] Anuraj Ennai and Siddhartha Bose. Mobilesoa: a service oriented web 2.0 framework for context-aware, lightweight and flexible mobile applications. In *Enterprise Distributed Object Computing Conference Workshops, 2008 12th*, pages 345–352. IEEE, 2008.

[34] Mobile-edge computing – introductory technical white paper. Technical report, European Telecommunications Standards Institute (ETSI), 2014.

[35] Facebook. `https://www.facebook.com`. Accessed: 2015-09-09.

[36] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, Information and Computer Science Department, University of California, Irvine, 2000.

[37] H. Flores, S. N. Srirama, and R. Buyya. Computational offloading or data binding? bridging the cloud infrastructure to the proximity of the mobile user. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, pages 10–18, 2014.

[38] Huber Flores. *Mobile Cloud Middleware*. Msc, Mathematics and Computer Science, University of Tartu, 2011.

[39] Huber Flores, Satish Narayana Srirama, and Carlos Paniagua. A generic middleware framework for handling process intensive hybrid cloud services from mobiles. In *Proceedings of the 9th International Conference on Ad-*

vances in Mobile Computing and Multimedia, pages 87–94, 2095715, 2011. ACM.

[40] Spring for Android. `http:// http://projects.spring.io/spring-android`. Accessed: 2015-09-09.

[41] Quartz Job Scheduler for Java. `http://quartz-scheduler.org`. Accessed: 2015-09-09.

[42] Cloud Foundry. `https://www.cloudfoundry.org`. Accessed: 2016-01-05.

[43] Foursquare. `https://foursquare.com`. Accessed: 2015-09-09.

[44] Ioana Giurgiu, Oriana Riva, and Gustavo Alonso. *Dynamic software deployment from clouds to mobile devices*, pages 394–414. Springer, 2012.

[45] Ioana Giurgiu, Oriana Riva, Dejan Juric, Ivan Krivulev, and Gustavo Alonso. Calling the cloud: enabling mobile phones as interfaces to cloud applications. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, pages 1–20, 1656987, 2009. Springer-Verlag New York, Inc.

[46] P. H. Gomes, N. L. Saldanha da Fonseca, and O. Carvalho Branquinho. Radio resource allocation and greenoperation for mobile access networks basedon radio-over-fiber. *Mobile Computing, IEEE Transactions on*, 13(4):894–906, 2014.

[47] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1):1–18, 2005.

[48] Kiryong Ha and Mahadev Satyanarayanan. Openstack++ for cloudlet deployment. Technical report, Carnegie Mellon University Pittsburgh, 2015.

[49] Bo Han, Weijia Jia, Ji Shen, and Man-Ching Yuen. *Context-Awareness in Mobile Web Services*, volume 3358 of *Lecture Notes in Computer Science*, chapter 62, pages 519–528. Springer Berlin Heidelberg, 2005.

[50] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, page 10 pp. vol.2, 2000.

[51] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger. Context-awareness on mobile devices - the hydrogen approach. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, page 10 pp., 2003.

[52] Jongyi Hong, Eui-Ho Suh, Junyoung Kim, and SuYeon Kim. Context-aware system for proactive personalized service based on context history. *Expert Systems with Applications*, 36(4):7448–7457, 2009.

[53] Liang Hongbin, L. X. Cai, Huang Dijiang, Shen Xuemin, and Peng Daiyuan. An smdp-based service model for interdomain resource allocation in mobile cloud networks. *Vehicular Technology, IEEE Transactions on*, 61(5):2222–2232, 2012.

[54] Matthew Horridge and Sean Bechhofer. The owl api: A java api for owl ontologies. *Semantic Web*, 2(1):11–21, 2011.

[55] Amazon AWS Mobile Hub. `https://aws.amazon.com/mobile`. Accessed: 2016-01-05.

[56] Gonzalo Huerta-Canepa and Dongman Lee. A virtual cloud computing provider for mobile devices. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, pages 1–5, 1810937, 2010. ACM.

[57] La Hyun Jung and Kim Soo Dong. A conceptual framework for provisioning context-aware mobile cloud services. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 466–473, 2010.

[58] Apple iCloud. `https://www.icloud.com`. Accessed: 2015-09-09.

[59] International Data Corporation (IDC). Worldwide saas and cloud software 2013–2017 forecast and 2012 vendor shares. `http://www.idc.com/getdoc.jsp?containerId=245084`. Accessed: 2014-07-09.

[60] XPP3/MXP1 XMLPULL Parser Implementation. `http://www.extreme.indiana.edu/xgws/xsoap/xpp/mxp1`. Accessed: 2015-09-09.

[61] Apache JMeter. `http://jmeter.apache.org`. Accessed: 2015-09-26.

[62] jsoup: Java HTML Parser. `http://jsoup.org`. Accessed: 2015-09-10.

[63] R. Kaewpuang, D. Niyato, Wang Ping, and E. Hossain. A framework for cooperative resource management in mobile cloud computing. *Selected Areas in Communications, IEEE Journal on*, 31(12):2685–2700, 2013.

[64] A. Kazi, R. Kazi, and R. Deters. Supporting the personal cloud. In *Cloud Computing Congress (APCloudCC), 2012 IEEE Asia Pacific*, pages 25–30, 2012.

[65] R. Kazi, Zhang Xiaobo, and R. Deters. Supporting apps in the personal cloud: Using websockets within hybrid apps. In *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*, pages 110–115, 2012.

[66] Outi Kivekas, J. Ollikainen, T. Lehtiniemi, and P. Vainikainen. Bandwidth, sar, and efficiency of internal mobile phone antennas. *Electromagnetic Compatibility, IEEE Transactions on*, 46(1):71–86, 2004.

[67] Matthias Klusch, Benedikt Fries, and Katia Sycara. Automated semantic web service discovery with owls-mx. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922, 1160796, 2006. ACM.

[68] T. Koponen and T. Virtanen. A service discovery: a service broker approach. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, page 7 pp., 2004.

[69] Panu Korpipää and Jani Mäntyjärvi. *An Ontology for Mobile Device Sensor-Based Context Awareness*, volume 2680 of *Lecture Notes in Computer Science*, chapter 37, pages 451–458. Springer Berlin Heidelberg, 2003.

[70] S. Kosta, A. Aucinas, Hui Pan, R. Mortier, and Zhang Xinwen. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953, 2012.

[71] D. Kovachev, Yu Tian, and R. Klamma. Adaptive computation offloading from mobile devices into the cloud. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 784–791, 2012.

[72] K. Kumar and Lu Yung-Hsiang. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, 2010.

[73] Sun Le, Dong Hai, and J. Ashraf. Survey of service description languages and their issues in cloud computing. In *Semantics, Knowledge and Grids (SKG), 2012 Eighth International Conference on*, pages 128–135, 2012.

[74] Youngki Lee, S. S. Iyengar, Chulhong Min, Younghyun Ju, Seungwoo Kang, Taiwoo Park, Jinwon Lee, Yunseok Rhee, and Junehwa Song. Mobicon: a mobile context-monitoring platform. *Commun. ACM*, 55(3):54–65, 2012.

[75] Bo Li, Li Yin, K. Y. Michael Wong, and Si Wu. An efficient and adaptive bandwidth allocation scheme for mobile wireless networks using an online local estimation technique. *Wirel. Netw.*, 7(2):107–116, 2001.

[76] Hongbin Liang, Dijiang Huang, and Daiyuan Peng. *On economic mobile cloud computing model*, pages 329–341. Springer, 2012.

[77] Xu Liang, Shen Xuemin, and J. W. Mark. Dynamic bandwidth allocation with fair scheduling for wcdma systems. *Wireless Communications, IEEE*, 9(2):26–32, 2002.

[78] Apache Commons Math Library. `http://commons.apache.org/proper/commons-math`. Accessed: 2015-09-09.

[79] Apache OpenNLP Library. `https://opennlp.apache.org`. Accessed: 2015-09-09.

[80] JScience Library. `http://jscience.org`. Accessed: 2015-09-09.

[81] JSON2HTML Library. `http://json2html.com`. Accessed: 2015-02-23.

[82] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. *NIST Cloud Computing Reference Architecture: Recommendations of the National Institute of Standards and Technology (Special Publication 500-292)*. CreateSpace Independent Publishing Platform, 2012.

[83] R. K. Lomotey and R. Deters. Reliable consumption of web services in a mobile-cloud ecosystem using rest. In *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*, pages 13–24, 2013.

[84] Michael Lones. Encrypting files with public key encryption in java. `http://www.macs.hw.ac.uk/~ml355/lore/pkencryption.htm`. Accessed: 2015-09-18.

[85] R. Lowe, P. Mandl, and M. Weber. Context directory: A context-aware service for mobile context-aware computing applications by the example of google android. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, pages 76–81, 2012.

[86] R. K. K. Ma and Wang Cho-Li. Lightweight application-level task migration for mobile cloud computing. In *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*, pages 550–557, 2012.

[87] R. K. K. Ma, Lam King Tin, and Wang Cho-Li. excloud: Transparent runtime support for scaling mobile applications in cloud. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 103–110, 2011.

[88] P. Massuthe, W. Reisig, and K. Schmidt. An operating guideline approach to the soa. *Annals Of Mathematics, Computing & Teleinformatics*, 1:35–43, 2005.

[89] Karan Mitra, Saguna Saguna, Christer Ahlund, and Daniel Granlund Lulea. M2c2: A mobility management system for mobile cloud computing. In *Wireless Communications and Networking Conference (WCNC), 2015 IEEE*, pages 1608–1613, 2015.

[90] V. S. K. Nagireddi and S. Mishra. An ontology based cloud service generic search engine. In *Computer Science and Education (ICCSE), 2013 8th International Conference on*, pages 335–340, 2013.

[91] Yuri Natchetoi, Viktor Kaufman, and Albina Shapiro. Service-oriented architecture for mobile applications. In *Proceedings of the 1st international workshop on Software architectures and mobility*, pages 27–32. ACM, 2008.

[92] Takayuki Nishio, Ryoichi Shinkuma, Tatsuro Takahashi, and Narayan B. Mandayam. Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud. In *Proceedings of the first international workshop on Mobile cloud computing & networking*, pages 19–26, 2492354, 2013. ACM.

[93] Google Now. `https://www.google.com/landing/now`. Accessed: 2015-09-09.

[94] Department of Computer Science and University of Washington Engineering. Xmldata repository. `http://www.cs.washington.edu/research/xmldatasets`. Accessed: 2015-09-09.

[95] OpenStack. `https://www.openstack.org`. Accessed: 2016-01-05.

[96] M. J. O'Sullivan and D. Grigoras. The cloud personal assistant for providing services to mobile clients. In *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*, pages 478–485, 2013.

[97] M. J. O'Sullivan and D. Grigoras. Mobile cloud application models facilitated by the cpa. In *MOBILe Wireless MiddleWARE, Operating Systems and Applications (Mobilware), 2013 International Conference on*, pages 120–128, 2013.

[98] M. J. O'Sullivan and D. Grigoras. User experience of mobile cloud applications - current state and future directions. In *Parallel and Distributed Computing (ISPDC), 2013 IEEE 12th International Symposium on*, pages 85–92, 2013.

[99] M. J. O'Sullivan and D. Grigoras. Mobile cloud contextual awareness with the cloud personal assistant. In *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*, pages 82–89, 2014.

[100] M. J. O'Sullivan and D. Grigoras. Delivering mobile cloud services to the user: Description, discovery, and consumption. In *Mobile Services (MS), 2015 IEEE International Conference on*, pages 49–56, 2015.

[101] M. J. O'Sullivan and D. Grigoras. Context aware mobile cloud services: A user experience oriented middleware for mobile cloud computing. In *Mobile Cloud Computing, Services, and Engineering (Mobile Cloud), 2016 IEEE 4th International Conference on*, To Appear, 2016.

[102] Michael J. O'Sullivan and D. Grigoras. Mobile cloud application models facilitated by the cpa. *EAI Endorsed Transactions on Scalable Information Systems*, 15(4), 2015.

[103] Michael J. O'Sullivan and Dan Grigoras. Integrating mobile and cloud resources management using the cloud personal assistant. *Simulation Modelling Practice and Theory*, 50:20–41, 2015.

[104] Web Ontology Language (OWL). `http://www.w3.org/2001/sw/wiki/OWL`. Accessed: 2016-01-05.

[105] Panagiotis Papakos, Licia Capra, and David S. Rosenblum. Volare: context-aware adaptive cloud service discovery for mobile systems. In *Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware*, pages 32–38, 1891706, 2010. ACM.

[106] Trepn Power Profiler. `https://developer.qualcomm.com/software/trepn-power-profiler`. Accessed: 2015-09-09.

[107] XMLPULL Project. `http://http://www.xmlpull.org`. Accessed: 2015-09-09.

[108] M. Raento, A. Oulasvirta, R. Petit, and H. Toivonen. Contextphone: a prototyping platform for context-aware mobile applications. *Pervasive Computing, IEEE*, 4(2):51–59, 2005.

[109] M Reza Rahimi, Nalini Venkatasubramanian, Sharad Mehrotra, and Athanasios V Vasilakos. On optimal and fair service allocation in mobile cloud computing. *arXiv:1308.4391*, 2013.

[110] P. Ramanathan, K. M. Sivalingam, P. Agrawal, and S. Kishore. Dynamic resource allocation schemes during handoff for mobile multimedia wireless networks. *Selected Areas in Communications, IEEE Journal on*, 17(7):1270–1283, 1999.

[111] Wei Ren, Linchen Yu, Ren Gao, and Feng Xiong. Lightweight and compromise resilient storage outsourcing with distributed secure accessibility in mobile cloud computing. *Tsinghua Science and Technology*, 16(5):520–528, 2011.

[112] Oriana Riva. Contory: a middleware for the provisioning of context information on smart phones. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, pages 219–239, 1516002, 2006. Springer-Verlag New York, Inc.

[113] F. J. Rivas Tocado, A. Diaz Zayas, and P. Merino Gomez. Characterizing traffic performance in cellular networks. *Internet Computing, IEEE*, 18(1):12–19, 2014.

[114] Jini Network Technology Specifications/Apache River. `https://river.apache.org/doc/spec-index.html`. Accessed: 2016-01-05.

[115] Z. Sanaei, S. Abolfazli, A. Gani, and M. Shiraz. Sami: Service-based arbitrated multi-tier infrastructure for mobile cloud computing. In *Commu-*

nications in China Workshops (ICCC), 2012 1st IEEE International Conference on, pages 14–19, 2012.

[116] J. Sankaranarayanan, H. Hacigumus, and J. Tatemura. Cosmos: A platform for seamless mobile services in the cloud. In *Mobile Data Management (MDM), 2011 12th IEEE International Conference on*, volume 1, pages 303–312, 2011.

[117] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.

[118] M. Satyanarayanan, R. Schuster, M. Ebling, G. Fettweis, H. Flinck, K. Joshi, and K. Sabnani. An open ecosystem for mobile-cloud convergence. *Communications Magazine, IEEE*, 53(3):63–70, 2015.

[119] Mahadev Satyanarayanan. Mobile computing: the next decade. *SIGMOBILE Mob. Comput. Commun. Rev.*, 15(2):2–10, 2011.

[120] Aaron Schulman, Vishnu Navda, Ramachandran Ramjee, Neil Spring, Pralhad Deshpande, Calvin Grunewald, Kamal Jain, and Venkata N. Padmanabhan. Bartendr: a practical approach to energy-aware cellular data scheduling. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, pages 85–96. ACM, 2010.

[121] Spring Security. `http://projects.spring.io/spring-security`. Accessed: 2015-09-18.

[122] Robert Sedgewick and Kevin Wayne. Queens java code. `http://http://introcs.cs.princeton.edu/java/23recursion/Queens.java.html`. Accessed: 2015-09-26.

[123] Q. Z. Sheng and B. Benatallah. Contextuml: a uml-based modeling language for model-driven development of context-aware web services. In *Mobile Business, 2005. ICMB 2005. International Conference on*, pages 206–212, 2005.

[124] P. Simoens, F. De Turck, B. Dhoedt, and P. Demeester. Remote display solutions for mobile cloud computing. *Computer*, 44(8):46–53, 2011.

[125] P. Simoens, P. Praet, B. Vankeirsbilck, J. De Wachter, L. Deboosere, F. De Turck, B. Dhoedt, and P. Demeester. Design and implementation of a hybrid remote display protocol to optimize multimedia experience on

thin client devices. In *Telecommunication Networks and Applications Conference, 2008. ATNAC 2008. Australasian*, pages 391–396, 2008.

[126] Apple Siri. `https://support.apple.com/en-ie/HT204389`. Accessed: 2015-09-09.

[127] OAuth Community Site. `http://oauth.net`. Accessed: 2015-09-09.

[128] Microsoft SkyDrive. `https://onedrive.live.com`. Accessed: 2015-09-09.

[129] Common Object Request Broker Architecture (CORBA) Specification. `http://www.omg.org/spec/CORBA`. Accessed: 2016-01-05.

[130] Simple Object Access Protocol (SOAP) W3C Specification. `http://www.w3.org/TR/soap`. Accessed: 2015-09-09.

[131] Satish Narayana Srirama, Matthias Jarke, and Wolfgang Prinz. Mwsmf: a mediation framework realizing scalable mobile web service provisioning. In *Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, pages 1–7, 1361546, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[132] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Workshop Proceedings*, 2004.

[133] V. Suraci, S. Mignanti, and A. Aiuto. Context-aware semantic service discovery. In *Mobile and Wireless Communications Summit, 2007. 16th IST*, pages 1–5, 2007.

[134] Cisco Systems. Cisco global cloud index: Forecast and methodology 2012–2017 whitepaper. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html`. Accessed: 2015-09-10.

[135] Renne Tergujeff, Jyrki Haajanen, Juha Leppänen, and Santtu Toivonen. Mobile soa: Service orientation on lightweight mobile devices. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 1224–1225. IEEE, 2007.

[136] N. Tolia, D. G. Andersen, and M. Satyanarayanan. Quantifying interactive user experience on thin clients. *Computer*, 39(3):46–52, 2006.

[137] Hong-Linh Truong and Schahram Dustdar. A survey on context-aware web service systems. *International Journal of Web Information Systems*, 5(1):5–31, 2009.

[138] W-K. Tsai, X. Sun, and J. Balasooriya. Service-oriented cloud computing architecture. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 684–689, 2010.

[139] Twitter. `https://twitter.com`. Accessed: 2015-09-09.

[140] Do Van Thanh and Ivar Jørstad. A service-oriented architecture framework for mobile services. In *Telecommunications, 2005. advanced industrial conference on telecommunications/service assurance with partial and intermittent resources conference/e-learning on telecommunications workshop. aict/sapir/elete 2005. proceedings*, pages 65–70. IEEE, 2005.

[141] Tim Verbelen, Pieter Simoens, Filip De Turck, and Bart Dhoedt. Cloudlets: bringing the cloud to the mobile user. In *Proceedings of the third ACM workshop on Mobile cloud computing and services*, pages 29–36, 2307858, 2012. ACM.

[142] Claudia Villalonga, Martin Strohbach, Niels Snoeck, Michael Sutterer, Mariano Belaunde, Ernö Kovacs, Anna V Zhdanova, Laurent Walter Goix, and Olaf Droegehorn. Mobile ontology: Towards a standardized semantic model for the mobile domain. In *Service-Oriented Computing-ICSOC 2007 Workshops*, pages 248–257. Springer, 2009.

[143] Qian Wang. *Mobile Cloud Computing*. Msc, Department of Computer Science, University of Saskatchewan, 2011.

[144] Qian Wang and Ralph Deters. Soa's last mile-connecting smartphones to the service cloud. In *Proceedings of the 2009 IEEE International Conference on Cloud Computing*, pages 80–87. IEEE Computer Society, 2009.

[145] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

[146] Huijun Wu and Dijiang Huang. Mosec: Mobile-cloud service composition. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2015 3rd IEEE International Conference on*, pages 177–182, 2015.

[147] Yunqi Ye, Nisha Jain, Longsheng Xia, Suhas Joshi, I Yen, Farokh Bastani, Kenneth L Cureton, and Mark K Bowler. A framework for qos and

power management in a service cloud environment with mobile devices. In *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on*, pages 236–243. IEEE, 2010.

[148] Pengfei Yuan, Yao Guo, and Xiangqun Chen. Uniport: A uniform programming support framework for mobile cloud computing. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2015 3rd IEEE International Conference on*, pages 71–80, 2015.

[149] Irene Zhang, Adriana Szekeres, Dana Van Aken, and Isaac Ackerman. Customizable and extensible deployment for mobile/cloud applications. In *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.

[150] Zhou Zhibin and Huang Dijiang. Efficient and secure data storage operations for mobile cloud computing. In *Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm)*, pages 37–45, 2012.

[151] J. Zhu, M. Oliya, H. K. Pung, and Wong Wai Choong. Laspd: A framework for location-aware service provision and discovery in mobile environments. In *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific*, pages 218–225, 2010.

Appendix A

User Study Survey Responses

The complete responses collected from the nine out of ten volunteers in the user study are presented here. These volunteers consisted of undergraduate, masters, and PhD research students in the Department of Computer Science, at University College Cork. Each participant downloaded the CAMCS Client on his/her mobile device, registered with CAMCS (obtaining a CPA), and used it to complete tasks for a minimum of four days, up until a maximum time of one week.

A.1 Part 1: Mobile Experience

Question 1: Before taking part in this study, how familiar were you with the concept of "mobile cloud computing"?

Responses: 9

- Very Familiar: 3 (33.33%)
- Familiar: 4 (44.44%)
- Unfamiliar: 2 (22.22%)
- Very Unfamiliar: 0 (0%)

Question 2: Are you aware of how "apps" installed on your mobile device use resources such as battery power, storage space, memory, and network data?

Responses: 9

- Very Aware: 4 (44.44%)

- Aware: 4 (44.44%)
- Undecided: 0 (0%)
- Unaware: 1 (11.11%)
- Very Unaware: 0 (0%)

Question 3: Are there applications you would like to be able to use on your mobile device, that cannot run on your mobile device because of the lack of resources, when compared with a desktop PC or laptop? If yes, please give details.

Responses: 6

- "No" - 3 Responses
- "programming; compilers; program testing etc"
- "games and video intensive apps"
- "Yes - to a certain extent. I have apps that slow down my phone most likely due lack of memory."

A.2 Part 2: CAMCS and Your CPA

Question 4: Please describe how easy/difficult you found the CAMCS Client to use?

Responses: 9

- Simple: 0 (0%)
- Very Easy: 3 (33.33%)
- Easy: 6 (66.67%)
- Not Sure: 0 (0%)
- Difficult: 0 (0%)
- Very Difficult: 0 (0%)
- Impossible: 0 (0%)

Question 5: Did you find your CPA to be a useful tool for completing work on a day-to-day basis?

Responses: 9

- Very Useful: 1 (11.11%)
- Useful: 7 (77.78%)
- Undecided: 1 (11.11%)
- Not Useful: 0 (0%)
- Useless: 0 (0%)

Question 6: In this study, there were only a few services to choose from. With reference to the previous question, if there were more services to choose from, how useful do you think you would find using a CPA to complete your work on a day-to-day basis?

Responses: 9

- Very Useful: 4 (44.44%)
- Useful: 5 (55.56%)
- Undecided: 0 (0%)
- Not Useful: 0 (0%)
- Useless: 0 (0%)

Question 7: Please give examples of services you believe you may find useful on a daily basis, that could be used by your CPA? If you cannot think of any, please state in the box if you are just unsure, or if you believe that the CPA, in its current form, is not capable of working with services that you would find useful.

Responses: 9

- "For example, Opening daily news letter online every morning"
- "just unsure; I think it has great potential"
- "A weather service Translation Service Dictionary service Daily news"
- "news apps postage tracking notifications"
- "Reminders, Bill Enquiries"
- "- Traffic levels for going to/from work. - Buy bus tickets for regular commutes (e.g. if you visit friends/family in another county every other weekend). Also download the ticket to your phone so you have it ready to display to driver. - More refined "places of interest" service - A daily newsletter of selected friends updates on facebook (scrolling through

newsfeed can be tiresome with a large friendlist) - If your location changes from your "home" location, e.g. if you leave Cork - an update of places of interest and events (possibly facebook) in your new location"

- "Daily google calendar (I know its there just didnt use it) A social networking summary of the day (in the morning what happened while you were asleep)"
- "unsure" - 2 Responses

Question 8: If the CPA could work with services that you would find useful on a day-to-day basis, how likely would you be to use CAMCS (or a similar service) as a consumer product?

Responses: 9

- Very Likely: 2 (22.22%)
- Likely: 5 (55.56%)
- Undecided: 2 (22.22%)
- Unlikely: 0 (0%)
- Very Unlikely: 0 (0%)

Question 9: CAMCS aims to provide useful, relevant, and personalised services/information, to your mobile device, by making use of mobile cloud services. Do you feel that the use of mobile cloud services, could take the place of the current "app store" model on mobile devices?

Responses: 9

- Yes: 2 (22.22%)
- In Some Cases: 4 (44.44%)
- Undecided: 2 (22.22%)
- In a Few Cases: 1 (11.11%)
- No: 0 (0%)

Question 10: Did you find the descriptions of the services and their required data parameters to be user-friendly and informative? This applies to the steps in the CAMCS Client where you chose a service for your task, and typed-in the data required.

Responses: 9

- Very Informative: 0 (0%)
- Informative: 4 (44.44%)
- Undecided: 4 (44.44%)
- Uninformative: 1 (11.11%)
- Very Uninformative: 0 (0%)

A.3 Part 3: Contextual Awareness Support with the CPA

Question 11: Would the storage of personal context data (such as, your location history, current activity, education/work history) with the CPA in the cloud be a significant privacy concern to you?

Responses: 9

- Very Concerned: 1 (11.11%)
- Concerned: 2 (22.22%)
- A Little Concerned: 3 (33.33%)
- Undecided: 0 (0%)
- A Little Unconcerned: 1 (11.11%)
- Unconcerned: 2 (22.22%)
- Very Unconcerned: 0 (0%)

Question 12: Would the sharing of your context data collected collected by your CPA, with third-party mobile cloud services, for the purposes of personalisation, be a significant privacy concern to you?

Responses: 8

- Very Concerned: 1 (12.5%)
- Concerned: 2 (25%)
- A Little Concerned: 3 (37.5%)
- Undecided: 0 (0%)

- A Little Unconcerned: 2 (25%)
- Unconcerned: 0 (0%)
- Very Unconcerned: 0 (0%)

Question 13: Did you try the Foursquare "Places of Interest" Service during the study? (The Tourism Service)

Responses: 9

- Yes: 4 (44.44%)
- No: 5 (55.56%)

Questions 14 and 15 were only to be answered by those who answered "Yes" to Question 13.

Question 14: Did you find the CAMCS model of asking for your consent to use your collected context data with the third-party service (Foursquare), to be re-assuring in terms of the privacy of the data?

Responses: 4

- Strongly Re-assuring: 0 (0%)
- Re-assuring: 4 (100%)
- Undecided: 0 (0%)
- Not Re-assuring: 0 (0%)
- Strongly Not Re-assuring: 0 (0%)

Question 15: Did you find the capabilities of CAMCS to share your context data with third-party services, for the purposes of task result personalisation, to be a useful feature?

Responses: 4

- Very Useful: 2 (50%)
- Useful: 2 (50%)
- Undecided: 0 (0%)
- Not Useful: 0 (0%)
- Useless: 0 (0%)

A.4 Part 4: User Experience Perceptions

Question 16: Did you notice any adverse effects on your mobile device from the installation of the CAMCS Client (slow/unresponsive system, high battery power drain, frequent network activity)?

Responses: 9

- Yes: 1 (11.11%)
- No: 8 (88.89%)

Question 17: CAMCS aims to deliver an integrated user experience of mobile cloud applications. This means it was designed to operate in a disconnected fashion, with low data transfer requirements. It was hoped that this design would result in low resource usage, such as low demand for battery power, memory, and storage space on your mobile. To what extent do you feel CAMCS meets the integrated user experience design goal?

Responses: 9

- CAMCS strongly achieves this goal: 3 (33.33%)
- CAMCS achieves this goal: 4 (44.44%)
- Undecided: 2 (22.22%)
- CAMCS does not achieve this goal: 0 (0%)
- CAMCS strongly does not achieve this goal: 0 (0%)

Appendix B

OWL Context Ontology Example

Listing B.1: OWL User Current Context Example for Location Context.

```
<rdf:RDF>
    <owl:ObjectProperty rdf:about="http://cs1.ucc.ie/~
        mos10/camcs/ontologies/context/context.owl#
        hasLocation"/>

    <owl:DatatypeProperty rdf:about="http://cs1.ucc.ie/~
        mos10/camcs/ontologies/context/context.owl#InHome
        "/>
    <owl:DatatypeProperty rdf:about="http://cs1.ucc.ie/~
        mos10/camcs/ontologies/context/context.owl#
        LocationLatitude"/>
    <owl:DatatypeProperty rdf:about="http://cs1.ucc.ie/~
        mos10/camcs/ontologies/context/context.owl#
        LocationLongitude"/>
    <owl:DatatypeProperty rdf:about="http://cs1.ucc.ie/~
        mos10/camcs/ontologies/context/context.owl#
        LocationPlaceName"/>
    <owl:DatatypeProperty rdf:about="http://cs1.ucc.ie/~
        mos10/camcs/ontologies/context/context.owl#
        LocationTimestamp"/>

    <owl:Class rdf:about="http://cs1.ucc.ie/~mos10/camcs/
        ontologies/context/context.owl#Location"/>
    <owl:Class rdf:about="http://ontology.ist-spice.org/
```

```
    mobile-ontology/1/0/presence/0/presence.owl#User
    "/>

<owl:NamedIndividual rdf:about="http://cs1.ucc.ie/~
    mos10/camcs/ontologies/context/context.owl#
    Location">
    <rdf:type rdf:resource="http://cs1.ucc.ie/~mos10/
        camcs/ontologies/context/context.owl#Location
        "/>
    <context:LocationLongitude rdf:datatype="http://
        www.w3.org/2001/XMLSchema#string">-2.47061</
        context:LocationLongitude>
    <context:LocationTimestamp rdf:datatype="http://
        www.w3.org/2001/XMLSchema#string">2014.10.11
        12:45:59 GMT</context:LocationTimestamp>
    <context:LocationLatitude rdf:datatype="http://
        www.w3.org/2001/XMLSchema#string">67.89797</
        context:LocationLatitude>
    <context:LocationPlaceName rdf:datatype="http://
        www.w3.org/2001/XMLSchema#string">Galway,
        Ireland</context:LocationPlaceName>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="http://cs1.ucc.ie/~
    mos10/camcs/ontologies/users#test25">
    <rdf:type rdf:resource="http://ontology.ist-spice
        .org/mobile-ontology/1/0/presence/0/presence.
        owl#User"/>
    <context:InHome rdf:datatype="http://www.w3.org
        /2001/XMLSchema#string">false</context:InHome>
    <context:hasLocation rdf:resource="http://cs1.ucc
        .ie/~mos10/camcs/ontologies/context/context.
        owl#Location"/>
</owl:NamedIndividual>

<!-- Generated by the OWL API (version 1.3.8.1099)
    http://owlapi.sourceforge.net -->
```

```
</rdf:RDF>
```

Appendix C

User Oriented Mobile Cloud Service Description Examples

C.1 Google Calendar Service

This is the service description for the Google Calendar service, demonstrating a single-call service which takes required and optional data parameters.

Listing C.1: Google Calendar Service Description.

```
{
  "service":{
    "type":"rest",
    "id":null,
    "name":"Calendar",
    "provider":"Google",
    "description":"With Google's free online calendar, it
       's easy to keep track of life's important events
       all in one place",
    "descriptionTerms":[
      "calendar",
      "google",
      "appointments",
      "schedule",
      "cal",
      "organiser"
    ],
```

```
"usesContextData":false,
"type":"rest",
"operations":[
  {
    "name":"List Events",
    "description":"Returns events on the specified
       calendar",
    "resultTemplateUrl":"http://cs1.ucc.ie/~mos10/
       camcs/google_calendar_result_template.html",
    "serviceMethod":{
      "path":"https://www.googleapis.com/calendar/v3/
         calendars/$calendarId/events?key=REMOVED&
         singleEvents=true",
      "type":"GET",
      "dataParameters":[
        {
          "name":"Earliest Time",
          "pathPlaceholderText":"timeMin",
          "paramType":"URL_PARAM",
          "optional":true,
          "description":"Lower bound (inclusive) for
             an event's end time to filter by.
             Optional. The default is not to filter
             by end time.",
          "value":null,
          "dataParamType":"DATETIME"
        },
        {
          "name":"Query",
          "pathPlaceholderText":"q",
          "paramType":"URL_PARAM",
          "optional":true,
          "description":"Free text search terms to
             find events that match these terms in
             any field, except for extended
             properties. Optional.",
          "value":null,
```

```
      "dataParamType":"STRING"
    },
    {
      "name":"Calendar ID",
      "pathPlaceholderText":"calendarId",
      "paramType":"URL_PARAM",
      "optional":false,
      "description":"The Id of your Calendar (
        normally your gmail address)",
      "value":null,
      "dataParamType":"STRING"
    },
    {
      "name":"Latest Time",
      "pathPlaceholderText":"timeMax",
      "paramType":"URL_PARAM",
      "optional":true,
      "description":"Upper bound (exclusive) for
        an event's start time to filter by.
        Optional. The default is not to filter
        by start time.",
      "value":null,
      "dataParamType":"DATETIME"
    }
  ],
  "contextParameters":[],
  "dataParameterDescriptions":[
    {
      "name":"Earliest Time",
      "description":"Lower bound (inclusive) for
        an event's end time to filter by.
        Optional. The default is not to filter
        by end time.",
      "defaultValue":null,
      "dataType":"datetime",
      "optional":true,
      "options":[]
```

```
        },
        {
          "name":"Query",
          "description":"Free text search terms to
             find events that match these terms in
             any field, except for extended
             properties. Optional.",
          "defaultValue":null,
          "dataType":"string",
          "optional":true,
          "options":[]
        },
        {
          "name":"Calendar ID",
          "description":"The Id of your Calendar (
             normally your gmail address)",
          "defaultValue":null,
          "dataType":"string",
          "optional":false,
          "options":[]
        },
        {
          "name":"Latest Time",
          "description":"Upper bound (exclusive) for
             an event's start time to filter by.
             Optional. The default is not to filter
             by start time.",
          "defaultValue":null,
          "dataType":"datetime",
          "optional":true,
          "options":[]
        }
      ],
      "contextParameterDescriptions":[]
    },
    "serviceFlow":null,
    "descriptiveTags":[
```

```
            "events",
            "list",
            "items",
            "meeting",
            "schedule"
          ]
        }
      ]
    }
}
```

C.2 Foursquare Tourism/Places of Interest Service

This is the service description for the Foursquare Tourism/Places of Interest service, demonstrating a single-call service which takes required context parameters, of Location and DateTimeContexts.

Listing C.2: Foursquare Tourism/Places of Interest Service Description.

```
{
  "service":{
    "type":"rest",
    "id":null,
    "name":"Venue Service",
    "provider":"Foursquare",
    "description":"Location information from Foursquare",
    "descriptionTerms":[
      "foursquare",
      "location",
      "poi"
    ],
    "usesContextData":true,
    "type":"rest",
    "operations":[
      {
        "name":"Explore",
        "description":"Returns a list of recommended
          venues near the current location.",
```

```
"resultTemplateUrl":"http://cs1.ucc.ie/~mos10/
    camcs/foursquare_result_template.html",
"serviceMethod":{
  "path":"https://api.foursquare.com/v2/venues/
     explore?near=$location&client_id=REMOVED&
     client_secret=REMOVED&v=$date",
  "type":"GET",
  "dataParameters":[],
  "contextParameters":[
    {
      "name":"DateTime",
      "pathPlaceholderText":"date",
      "paramType":"URL_PARAM",
      "optional":false,
      "contextType":"DATETIME",
      "explanation":"Foursquare requires the
         current date and time to secure your
         information request.",
      "optionString":"yyyyMMdd"
    },
    {
      "name":"name",
      "pathPlaceholderText":"location",
      "paramType":"URL_PARAM",
      "optional":false,
      "contextType":"LOCATION",
      "explanation":"Foursquare requires your
         location in order to discover places of
         interest nearby.",
      "optionString":null
    }
  ],
  "dataParameterDescriptions":[],
  "contextParameterDescriptions":[
    {
      "name":"DateTime",
      "contextType":"DateTime",
```

```
            "explanation":"Foursquare requires the
               current date and time to secure your
               information request.",
            "optional":false
          },
          {
            "name":"name",
            "contextType":"Location",
            "explanation":"Foursquare requires your
               location in order to discover places of
               interest nearby.",
            "optional":false
          }
        ]
      },
      "serviceFlow":null,
      "descriptiveTags":[
        "explore",
        "venue"
      ]
    }
  ]
 }
}
```

C.3 Lufthansa Flight Information Service

This is the service description for the Lufthansa Flight Information service, demonstrating a complex Service Flow, consisting of three Flow Steps. Steps 1 and 3 are data entry steps, step 2 is a selection step.

Listing C.3: Lufthansa Flight Information Service Description.

```
{
  "service":{
    "type":"rest",
    "id":null,
```

```
"name":"Flight Information Service",
"provider":"Lufthansa",
"description":"Find information on flights by
   Lufthansa, and affiliated Star Alliance carriers
   .",
"descriptionTerms":[
  "flight",
  "flights",
  "status",
  "travel",
  "airline",
  "airlines",
  "lufthansa",
  "star alliance",
  "holiday"
],
"usesContextData":false,
"type":"rest",
"operations":[
  {
    "name":"Flight Status",
    "description":"Find the status of a flight,
       either by flight number, or by route",
    "resultTemplateUrl":"http://cs1.ucc.ie/~mos10/
       camcs/lufthansa_flight_status_result_template.
       html",
    "serviceMethod":{
      "path":"http://52.2.5.6:8080/flight/status/
         $departure/$arrival/$number",
      "type":"GET",
      "dataParameters":[
        {
          "name":"Arrival Airport",
          "pathPlaceholderText":"arrival",
          "paramType":"URL_PARAM",
          "optional":false,
```

```
      "description":"The IATA code of the arrival
          airport (e.g. JFK)",
      "value":null,
      "dataParamType":"STRING"
    },
    {
      "name":"Flight Number",
      "pathPlaceholderText":"number",
      "paramType":"URL_PARAM",
      "optional":false,
      "description":"The flight number including
        carrier code and any suffix, e.g. LH400
        ",
      "value":null,
      "dataParamType":"STRING"
    },
    {
      "name":"Departure Airport",
      "pathPlaceholderText":"departure",
      "paramType":"URL_PARAM",
      "optional":false,
      "description":"The IATA code of the
        departure airport (e.g. FRA)",
      "value":null,
      "dataParamType":"STRING"
    }
  ],
  "contextParameters":[],
  "dataParameterDescriptions":[
    {
      "name":"Arrival Airport",
      "description":"The IATA code of the arrival
          airport (e.g. JFK)",
      "defaultValue":null,
      "dataType":"string",
      "optional":false,
      "options":[]
```

```
        },
        {
          "name":"Flight Number",
          "description":"The flight number including
             carrier code and any suffix, e.g. LH400
             ",
          "defaultValue":null,
          "dataType":"string",
          "optional":false,
          "options":[]
        },
        {
          "name":"Departure Airport",
          "description":"The IATA code of the
             departure airport (e.g. FRA)",
          "defaultValue":null,
          "dataType":"string",
          "optional":false,
          "options":[]
        }
      ],
      "contextParameterDescriptions":[]
    },
    "serviceFlow":{
      "flowSteps":[
        {
          "stepNumber":3,
          "name":"Flight Date Search Selection",
          "description":"Please provide the date of
             the flight that you would like to search
              for.",
          "flowStepType":"DATA_ENTRY",
          "serviceMethod":{
            "path":"http://52.2.5.6:8080/flight/
               status/result/$date",
            "type":"GET",
            "dataParameters":[
```

```
            {
              "name":"Date",
              "pathPlaceholderText":"date",
              "paramType":"URL_PARAM",
              "optional":false,
              "description":"Departure date in
                 local time of departure airport (
                 YYYY-MM-DD)",
              "value":null,
              "dataParamType":"STRING"
            }
          ],
          "contextParameters":[],
          "dataParameterDescriptions":[
            {
              "name":"Date",
              "description":"Departure date in
                 local time of departure airport (
                 YYYY-MM-DD)",
              "defaultValue":null,
              "dataType":"string",
              "optional":false,
              "options":[]
            }
          ],
          "contextParameterDescriptions":[]
        }
      },
      {
        "stepNumber":2,
        "name":"Search Method",
        "description":"Would you like to search for
            your flight by Flight Number or by
           Route?",
        "flowStepType":"SELETION",
        "serviceMethod":{
```

```
        "path":"http://52.2.5.6:8080/flight/
           status/$option",
        "type":"POST",
        "dataParameters":[
          {
            "name":"Search Option",
            "pathPlaceholderText":"option",
            "paramType":"URL_PARAM",
            "optional":false,
            "description":"Select the Search
               Option",
            "value":null,
            "dataParamType":"CHECKBOX"
          }
        ],
        "contextParameters":[],
        "dataParameterDescriptions":[
          {
            "name":"Search Option",
            "description":"Select the Search
               Option",
            "defaultValue":null,
            "dataType":"checkbox",
            "optional":false,
            "options":[]
          }
        ],
        "contextParameterDescriptions":[]
      }
    },
    {
      "stepNumber":1,
      "name":"Flight Information",
      "description":"Departure and Arrival, or
         Flight Number information is required to
          find your flight",
      "flowStepType":"DATA_ENTRY",
```

```
"serviceMethod":{
  "path":"http://52.2.5.6:8080/flight/
     status/$departure/$arrival/$number",
  "type":"GET",
  "dataParameters":[
    {
      "name":"Arrival Airport",
      "pathPlaceholderText":"arrival",
      "paramType":"URL_PARAM",
      "optional":false,
      "description":"The IATA code of the
         arrival airport (e.g. JFK)",
      "value":null,
      "dataParamType":"STRING"
    },
    {
      "name":"Flight Number",
      "pathPlaceholderText":"number",
      "paramType":"URL_PARAM",
      "optional":false,
      "description":"The flight number
         including carrier code and any
         suffix, e.g. LH400",
      "value":null,
      "dataParamType":"STRING"
    },
    {
      "name":"Departure Airport",
      "pathPlaceholderText":"departure",
      "paramType":"URL_PARAM",
      "optional":false,
      "description":"The IATA code of the
         departure airport (e.g. FRA)",
      "value":null,
      "dataParamType":"STRING"
    }
  ],
```

```
            "contextParameters":[],
            "dataParameterDescriptions":[
              {
                "name":"Arrival Airport",
                "description":"The IATA code of the
                   arrival airport (e.g. JFK)",
                "defaultValue":null,
                "dataType":"string",
                "optional":false,
                "options":[]
              },
              {
                "name":"Flight Number",
                "description":"The flight number
                   including carrier code and any
                   suffix, e.g. LH400",
                "defaultValue":null,
                "dataType":"string",
                "optional":false,
                "options":[]
              },
              {
                "name":"Departure Airport",
                "description":"The IATA code of the
                   departure airport (e.g. FRA)",
                "defaultValue":null,
                "dataType":"string",
                "optional":false,
                "options":[]
              }
            ],
            "contextParameterDescriptions":[]
          }
        }
      ]
    },
    "descriptiveTags":[
```

```
            "flight",
            "flights",
            "status",
            "travel",
            "airline",
            "airlines",
            "lufthansa",
            "star alliance",
            "holiday"
          ]
        }
      ]
    }
}
```