Mobile Cloud Contextual Awareness with the Cloud Personal Assistant

Michael J. O'Sullivan, Dan Grigoras
Department of Computer Science
University College Cork, Cork, Ireland
{m.osullivan, grigoras}@cs.ucc.ie

***Abstract* – This paper presents our efforts to bridge the gap between mobile context awareness, and mobile cloud services, using the Cloud Personal Assistant (CPA). The CPA is a part of the Context Aware Mobile Cloud Services (CAMCS) middleware, which we continue to develop. Specifically, we discuss the development and evaluation of the Context Processor component of this middleware. This component collects context data from the mobile devices of users, which is then provided to the CPA of each user, for use with mobile cloud services. We discuss the architecture and implementation of the Context Processor, followed by the evaluation. We introduce context profiles for the CPA, which influence its operation by using different context types. As part of the evaluation, we present two experimental context-aware mobile cloud services to illustrate how the CPA works with user context, and related context profiles, to complete tasks for the user.**

Keywords: mobile cloud, applications, services, user experience, context awareness

I. INTRODUCTION

With the introduction of the mobile cloud paradigm, new application models and services can be offered to mobile devices from the cloud. These may require computation resources that are scarcely available on mobile devices. Specifically, our research interest lies in providing an integrated user experience of mobile cloud applications to the user. To achieve this, in our previous work [1], we introduced Context Aware Mobile Cloud Services (CAMCS). This is a cloud-based middleware that can tackle mobility factors that result in a detrimental impact on the user experience, such as network disconnection, variable bandwidth, and low energy supply from the battery.

In our previous work, we outlined several requirements we believe to be essential to providing this user experience. One of these calls for the need for contextual awareness, to play a part. This will 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.

To avail of the services provided by CAMCS, each user has to register an account using their email address, which can be done using a thin client running on the mobile device. Each user is then assigned his/her own Cloud Personal Assistant (CPA) [2]. The core motivation behind the CPA is that a user offloads demanding tasks they wish to complete to their CPA in the cloud. The CPA then discovers the appropriate cloud services to fulfil those tasks. Once a task is complete, the CPA will store the result until the mobile device is ready to receive it. This implies that the mobile device can become disconnected from the cloud after the task has been offloaded; a continuous connection is not required. The user provides task information using the mobile thin client application. In this work, the thin 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. Or the context data can be provided as input to the services if it can be of use in executing the offloaded task and getting the result. This is aided with the introduction of Context Profiles in this work. 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 is currently disconnected, new context can be inferred from historical context. 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. 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.

In our previous CAMCS work, we described the structure of the system. One of the components of this system, is the Context Processor, which gathers the contextual data from the

978-1-4799-4357-9/14 $31.00 © 2014 IEEE
DOI 10.1109/FiCloud.2014.23

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 paper describes our current implementation of the Context Processor component of the CAMCS middleware. In addition, we present an example of two experimental mobile cloud services that the CPA can use, along with the gathered context. 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, and a car context profile, which makes use of the Twitter API.

Many different approaches have been taken to collect context information from the mobile device, along with use of this context with mobile applications in related work. Projects usually consist of middleware approaches that collect the context, perform processing, feature-extraction, and inference, before finally providing the context to some example applications. Some works do utilise the cloud for these collective aspects of context collection, other works solely use a mobile middleware approach. Most works focus on the middleware collection and processing itself, rather than how it can be provided and consumed in a generic manor. The contributions of this paper are that we focus not only on how we handle the context data in our CAMCS middleware, 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.

The remainder of this paper is organised as follows. Section 2 describes the design and architecture of the Context Processor within the CAMCS middleware. Section 3 describes our current implementation and features. Section 4 details our evaluation of the Context Processor with the experimental mobile cloud services. Section 5 contains the related work. Finally, we conclude and describe future work in Section 6.

II. 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. We considered two context representations for the middleware, Ontology based with the OWL API [3], and an Object Oriented approach using the Java Context Awareness Framework (JCAF) [4]. We ultimately chose the OWL API approach; this can persistently store context as Ontologies in XML files, and provides an Inference Engine for context reasoning. These aspects are crucial for the disconnected nature of the middleware. JCAF does not provide an inference engine, or a form of persistent storage. Fig. 1 shows the Context Processor components.

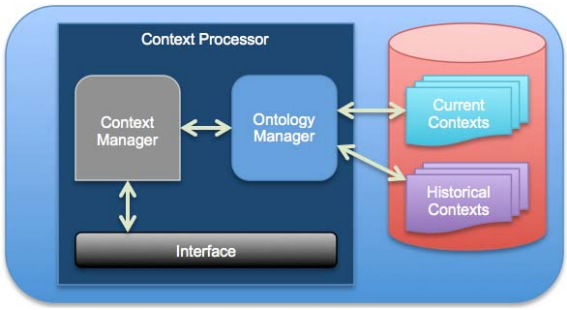

Fig. 1 The Context Processor within the CAMCS middleware. It features an interface for the CPA of a user to request and store context. The Context Manager handles the different context operations. The Ontology Manager reads and writes context to the Current and Historical context Ontology files on the cloud storage.

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 future approach we are considering is to store these files on the user's personal account with a cloud storage provider, such as Dropbox. A database approach would make 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. We are currently experimenting with how long context history should be stored for. 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. 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, in the case that further down the line, another approach is taken to model context, the CPAs will not need to change the way they request and use context. This approach also assisted in the evaluation of the

OWL API against JCAF, as we were able to switch between both approaches quickly, although ultimately JCAF quickly fell short of the requirements.

One of our goals is to provide a public API for the middleware; this will allow developers to issue requests for tasks to be completed in the cloud by the CPA of a user. This API will also allow developers to create cloud services usable by CPAs. The Context Processor is designed to be extendable, so that developers can create their own contexts if required. This also assists us in quickly adding new contexts into the middleware for new services and experimentation.

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.

A. *Sending User Context to the Context Processor with the Context Wrapper*

The thin 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 thin 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 our architecture, we have a simple Context class. Each context extends this class. 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 ContextUpdate 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 Fig. 2. With this design, whenever a new context is required, the developers need only provide their own extensions to the base classes.

B. *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

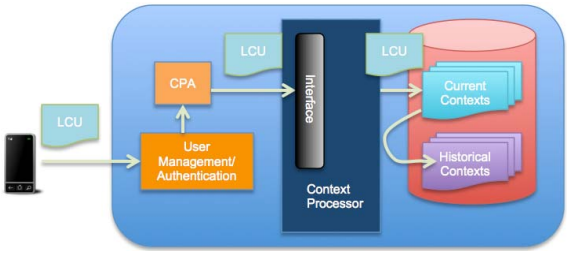

Fig 2. The Context Processor receives a Location Context Update (LCU), from the user's mobile device, via the CPA of that user. The context update is written to the current context Ontology of the user. The old location context is moved to the historical context.

context, it simply sends a ContextRequest to the Context Processor interface. There are several subclasses of ContextRequest, for the different types of Context. 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 the Context class. See Fig. 3 in Section 4 for a graphical example, related to the use-case example given in that section.

C. *Discussion*

Pending further discussion in Section 4, it is clear that we have taken an object-oriented approach, similar to other works. One can easily see why originally, using JCAF would have been our 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 our Context Processor as extendable Java objects, the representation, storage, and inference takes place in the Ontology XML files. The object-oriented abstraction in our code makes it easier to handle at the programming level. One may argue that asking developers to extend these classes and create their own contexts is putting much burden on their time. Our opinion is that context can only be exploited when meaningfully described as a compound context; that is a context made up of several smaller, simpler contexts, already provided by our middleware (e.g. location, time). This way, contexts can be personally adapted accurately to the situation. Our aim is that the Context Processor will hide away all the Ontology and XML details of the contexts, so the developer need only work with high-level objects. Once the required classes have been created which describe the properties of the context, the developer need only work with the Context Processor through the interface.

III. IMPLEMENTATION

Following, the implementation of the system is discussed. The CAMCS middleware is being developed as a Java project, which can be deployed on cloud servers, such as Amazon EC2. Our cloud server for experimentation is located within our University; the server has a 1.7Ghz CPU, and 2GB RAM.

A. Contexts Utilised

Currently, we are trying 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 context to infer new context without having to request it from the mobile device. For example, if the mobile device sends the latitude and longitude of the mobile device location, geocoding can take place at the CAMCS middleware, rather than at the mobile device. If we know the users location, the middleware 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.

Our mobile device platform for development is the Android OS. We currently use the contexts provided by Google Play Services in this work, namely, location, and user activity recognition. Using Google Play Services is a choice that considers our project user experience goal. 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).

B. Ontology Implementation

The basis of the context representation as Ontologies in our middleware is based on the SPICE mobile ontology [5]. 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.

We have created our own Ontology, which imports and extends the SPICE Ontology, specifically its existing classes and properties. We use a NamedIndividual to represent each user. A NamedIndividual can be thought of an instantiation of a class. Specifically, we have an axiom specifying 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 our Ontology, we created our own Place class, which subclasses Location from the SPICE ontology, to represent a user location. The benefit of taking this subclass approach is that we can add our own required properties and data to the Ontology, while using the existing Ontology entities and axioms. For example, our Place class contains a latitude/longitude, and timestamp properties.

C. Mobile Thin Client

The thin client running on the mobile device, as previously described, is used to communicate with the CPA of the user. In this work, 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 thin 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 the thin client services with the context updates, which are then placed into the Context Wrapper. Our mobile device uses the Spring Android framework [6], to send a HTTP(S) PUT request to the CAMCS middleware. Authentication takes place, and the context data is given to the CPA of the user.

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 thin client should receive the context update events.

D. Context Profiles

Our implementation is based on the use of context profiles. The profiles correspond to the daily activities or status of the user. Presently, we have a profile for home, work, and for our evaluation with cloud services in the next section, a tourist and car profile. The home, work, and car 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 car profiles, and fetch traffic information for the best route to work. During the weekend, this mode will not activate.

The user manually activates the tourist profile with the thin 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 large advances to the middleware, 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.

E. 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.

Our thin client and the CAMCS middleware 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 thin client then fetches the result of that task from the CPA. The CPA currently stores the task result as a HTML web page. This page is generated by the CPA when a task is complete. The HTML is then written into our MongoDB NoSQL database, which we use for storing user, CPA, and task data. This is a somewhat bloated approach. We are considering storing the HTML result file in personal cloud storage owned by the user, such as Dropbox.

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.

Ideally, we had considered implementing our own 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. We cannot implement this with GCM, as it has a message payload limit of 4KB. Our 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 our positive user experience aim. Despite the need for the user to explicitly request to fetch the result of a task from the CPA, we adhered to Google's recommendation to conserve battery life.

IV. EVALUATION

We now turn our attention 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, we have implemented two experimental mobile cloud services 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. We also discuss the benefits and limitations of this work.

A. Tourism Mobile Cloud Service

1) Location Services with Foursquare

Foursquare [7] 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 system, we hooked into this API 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 geocoded name of a place, given by the coordinates.

2) Tourism Service Implementation

To implement this tourism service, we developed a cloud-based service that can be used by a CPA. For the purposes of this work, this service project is simply a dependency of the CAMCS middleware. As a result, the CPA is aware of the existence of this service already, and does not need to discover it.

Whenever the tourist context profile is active at the CPA, the CPA assumes that the user is on some form of 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 (for example, 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 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 Fig. 3, which shows the series of steps involved. Fig. 4 presents a screenshot from the mobile device, displaying the location information result sent back by the CPA.

B. Traffic Mobile Cloud Service

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 work, we tried a different approach. Several agencies, such as the AA in the UK and Ireland, provide traffic information through their Twitter [8] 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.

2) Traffic Services Implementation

To implement this service, a "car" 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 activities provided by Google Play Services is "in_vehicle". When the CPA receives this context update from the thin 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 discussed in a previous work [9], the CPA integrates with external service providers such as Facebook and Dropbox. Twitter is one of these providers. The thin 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 project, the CPA will 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 in our MongoDB NoSQL database.

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 thin client will fetch the traffic information result from the CPA, without the user having to tap and open the notification. When the HTML traffic result is returned, the tags are stripped out of the result, and the Android Text-To-Speech engine reads out the Tweets - see Fig. 5. Note that use of Android Text-To-Speech requires a speech engine on the device. The Samsung Galaxy S3 device used in the project has such an engine.

Of course, 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. Tweets also tended to contain other useful transport information not related to drivers, such as information about trains, subways, and buses.

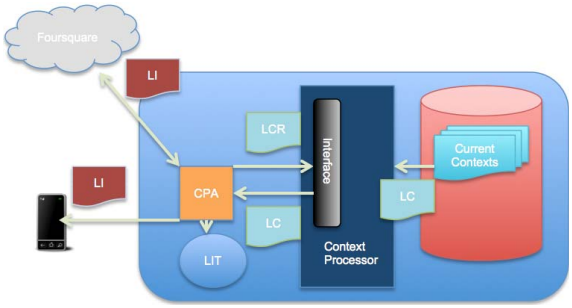

Fig. 3 With the Tourist Profile active, the CPA is running a Location Information Task (LIT). It sends a Location Context Request (LCR) to the Context Processor, which reads the users Location Context (LC) from their Current Context Ontology. The CPA contacts Foursquare with this information, which responds with Location Information (LI). This is returned to the device.

C. Discussion

The complexity and potential is in the ability to create services that the CPA can use with the gathered context. These services can exist in the mobile cloud, and can work with existing cloud and web-based services to complete work, or provide information, to the user, through their CPA. This can happen as the 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. Developers can build their own Context Profiles to define events or tasks that should occur automatically, using gathered contextual data.

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. We therefore need to build wrappers around these existing services, for the purposes of contacting and utilising them.

Fig. 4 Screenshot of Foursquare location results from the CPA, displayed on the Android thin client.

Fig. 5 Screenshot of traffic information provided by Tweets for Cork City, Ireland, displayed on the thin client.

This is exactly what we are doing with our example services. They serve as wrappers for the developer to specify how the CPA should work with an existing cloud-based service. Currently, this must be done for any existing services. 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.

We feel the time commitment for developers is worthwhile, as this will deliver a high level of personalisation, benefitting the user experience aim of the middleware.

V. RELATED WORK

Context awareness is widely studied in relation to context consumption and dissemination on mobile devices. We are unaware of any related works that utilise context data for providing services to the mobile user through a cloud based user representative, such as the CPA.

Hofer et al [10] developed the Hydrogen approach for context-awareness on mobile devices; this is a three-tier framework for capturing, storing, and providing contexts to consumers, from a mobile device. It does provide the ability to share context with neighbouring devices in an ad-hoc fashion. Lowe et al [11] developed the Context Directory, which is a directory that stores context as key-value pairs. This does use a server for processing and representing the directory. Raento et al [12] developed ContextPhone, a context-awareness framework that allows development of context-aware applications. For example, it can share the context of a user with others who have that user in their contacts.

In terms of context representation and modelling, we use the SPICE mobile ontology in this paper by Villalonga et al [5]. This is a standardised Ontology, which can be used to represent elements of mobile context. We use the OWL API by Horridge and Bechhofer [3], which is a Java API for working with OWL Ontologies. However, this is not the only Ontology-based approach to context representation. Korpipää and Mäntyjärvi [13] have developed a mobile Ontology for representing context information gathered from the mobile device sensors. Naturally, Ontologies are not the only way to represent context data. The survey work by Strang & Linnhoff-Popien [14] looked at the advantages and disadvantages of several different approaches to context representation. This paper mentions the other approach we evaluated for this work, an Object-Oriented approach, the Java Context Awareness Framework (JCAF) by Bardram [4]. Other approaches include a model driven approach, such as the approach taken by Sheng and Benatallah [15], which uses UML-based modelling to represent context.

In regards to using context data with web services, Truong and Dustdar [16] survey existing approaches to how context can be used with web services. Several context frameworks were evaluated in terms of their compatibility with web services under different criteria. An example of one of these approaches is a middleware system by Gu et al [17]. This middleware provides context information to SOA architectures. In regards to mobile cloud and context awareness, we are aware of a work by Han et al [18], which 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. In addition, a work by La and Kim [19] 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 idea of using a context history for inferring missing context comes from the work by Hong et al [20]. New context can be inferred from context history, so that if new context is missing, or if we want to save energy on the mobile device by not collecting new context, the context engine will not encounter any problems. In the work by Beach et al [21], they 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 existing mobile cloud applications and middleware systems which make use of context. The tourism service in this paper is inspired by a work from Abowd et al [22], called Cyberguide, which uses the location and location history of a user to replicate functionality and services offered by a real tour guide. A mobile cloud middleware by Wang and Deters [23] utilises context for providing web service mash-ups to users, along with experimental applications they developed for testing.

VI. CONCLUSIONS

In this paper, we have introduced our development efforts in building the Context Processor component of our Context Aware Mobile Cloud Services (CAMCS) middleware. 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. This occurs through the Cloud Personal Assistant (CPA), the main representation of each user in the middleware. When the 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.

We took an Ontology based approach 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 we cannot gather new context. We presented the architecture of the Context Processor, along with our experimental implementation. Mobile context is collected by our Android thin client, based on Google Play Services, and is sent to the CPA to be stored by the Context Processor. We provided examples of its use with two mobile cloud services; a tourism service, which works with Foursquare to provide tourism context services based on location, when the tourist context profile is active at the CPA, and a traffic information service, which uses a car context profile, activated by the activity context, which pulls tweets from Twitter containing traffic information for the user location. This is read out to the user at the thin client using the Android Text-To-Speech engine. We evaluated the advantages and disadvantages of our approaches.

In our future work, we will be developing additional services to be used for evaluation as part of the overall development goal of the middleware. Future work on service discovery with the CAMCS middleware will assist this goal. We will also be looking at adding other sources of context, such as from social networks, as proposed in the related work. Ultimately, as with the entire CAMCS system, we hope to publish it as an API to help other developers so that they can introduce their own context-based services.

ACKNOWLEDGMENT

The PhD research of Michael J. O'Sullivan is funded by the Embark Initiative of the Irish Research Council. We would like to extend our thanks to the reviewers of this paper for their helpful comments and suggestions for improvement.

REFERENCES

[1] M. J. O'Sullivan, D. Grigoras. User Experience of Mobile Cloud Applications – Current State and Future Directions, Proceedings of the 12th International Symposium on Parallel and Distributed Computing, Bucharest, Romania, 27-30 June, 2013, pp. 85-92.

[2] M. J. O'Sullivan, D. Grigoras. The Cloud Personal Assistant for Providing Services to Mobile Clients, IEEE MobileCloud, Redwood City, San Francisco Bay, USA, 2013, pp. 477-484.

[3] M. Horridge, S. Bechhofer. The OWL API: A Java API for OWL Ontologies, Semantic Web Journal, Special Issue on Semantic Web Tools and Systems, Vol. 2, No. 1, 2011, pp. 11-21.

[4] J. E. Bardram. The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications. Proceedings of the 3rd International Conference on Pervasive Computing (Pervasive 2005), Lecture Notes in Computer Science, Springer Verlag, Munich, Germany, May 2005.

[5] C. Villalonga, M. Strohbach, N. Snoeck, M. Sutterer, M. Belaunde, E. Kovacs et al. Mobile Ontology: Towards a Standardized Semantic Model for the Mobile Domain, *Service-Oriented Computing-ICSOC 2007 Workshops*, Springer Berlin Heidelberg, pp. 248-257.

[6] Spring Android Framework. http:// http://projects.spring.io/spring-android

[7] Foursquare API. https://developer.foursquare.com/

[8] Twitter Developers. https://dev.twitter.com/

[9] M. J. O'Sullivan, D. Grigoras. Application Models Facilitated by the CPA. Proceedings of the 6th International Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications (MOBILWARE), Bologna, Italy, 11-12 November, 2013

[10] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, W. Retschitzegger. Context-awareness on mobile devices - the hydrogen approach, Proceedings of the 36th Annual Hawaii International Conference on System Sciences, Hawaii, USA, 6-9 January, 2003.

[11] R. Lowe, P. Mandl, M. Weber. Context Directory: A context-aware service for mobile context-aware computing applications by the example of Google Android, Proceedings of the IEEE Pervasive Computing and Communications Workshops (PERCOM Workshops), Lugano, Switzerland, 19-23 March, 2012, pp. 76-81.

[12] M. Raento, A. Oulasvirta, R. Petit, H. Toivonen. ContextPhone: a prototyping platform for context-aware mobile applications, IEEE Pervasive Computing, 2005, Vol. 4, No. 2, pp. 51-59.

[13] P. Korpipää, J. Mäntyjärvi. An Ontology for Mobile Device Sensor-Based Context Awareness, Modeling and Using Context: Proceedings of 4th Int'l and Interdisciplinary Conf. (Context 2003), LNCS 2680, Springer-Verlag, 2003, pp. 451–458.

[14] T. Strang, C. Linnhoff-Popien. A Context Modeling Survey, Proceedings of First International Workshop on Advanced Context Modelling, Reasoning And Management at UbiComp 2004, Nottingham, England, September 7, 2004, 2004-09-07.

[15] Q. Z. Sheng, B. Benatallah. ContextUML: a UML-based modeling language for model-driven development of context-aware Web services, International Conference on Mobile Business (ICMB), 11-13 July, 2005, pp. 206-212.

[16] H.-L. Truong, S. Dustdar. A survey on context-aware web service systems, International Journal of Web Information Systems, vol. 5 no. 1, 2005, pp.5 – 31.

[17] T. Gu, H. K. Pung, D. Q. Zhang. A service-oriented middleware for building context-aware services, Journal of Network and Computer Applications, Vol. 28, No. 1, 2005, pp. 1-18.

[18] B. Han, W. Jia, J. Shen, M-C. Yuen. Context-Awareness in Mobile Web Services, Parallel and Distributed Processing and Applications, Springer-Verlag, Heidelberg, 2005, pp. 519-28.

[19] H. J. La, S. D. Kim. A Conceptual Framework for Provisioning Context-aware Mobile Cloud Services, IEEE 3rd International Conference on Cloud Computing (CLOUD), Miami, Florida, USA, 5-10 July, 2010, pp. 466-473.

[20] J. Hong, E.-H. Suh, J. Kim, S. Kim, Context-aware system for proactive personalized service based on context history, Expert Systems with Applications, vol. 36, no. 4, 2009, pp. 7448-7457.

[21] A. Beach, M. Gartrell, X. Xing, R. Han, Q. Lv, S. Mishra, K. Seada. Fusing mobile, sensor, and social data to fully enable context-aware computing, Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications (HotMobile), Annapolis, Maryland, USA, 22-23 February, 2010, pp. 60-65.

[22] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, M. Pinkerton. Cyberguide: A mobile context-aware tour guide, Wireless Networks, vol. 3, no. 5, October, 1997, pp 421-433.

[23] Q. Wang, R. Deters. SOA's Last Mile-Connecting Smartphones to the Service Cloud. Proceedings of the IEEE International Conference on Cloud Computing (CLOUD), Bangalore, India, 21-25 September, 2009, pp. 80-87.