CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Context aware mobile cloud services: a user experience oriented middleware for mobile cloud computing
Authors	O'Sullivan, Michael J.;Grigoras, Dan
Publication date	2016-03-29
Original Citation	O'Sullivan, M. J. and Grigoras, D. (2016) "Context aware mobile cloud services: a user experience oriented middleware for mobile cloud computing", 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud). Oxford, United Kingdom, 29 March-1 April.
Type of publication	Conference item
Rights	© Copyright 2016 IEEE. All rights reserved. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Download date	2026-11-03 03:06:06
Item downloaded from	https://hdl.handle.net/10468/2484

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Context Aware Mobile Cloud Services: A User Experience Oriented Middleware for Mobile Cloud Computing

Michael J. O'Sullivan*
IBM - Ireland,
Airport Business Park, Cork, Ireland
MichaelOSullivan@ie.ibm.com

Dan Grigoras
Department of Computer Science,
University College Cork, Cork, Ireland
grigoras@cs.ucc.ie

***Abstract*—Existing research on implementing the mobile cloud computing paradigm is typically based on offloading demanding computation from mobile devices to cloud-based servers. A continuous, high quality connection to the cloud infrastructure is normally required, with frequent high-volume data transfer, which can have a detrimental impact on the user experience of the application or service. In this paper, the Context Aware Mobile Cloud Services (CAMCS) middleware is presented as a solution that can deliver an integrated user experience of the mobile cloud to users. Such an experience respects the resource limitations of the mobile device. This is achieved by the Cloud Personal Assistant (CPA), the user's trusted representative within CAMCS, which completes user-assigned tasks using existing cloud-based services, with an asynchronous, disconnected approach. A thin client mobile application, the CAMCS Client, allows the mobile user to send descriptions of tasks to his/her CPA, and view task results saved at the CPA, when convenient. The design and implementation of the middleware is presented, along with results of experimental evaluation on Amazon EC2. The resource usage of the CAMCS client is also studied. Analysis shows that CAMCS delivers an integrated user experience of mobile cloud applications and services.**

Keywords—mobile, cloud, middleware, distributed system, services, user experience

I. INTRODUCTION

While there are many different research approaches to implementing the mobile cloud computing (MCC) paradigm, a common aspect includes offloading compute-intensive work to cloud infrastructure. Existing research, particularly in areas related to Cloudlets [1] and code-offloading techniques [2] have shown promising results and exciting use-cases. However, these approaches typically come with their own requirements and resource costs, such as the presence of a continuous high-quality network connection to the cloud infrastructure, which can be difficult to achieve in practice. The data connection is often used frequently to transfer considerable data payloads between the mobile device and the cloud, in the form of application components, code-bases, or even virtual machine outputs. The requirement of a continuous connection, and frequent data transfer, can drain the battery of the mobile device quickly. These requirements result in a detrimental user experience of the mobile cloud application.

In our previous work [3], requirements were outlined that should be implemented by future work, in order for MCC solutions to provide an integrated user experience of the mobile cloud; such an experience will respect the resource limitations of the mobile device (a more complete definition is presented in the previous work). The reference architecture for a middleware realising this goal was also presented. This paper presents the complete architecture and implementation of Context Aware Mobile Cloud Services (CAMCS), a mobile cloud middleware solution that has been designed to deliver cloud-based services to mobile users, while respecting the goal of providing an integrated user experience of mobile cloud applications and services. CAMCS delivers such an experience by means of an asynchronous, disconnected approach, to MCC. This is realised by the Cloud Personal Assistant (CPA) [4]. Each user of CAMCS is provided with his/her own CPA. The CPA is the trusted, third party representative of the mobile user, which completes user-assigned tasks, received as descriptions from the mobile device, by working with cloud-based services.

By means of the CAMCS Client, an Android-based thin-client mobile application, the mobile user can request his/her CPA to complete a task, given a name and description. CAMCS engages in service discovery on behalf of CPAs, to find an appropriate service to complete the task. The mobile user can then choose which discovered service they deem suitable for completing the task, provide input parameters, and the CPA will work with that cloud service to complete the work and save the result. The mobile user can view the result at their leisure. As the CPA is responsible for completing the task, the user is free to disconnect from the cloud; no continuous connection is required, and there is no requirement for frequent or high-volume data transfer.

In this paper, the following contributions are made: the finalised design and implementation of the complete CAMCS middleware is presented. Results of the first detailed experimental evaluation of the performance of the complete system are also provided, while CAMCS was deployed on Amazon EC2. The first resource usage evaluation for the CAMCS Client running on an Android mobile device is also provided. Finally, an original analysis is presented to determine if CAMCS meets the requirements outlined in our previous work for delivering an integrated user experience of MCC applications and services.

*Research completed while undertaking PhD study at the Department of Computer Science, University College Cork.

The remainder of the paper is organized as follows: Section II introduces CAMCS and its features. Section III presents the design of CAMCS, while Section IV presents the results of the experimental evaluation of the CAMCS middleware, and the CAMCS Client. An analysis of CAMCS against the integrated user experience requirements is presented in Section V. Related work is presented in Section VI, followed by the conclusions in Section VII.

II. Context Aware Mobile Cloud Services

The reference middleware architecture presented in [3] has been developed into the complete CAMCS solution now described here. The various features and components originally proposed were developed to enable the user experience goal of MCC in a personalised way. These individual features have been the subject of previous works, but are now described here briefly for completeness.

A. Cloud Personal Assistant Task Models

CAMCS provides two user task models for CPAs, which were described in a previous work [5]. The first is the user-initiated task; the user creates a task with a name and description using the CAMCS Client installed on his/her mobile device. The task description is then sent to his/her CPA in CAMCS to begin the work required. Tasks finish when the CPA receives a result for the task from the cloud-based service, which is stored with the CPA. CAMCS notifies the user of task completion with a Push notification. The user can then fetch the result from his/her CPA for viewing on the CAMCS Client. The second model is automatic-task execution. Here, the CPA can take the initiative to complete work for the mobile user without an explicit request. Users can set tasks to run automatically at their CPA, specifying days of the weeks and times when the CPA should run a task again. This model supports the goal of disconnected operation.

CPAs store all task details, including services chosen for task completion, and input parameters previously provided by the mobile user. A CPA uses the stored information to run automatic tasks. The mobile user can also signal his/her CPA to run a previous task at any time, without having to perform service discovery/selection, and parameter input again.

Task results are based on HTML to give service developers the flexibility to customise the format of their service results to their choosing. Services specify a template HTML result page as part of their description (described shortly), which can use company specific CSS and JavaScript files. When a CPA has finished running a task, it fetches a copy of the result template and writes the JSON result data into it, converting to HTML markup in the process. This page is saved on the CAMCS cloud storage, or the user can choose to store it in his/her own cloud storage provider account (e.g. Dropbox). Task result pages are viewed using the Android WebView Activity on the CAMCS Client.

B. Context-Awareness with the Context Processor

As described in our requirements outlined for an integrated user experience of MCC, CAMCS provides contextual-awareness support by means of its Context Processor component, described in a previous work [6].

The Context Processor is included with CAMCS to provide personalisation of task execution with cloud services, and task results. When a mobile user chooses a service for completing a task that can benefit from user context data as an input, a CPA can pull context-data for their owner from the Context Processor, and pass this information to the cloud service, with user consent. The Context Processor stores user context data as Ontologies in XML. Each user has current and historical context data stored. Historical context data can be used to imply current context if the mobile user is disconnected and unable to provide fresh context data.

The CAMCS Client collects context data from the sensors of the mobile device. With the user's consent (by switching the feature on in the client Settings), a background service runs and collects context data, such as user location and activity, at set intervals provided by the user. These updates are sent to CAMCS, which stores the collected data for the user on behalf of his/her CPA with the Context Processor.

CAMCS also supports a feature known as Context Profiles. A user can set different profiles to be active at his/her CPA at any given time. These profiles can influence the tasks that a CPA carries out automatically. For example, a work profile can allow some automatic tasks, such as fetch the traffic on the user's route to work, to run during the week, but not during the weekend.

C. Service Discovery and Consumption

CAMCS differs from other common MCC models, in that it relies on existing services deployed on cloud infrastructures. These services are deployed using common SOA web-service technologies and styles such as the Simple Object Access Protocol (SOAP) and RESTful architecture, and are capable of delivering useful information and functionality to users. Such services are not suitable for direct human-consumption, as they are commonly described with XML-based technologies such as WSDL. CAMCS uses a custom user-oriented mobile cloud service description format for describing existing web-based services [5]. These feature user-friendly service descriptions, which allow the mobile user to take part in the service discovery process, to find a service that his/her CPA can use to complete a task. This format also allows users to provide input parameters to services, with user-friendly names and descriptions of parameters shown through the CAMCS Client. These can be ordinary data parameters, or contextual parameters.

III. Design

The final architectural design of CAMCS is shown in Fig. 1. The original components proposed in the reference architecture, and new supporting components, have been developed for CAMCS into three layers; the *Management Layer*, the *CPA Layer*, and the *Execution Layer*. A flow diagram showing how the execution of a new task uses the components in the three layers is shown in Fig. 2. Also shown is the interaction required between the mobile user through their mobile device with the CAMCS Client, and CAMCS.

A. Management Layer

The Management Layer contains common components shared by all CPAs to support their operation. Some of these

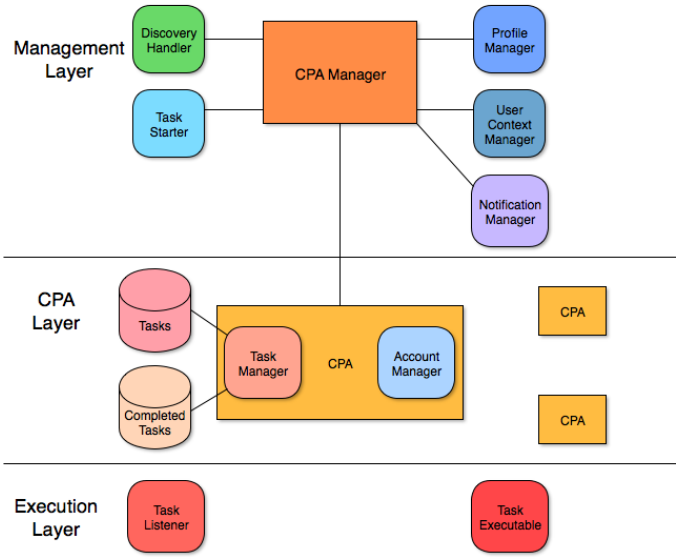


FIG. 1. A detailed view of the components of CAMCS, which can be divided into three layers. The Management Layer contains many components shared by all CPAs. The CPA Layer contains the CPAs, along with components that each CPA has a unique copy of, and finally, the Execution Layer contains components used during task execution.

CPA components are interfaces to allow CPAs to communicate with a specific component of CAMCS, such as the User Context Manager, which directs CPAs to the Context Processor. The other CPA components wrap their own functionality, such as the Notification Manager. A CPA uses the components in the Management Layer via the CPA Manager. The CPA Manager is also used for authenticating user access to CPAs when requests are received from the mobile device, using password-based HTTP authentication. Some of the main components are now briefly described:

- Discovery Handler

The Discovery Handler encapsulates all functionality related to Service Discovery. A CPA passes the user-provided task description to the handler, which contacts and queries the service registry over HTTP, with the task 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, a CPA passes this information to the Task Starter. The Task Starter creates a Task Executable for executing the task, and passes a map containing the user provided parameters.

B. *CPA Layer*

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

- Task Manager

The Task Manager stores a CPA's Current Tasks and Completed Tasks, making up the 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 his/her CPA to third-party cloud services, which are used for features such as context awareness. The Account Manager also contains the operations to contact those services, and push/pull information to/from them. This was presented in a previous work [7].

C. *Execution Layer*

The Execution Layer 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. The Task Executable, using all task information provided by the user, is the component that makes the remote call to the cloud-services over HTTP, passing the parameter information. It also prepares the result data retrieved back from the service for storage within CAMCS on behalf of a CPA.

- Task Listener

The Task Listener listens for task events such as task completion, or a task error. It notifies the CPA of these events (so that 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.

IV. Evaluation

CAMCS was deployed on the free tier of Amazon EC2 for evaluation. The CAMCS Client, running on Android-based mobile devices, is also evaluated in terms of resource usage. The free tier instances used in these evaluations are of type t2.micro. These are categorised as general-purpose instances, featuring an Intel Xeon CPU burstable to 3GHz, 1GB memory, "low" network performance, and no elastic block storage (EBS) optimisation.

A. *CAMCS Middleware Evaluation*

A CAMCS Docker image was used to run CAMCS within Docker Containers on the EC2 instances. This initial setup was used to create a custom Amazon Machine Image (AMI), which was used as a template for launching new EC2 instances where auto-scaling was used during these experiments.

For load testing CAMCS, Apache JMeter was used to simulate mobile users sending task descriptions to his/her CPA, to find a service to play a game of N-Queens, for a random value of N. A description of the N-Queens service running on another EC2 instance was manually inserted into the CAMCS registry before experiments began. Therefore, the sequence of events for the experiments are 1) CAMCS receiving the task from a user (a JMeter thread here), 2) perform service discovery to find the service, 3) contact the service to play the N-Queens game for the random N, 4)

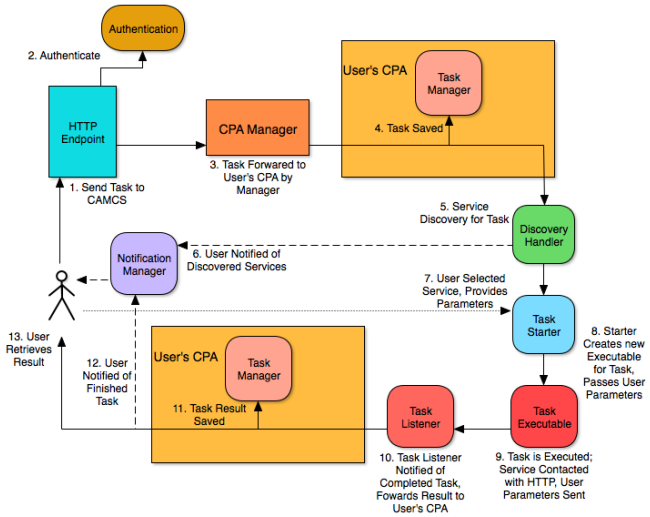


FIG. 2. A sequence diagram showing how the components of CAMCS are used by CPAs to complete a task. The solid lines show the main flow of steps through the system. The dashed lines show steps where the user is notified of an event using the Notification Manager. The dotted line shows where the user provides more information before the main flow can proceed.

obtain the result (time taken to solve for random N) and return it to the JMeter "user".

It is important to note for these experiments, that they do not represent the expected way a mobile user would interact with CAMCS and his/her CPA. CAMCS provides an asynchronous, disconnected approach to task completion, making it difficult to gauge performance. For these experiments, CAMCS was effectively made synchronous. Once the task was received by CAMCS, no call-backs are made to mobile devices for users to choose a service, provide input parameters, and view results. Service discovery, execution, and the returning of task results, all occurred in a sequential, blocking fashion for these experiments. For this purpose, the concept of *Anonymous CPAs* was introduced; these are CPAs that are not tied to individual users; rather, they sit waiting to complete tasks received from any user.

JMeter was started with 1000 users and a ramp-up period of 60 seconds. The horizontal auto-scaling policies were set as follows: launch 1 new instance each time the average CPU utilisation of an instance was greater than or equal to 85% for a continuous period of 5 minutes, and terminate 1 EC2 instance when the average CPU utilisation dropped to 50% or lower for a continuous period of 5 minutes. The actual service discovery and consumption calls over HTTP were disabled, so that only the task processing performance was measured.

Fig. 3 shows the CPU utilization for the first instance used in the experiment; the CPU utilization jumps to 100% immediately. After 5 minutes, the scale-up alarm is fired, and at the 10-minute point, the second instance has become flagged as InService. After this, both instances achieved utilisation of approximately 75% for the duration of the test when they were both InService.

JMeter results showed that with 1000 users, before the second instance became InService, with 34650 sample requests at that point, average response time was 16.3 seconds, with throughput at 55.4 requests per second. The response time is twice that of the response time for 500 users, although with comparable throughput (500-user experiment omitted due to space constraints). 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.

The metrics also show a decrease in the median response time from 17.1 seconds to 2.7 seconds, while the 95% confidence interval increased from 18.8 seconds to 30 seconds, which is explained by the response time graph from JMeter (omitted due to space constraints). With one instance InService, the average response time for requests is between 17 and 18 seconds. Once instance two comes into service at the 10-minute point, the response times are shown to become far more erratic, between 2 and 35 seconds. This appears to be the result of the AWS load balancer. JMeter showed a request error rate of 2% at the end of the test. These errors 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 (not shown), these errors and erratic response times do not appear, and the error rate remains at 0%.

Recall that no user will be blocked and waiting on his/her mobile device for between 11 and 16 seconds for a task to complete. The model is asynchronous and non-blocking; the user is notified when each step of the task execution process completes. Task results load in seconds similar to any webpage.

The evaluation shows that CAMCS can scale and handle a high volume of growing users, although longer response times show the value of using the appropriate number of instances to handle the load, and the right instance type for the compute intensive operations. This is not the case here with the t2.micro type, which is not categorized by AWS for compute-intensive applications.

B. CAMCS Client Evaluation

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. All of the operations on the CAMCS Client are maintained asynchronous for the evaluation, as this is how the mobile user would interact with CAMCS.

To evaluate an example of using the CAMCS Client to perform task work, a new task was created for fetching events data from a calendar, using our sample Google Calendar service. The experiment was performed twice; once on a Wi-Fi connection, and once on a 3G cellular connection. The results were almost identical, and so only the cellular results are shown here, as cellular performance is typically the bottleneck with other MCC solutions. Metrics were recorded with the Trepn Profiler for Android [8]. A base-line power consumption of 1.5W was recorded for the duration of the

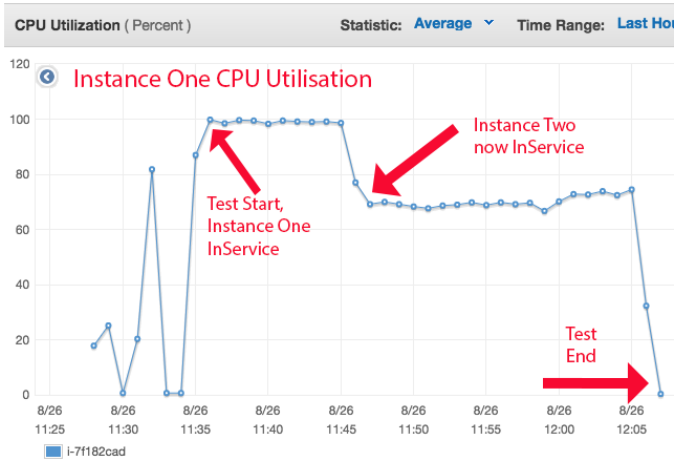


FIG. 3 AWS CloudWatch CPU Utilisation Graph for EC2 Instance One, with 1000 users and horizontal auto-scaling, captured at the end of the experiment, showing the initial average CPU utilisation of 100% followed by a decrease to approximately 70% after the second instance became InService.

work. Spikes were present each time network activity occurred, for the most part up towards 3W, with others spiking to 5.5W. A total of 5KB of data was sent to the CPA, while the client received 377.91KB from the CPA. By far, most of the data received is from the result HTML page being fetched and displayed. The profiler shows that the size of the result HTML file for the test individual's calendar was 372.37KB.

The Trepn Profiler application running on the Android device also recorded measurements during the test. Peak memory usage for the CAMCS Client during the interaction was 24.7MB, and CPU utilisation averaged at 2.21%. 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.

V. User Experience Analysis

The requirements for an integrated user experience that were presented in our previous work [3] will now be recalled. Based on the evaluation results presented, we determine if CAMCS meets these requirements.

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

By only transferring small amounts of data, and disconnecting between each step when completing a task, the effects of latency are not noticeable to the end user.

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, 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. The CAMCS Client evaluation section showed little resource usage during the required steps.

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

CAMCS with its disconnected and asynchronous task execution model has been guided by the principle of disconnected operation from the beginning. As the mobile device is not responsible for any aspect of the work involved with completing a task aside from user input, CAMCS meets this mobility requirement.

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

User context data can be used for both service discovery and task execution. Context can be retrieved from the mobile device and other sources, such as social networks. Support is included in the user-oriented service description format for context parameters, allowing services to describe what context data inputs they can receive. In the absence of fresh context, new context can be inferred from historical context data.

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

CAMCS addresses mobility concerns such as disconnection, unlike other solutions where desktop VMs are used. CAMCS takes advantage of unique mobile features. 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, can be displayed on all mobile devices with web browsers.

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

The CAMCS Client features a task sending decision maker to respect the current resource status of the mobile device [9]. The decision maker considers network signal strength, the size of the data to be sent, the battery status, and task priority. The algorithm is zero-overhead, performing no network profiling aside from reading GSM signal strength already available. Tasks descriptions will be sent to CAMCS either immediately, or queued until the network quality improves.

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

CAMCS works with existing service technologies, such as RESTful services. WSDL files that are used to describe SOAP-based services can easily be translated into the user-oriented mobile cloud service description format used by this work, as all the elements of the service description format have an equivalent markup in WSDL.

VI. Related Work

Many MCC solutions do not typically respect the user experience requirement that CAMCS was designed with from the outset. The Cloudlet architecture [1] uses low-resource compute infrastructure placed around Wi-Fi access points where many mobile users gather, such as at a coffee shop. Cloudlet approaches are based on virtual machines, and may be demanding on the device battery because of the continuous connectivity requirement, and frequent data transfer. Code-offloading and application partitioning approaches [2] [10] will offload methods of mobile application code-bases, or

components of a partitioned application, to the cloud infrastructure for execution there, with results returned to the mobile device. These approaches will also not function without a continuous connection to the cloud. Performance results on cellular networks are shown to be extremely poor, or not evaluated. This is crucial for the mobility aspect of the user experience. CAMCS evaluations showed identical performance on both Wi-Fi and cellular networks, because the disconnected approach and low data-transfer requirements.

Middleware approaches with SOA include a mobile cloud middleware [11] which performs result optimisations between XML and JSON-based web services. The solution required the mobile user to know the WSDL URL of the service, and the type of service (SOAP/RESTful). CAMCS hides such complexity. Another middleware [12] provides an adapter between a specific set of different cloud-based services, to format requests to those services correctly. CAMCS is designed to work with all cloud-based services. The Avatar middleware [13] uses a copy of the mobile operating system in the cloud as a virtual machine, known as an Avatar. Mobile applications can run either entirely on the Avatar, or components can be split between the Avatar and the mobile device.

Consumer products that deliver cloud-based services include Google Now [14], and Apple Siri [15]. Google Now can provide useful information such as current weather and traffic with a specific set of services, similar to CAMCS. However, Google Now carries out its work on the mobile device, rather than in the cloud as with CAMCS. Google Now only works with some service providers; with CAMCS, any developer can offer a service. Apple Siri is voice-based, and requires an Internet connection for voice recognition to extract user queries; it will not operate without an active connection.

VII. Conclusions

In this paper, the Context Aware Mobile Cloud Services (CAMCS) middleware was presented as a solution that delivers an integrated user experience of the mobile cloud, by respecting the limited resources available on mobile devices. This is achieved by means of the Cloud Personal Assistant (CPA). CPAs are representatives of mobile users within CAMCS that use cloud-based services to complete tasks assigned to them in a disconnected, asynchronous fashion. Tasks are received by CPAs as descriptions sent from the CAMCS Client running on the mobile device.

The following contributions were made: The features of CAMCS were described: task models, contextual-awareness support, and the user-oriented service discovery and consumption processes. The final design and implementation of CAMCS was also presented. An evaluation was presented using load testing while deployed on Amazon EC2, showing the performance benefits of scaling. The CAMCS Client was also analysed, showing the low resource usage in terms of power consumption and network activity. Finally, analysis of the results showed that CAMCS met the requirements outlined in our previous work for an integrated user experience.

In future work, CAMCS can be extended to support additional service models. Also of interest are groups of CPAs collaborating to complete large tasks together, and task results feeding into other tasks as input data.

Acknowledgment

The PhD Research of Michael J. O'Sullivan was funded by the EMBARK Initiative of the Irish Research Council.

References

[1] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, "The case for VM-based cloudlets in mobile computing", IEEE Pervasive Computing, 2009; 8(4), pp. 14-23.

[2] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, et al, "MAUI: making smartphones last longer with code offload", in Proceedings of the 8th international conference on Mobile systems, applications, and services, San Francisco, California, USA. 1814441, ACM, 2010, pp. 49-62.

[3] M. J. O'Sullivan, D. Grigoras, "User experience of mobile cloud applications – current state and future directions", in Proceedings of the 12th International Symposium on Parallel and Distributed Computing, Bucharest, Romania, 27-30 June, 2013, pp. 85-92.

[4] M. J. O'Sullivan, D. Grigoras, "The cloud personal assistant for providing services to mobile clients", in Proceedings of IEEE 7th International Symposium on Service Oriented System Engineering (SOSE), Redwood City, San Francisco Bay, California, USA, 2013, pp. 477-484.

[5] M. J. O'Sullivan, D. Grigoras, "Delivering mobile cloud services to the user: description, discovery, and consumption", in Proceedings of IEEE 4th International Conference on Mobile Services (MS), New York City, New York, USA, 2015, pp. 49-56.

[6] M. J. O'Sullivan, D. Grigoras, "Mobile cloud contextual awareness with the cloud personal assistant, in Proceedings of the 2nd International Conference on Future Internet of Things and Cloud (FiCloud-2014), Barcelona, Spain, 27-29th August, 2014, pp. 82-89.

[7] M. J. O'Sullivan, D. Grigoras, "Mobile cloud application models facilitated by the cpa", EAI Endorsed Transactions on Scalable Information Systems, 15(4), 2015.

[8] Trepn Profiler for Android. https://developer.qualcomm.com/software/trepn-power-profiler Last Accessed 25/11/15.

[9] M. J. O'Sullivan, D. Grigoras, "Integrating mobile and cloud resources management using the cloud personal assistant, Simulation Modelling Practice and Theory, Vol. 50, pp. 20-41, January 2015, ISSN 1569-

[10] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: Enabling mobile phones as interfaces to cloud applications", Middleware 2009, pp. 83-102.

[11] Q. Wang, R. Deters, "SOA's Last mile-connecting smartphones to the service cloud", in Proceedings of the IEEE International Conference on Cloud Computing (CLOUD), Bangalore, India, 21-25 September, 2009, pp. 80-87.

[12] H. Flores, S. N. Srirama and C. Paniagua, "A generic middleware framework for handling process intensive hybrid cloud services from mobiles", Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia, Ho Chi Minh City, Vietnam, 2095715, ACM, 2011, pp. 87-94.

[13] C. Borcea, X. Ding, N. Gehani, R. Curtmola, MA Khan, and H. Debnath, "Avatar: mobile distributed computing in the cloud", in Proceedings of 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud); March 30-April 3, 2015, pp. 151-156.

[14] Google Now. https://www.google.com/landing/now/ Last Accessed 25/11/15.

[15] Apple Siri. https://support.apple.com/en-ie/HT204389 Last Accessed 25/11/15.