On the Evaluation of a Contextual Sensitive Data Offloading Service: the COP case

Francisco Gomes, Windson Viana, Lincoln Rocha, Fernando Trinta

Universidade Federal do Ceará, Brazil
{franciscoanderson, windson, lincoln, fernandotrinta}@great.ufc.br

Abstract. Mobile and context-aware applications are now a reality thanks to the increased capabilities of mobile devices. Frequently, this kind of software characterizes user's situation as well as their profiles to adapt themselves (interfaces, services, content) according to user's contextual data. In the last twenty years, researchers had proposed several software infrastructures to help the development of context-aware applications. However, we verified that most of them do not store contextual data history because researchers have considered mobile devices as resource-constrained devices. Also, few of these infrastructures take into account the privacy of contextual data due to the fact those applications may expose contextual data without user's consent. This article presents a service named COP (Contextual data Offloading service with Privacy support) to mitigate these problems. Its foundations are: (i) a context model; (ii) a privacy model; and (iii) a list of synchronization policies. The COP aims at storing and processing the contextual data generated from several mobile devices, using the computational power of the cloud. We have implemented two performance evaluation experiments of COP. The first experiment evaluated the impact of contextual filter processing in the mobile device and the remote environment. In this experiment, we measured the processing time and the energy consumption of COP approach. The analysis detected that the migration of data from mobile device to a remote environment is advantageous. The second experiment evaluated the energy consumption to send contextual data. The analysis detected that sending contextual data periodically is the best way to save energy.

Categories and Subject Descriptors: D.1.5 [**Software**]: Object-oriented Programming; H.2 [**Database Management**]: Miscellaneous; H.3 [**Information Storage and Retrieval**]: Miscellaneous; K.6 [**Management of Computing and Information Systems**]: Miscellaneous

Keywords: Mobile Cloud Computing, Context-Aware, Middleware, Mobile Device, Offloading, Privacy

1. INTRODUCTION

Over the last few decades, mobile devices have become essential tools for modern society. Smartphones, tablets, and smartwatches are now widespread worldwide [Cisco 2017]. These devices have several sensors (e.g., accelerometer, gyroscope, and GPS) that allow you to gather information about the surrounding environment, as well as data from their users. These intrinsic features of mobile devices allow them to became a key element in the development of context-aware applications [Preuveneers and Berbers 2007]. This kind of system interprets and processes contextual data for characterizing the user's situation (e.g., location, humor, current activity). Context-aware applications monitor context changes in order to adapt themselves to these new conditions, and consequently improve the user experience by offering more relevant services taking into account the user new situation. Context-awareness is an essential feature in the Ubiquitous Computing scenario preconized by Mark Weiser [Weiser 1999], in which users would interact with objects with built-in computing resources to perform ordinary tasks, in a calm and natural way.

One of the hardest challenges for the adoption of Ubiquitous Computing is related to infrastructure required to support these applications [Caceres and Friday 2012]. Despite technological advances,

The authors gratefully acknowledge the support of CAPES, number 1725742.

mobile devices are resource constrained compared to traditional servers and desktops machines. Mobile devices have restricted computing power due to business requirements such as reduced size, lower costs, and energy efficiency. One possible solution to overcome this issue is the use of cloud services in a paradigm called Mobile Cloud Computing (MCC) [Shiraz et al. 2013; Fernando et al. 2013]. Briefly, MCC focuses on offloading processing tasks or even data storage from the mobile device into a remote execution environment with larger computational capacity. According to the scientific literature, the term "offloading" refers to the technique used to migrate storage and processing from a weak device to a more powerful one [Shiraz et al. 2013; Fernando et al. 2013]. While most existing context-aware infrastructures focuses on user's context management and complexity transparency, there are few works addressing data offloading and integration with mobile cloud computing techniques. To the best of our knowledge, we did not find any solution providing a configurable mechanism for sending the contextual data and other user's sensitive information from mobile devices to a remote environment.

The idea of overcoming context-aware application problems with offloading techniques is quite promising, but it also presents many challenges. For instance, data offloading must take into account possible user's sensitive data (e.g., location, apps in use, photos, etc.), which users may not always feel comfortable sharing them with remote devices. So, an offloading approach for context-aware applications should address questions such as: Which data should be anonymized? Who can access user data? Will the system give access to the whole data or only to a part of it?. These types of issues, when neglected, can lead to public disclosure of user's contextual data without their due consent, paving the way for costly legal disputes. In order to address these questions, we proposed a data offloading service with privacy support named COP (Contextual data Offloading service with Privacy support)[Gomes et al. 2016]. The COP is based on two solutions: *(i)* LoCCAM (Loosely Coupled Context Acquisition Middleware) [Maia et al. 2013], a middleware that helps context-aware applications developers get information from the sensors, as well as to separate questions related to contextual information acquisition from the applications business logic; and *(ii)* MpOS (Multiplatform Offloading System) [Costa et al. 2015], a service-oriented architecture that enables developers to mark methods on their applications using annotations, in order to set which methods should be transferred to cloud servers instead of being executed on the mobile device.

COP aims at (i) disseminating contextual data between the mobile device and the cloud environment in a transparent and configurable manner; and (ii) providing an adjustable privacy policy, with which users can restrict the dissemination and diffusion of their contextual information. This paper seeks to answer the following research question: *How can the use of offloading techniques improve or enrich the context-aware support infrastructures for migrating data from the mobile device to a cloud infrastructure?* The COP was evaluated in terms of processing time and the energy consumption. The analysis detected that the migration of data from mobile device to a remote environment is advantageous and that sending contextual data periodically is the best way to save energy.

The remainder of this paper is organized as follows. Section 2 discusses background and important concepts we adopted in this research. Section 3 describes a scenario motivating the need for our approach. In Section 4, we present COP, as well as the context model and the proposed privacy policy. Section 5 presents a proof of concept we developed showing the COP service usage. In Section 6, we describe evaluation experiments that measured COP performance and its energy consumption. Section 7 presents the related works. Finally, Section 8 concludes the paper and outlines possible future work.

2. BACKGROUND

In this section we present the concepts and solutions that underlies our research and paves the way towards our solution.

2.1 Context-Awareness

Context-awareness in Computer Science is a research subject studied for decades, which was first formalized by Schilit and Theimer [1994]. For these authors, context-aware is the ability of a system to discover and react to changes in the environment in which it is situated. Dey [2001] presented the most commonly used "Context" definition, whose focus is on user interaction with his/her application, its environment, and the situation of the resources involved.

Viana et al. [2011] expanded Dey's context definition to remove the restriction that limited context to the elements characterizing the interaction between users and the system. According to the definition proposed by Viana et al., the context elements (e.g., ambient temperature and user location) are defined based on their relevance to the system and on the possibility of the system sensing them at a certain instant of time. The idea is to include systems that acquire relevant contextual data independent of the user interaction, but which will use them for enhancing the system functionalities anytime or even to improve another system (e.g., a mobile photo annotation system). Viana et al. defined Context as the intersection between two dynamic and evolutionary sets, as shown in Figure 1. The first set is the Zone of Interest (ZI) and is composed of all environment information that the system and the user would like to know. The second set is the Zone of Observation (ZO). It is composed of all the environment information that the system and the user can provide or acquire. The intersection between ZI and ZO represents everything the system is interested in, and it can characterize in an instant of time t. So, "Context" is this intersection and the elements composing it will vary according to the system interests and capability to acquire them.

There are several challenges for the development of context-aware mobile apps. We can mention the limited processor power of the mobile devices, the mobile network bandwidth variation, and the device's limited storage. Additionally, the context-aware applications must deal with the diversity of context sources, and with security, privacy, and trust issues. Several support infrastructures (e.g., middleware, frameworks) have been proposed to facilitate the development, extensibility, and reuse of context-aware systems [Knappmeyer et al. 2013; Yurur et al. 2014; Baldauf et al. 2007]. They provide services for the acquisition and management of contextual data. These services allow software engineers to accelerate the development of context-aware applications since they reduce the development complexity or make it transparent. Middleware platforms, which are part of these infrastructures, act as an intermediate layer between the operating system and the application, aiming at providing interoperability and decreasing complexity in the development of context-aware applications. Some middleware platforms contain mechanisms for acquiring contextual data from the environment and the user's device sensors. The generation of higher context level information through inferences and aggregations are another standard feature. They also store context information as a history of sensor and inference data. Some infrastructures use a client/server architecture, in which the user's mobile device performs none or few context management tasks [Knappmeyer et al. 2013].

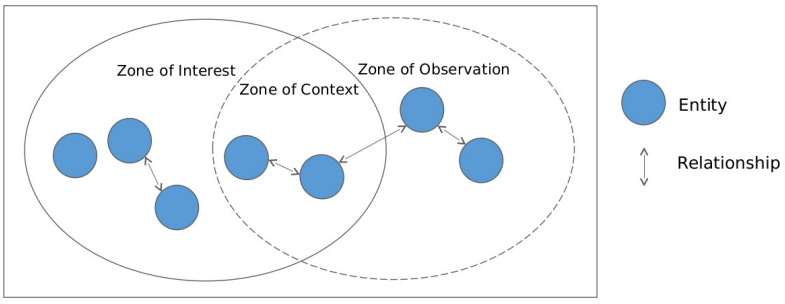


Fig. 1. Context definition (Adapted from [Viana et al. 2011]).

2.2 LoCCAM Middleware

A possible context-awareness benefit is the ability to reduce battery power consumption of mobile devices. Components responsible by capture of contextual information can be installed and/or started to according the interest of applications and stopped when they are not required anymore. LoCCAM (Loosely Coupled Context Acquisition Middleware) [Maia et al. 2013] adopt this definition. It is a context-aware support infrastructure developed by GREat laboratory [1] and it is able to adapt itself according to changes in both zones (i.e., Zone of Observation and Zone of Interest). LoCCAM is a middleware that supports the development and execution of context-aware systems on the Android platform. This middleware adapts the context acquisition process by starting or stopping software components in the mobile device. In the LoCCAM, a CAC (Contextual Acquisition Component) encapsulates each element responsible for capturing contextual data (e.g., device sensors, context aggregators). We classify CACs into physical or logical sensors. Physical CACs are those that only encapsulate access to mobile device sensor information (e.g., accelerometer, temperature, and luminosity sensors). Logical CACs may indicate an aggregation sensor (i.e., the fusion of two or more sensor information) or a component encapsulating an external source of contextual data (e.g., social networks, weather forecast Web service). Logical CACs should provide high-level context information.

LoCCAM uses a memory bus metaphor to provide a shared memory of contextual data, which is split up by CACs and applications. We implemented it as a tuple space. Applications can make direct queries (i.e., request-response type interaction) or subscribe to a context interest (e.g., user's location, environment luminosity). The second case follows a publish-subscribe interaction model. LoCCAM notifies applications when a CAC inserts context of their interest data into the tuple space. LoCCAM decouples the layer responsible for context acquisition from the application layer. Thus, context acquisition occurs in a transparent way. Applications do not need to know the origin of contextual data to use them. Also, LoCCAM adapts its context acquisition layer at runtime aim at optimizing the use of mobile device computing resources. In fact, it manages CACs life cycle according to context interests of applications subscribed to the middleware. LoCCAM has two main modules: the Context Acquisition Manager (CAM) framework and the System Support for Ubiquity (SysSU) module. Figure 2 shows an overview of LoCCAM architecture. CAM framework is divided in Adaptation Reasoner and CAC Manager. Adaptation Reasoner is responsible for maintaining a list of the context interests from all applications running on the mobile device. This component verifies any changes that may occur in this list and, if it is necessary, it makes the middleware intern reconfiguration to satisfy new context interests or to save device's resource (e.g., CPU cycles and free memory). Once the adaptation is established, CAC Manager assumes the task of executing it since it is responsible for managing CAC lifecycle.

SysSU module is an infrastructure that aims at supporting spontaneous interaction in ubiquitous environments [Lima et al. 2011]. SysSU saves contextual data in tuples format (Definition 1) and provides, to upper layers, interfaces to access tuples in a synchronous or asynchronous way. SysSU was designed as a distributed tuple space initially. However, LoCCAM first version supports a unique instance of SysSU, which runs on a single mobile device.

Definition 1 Tuple. *A tuple t is a sequence of n pairs, such that* $t = \langle$ *(n_ 0, v_ 0), (n_ 1, v_ 1), ..., (n_ n-1, v_ n-1)*$\rangle$*, each pair* (n_i, v_i) *consists of a name* (n_i) *and a value* (v_i)*. Each pair is also unique and* $\forall(n_j, v_j), (n_k, v_k) \in t, j \neq k$ *implies in* $n_j \neq n_k$*.*

SysSU intermediates all interaction between applications and LoCCAM. CACs publish contextual data sensed by them in SysSU tuple space. Applications CACs can perform synchronous or asynchronous queries on this context bus. Thus, SysSU acts as a broker for LoCCAM, mobile applications, and CACs. When applications or CACs register their interests in a particular contextual data type,

[1] http://www.great.ufc.br/

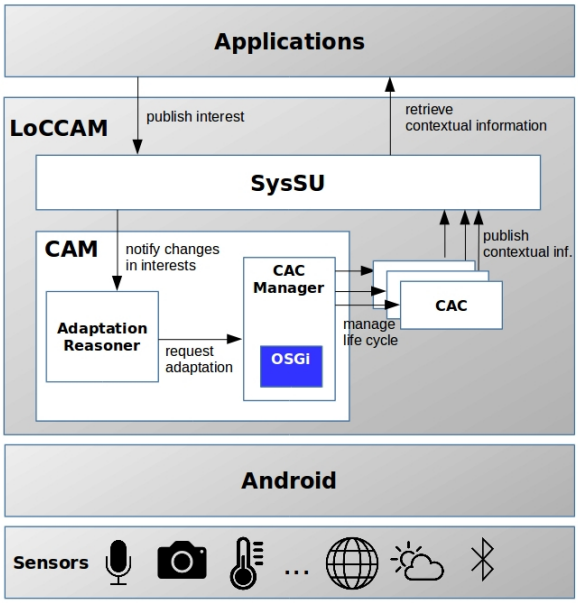


Fig. 2. LoCCAM Architecture (Adapted from [Duarte et al. 2015]).

SysSU notifies the Adaptation Reasoner. This component establishes which adaptations must be made in the context acquisition layer (e.g., start, stop, download CACs). In fact, SysSU is responsible for improving interoperability between LoCCAM components, and it acts to decouple middleware layers and mobile applications. Middleware and the applications use a shared vocabulary for communicating purposes. It represents the contextual data types. The shared vocabulary is a tree of Context Keys (CK). Each key is an identifier associated with the contextual data types of LoCCAM. This key is used to identify CACs responsible for providing contextual data. Also, mobile applications and CACs query contextual data or subscribe to events related to them by using CKs. LoCCAM has a hierarchical scheme based on the Management Information Base (MIB) [2] to organize CKs. In this scheme, a key is composed of a sequence of names separated by dots. For example, CK "`context.device.location`" is used by LoCCAM to identify device location data. In LoCCAM, tuples stored in the tuple space follow a context model and are represented as described in the Definition 2.

Definition 2 LoCCAM. *A tuple in LoCCAM is represented by 5 elements,* $t = \langle$(`contextkey`, "?"), (`source`, "?"), (`values`, "?"), (`accuracy`, "?"), (`timestamp`, "?")$\rangle$, `contextkey` *is used to identify the type of contextual data;* `source` *is used to inform the data source;* `values` *contains the value of the contextual data;* `accuracy` *informs the accuracy of a read operation; and* `timestamp` *provides the instant of time, in milliseconds, in which the data was gathered.*

LoCCAM retrieves contextual information in the SysSu tuple space through contextual filters. Two forms of recovery are possible: (i) pattern matching; and (ii) pattern matching with tuples filter. In the first one, LoCCAM provides a subset of tuple fields (name and value) and all tuples in the tuple space matching the given pattern are returned. For example, if an application wants to read all tuples related to ambient temperature, pattern (1) should be used.

$$q = (\texttt{contextkey}, \text{“}\texttt{context.ambient.temperature}\text{”}) \quad (1)$$

Tuple filters establish a distinct criteria for tuple retrieval. They are composed of a pattern matching and a logical expression. Both are expressions concerning the tuples field names. Also, filters contain

[2] http://www.ieee802.org/1/pages/MIBS.html

possible values that tuples may assume. For example, if the application needs to retrieve only ambient temperature readings whose value is greater than 80 °F, developers just need to write filter (2). First part of the tuple filter consists of a tuple pattern ((`contextkey`, "`context.ambient.temperature`")). Second part is a logical expression that restricts the limit of temperature values to be retrieved (`values` $>$ 80).

$$f = \langle\langle(\texttt{contextkey}, \text{“}\texttt{context.ambient.temperature}\text{”})\rangle, \quad \texttt{values} > 80\rangle \tag{2}$$

2.3 Mobile Cloud Computing

Mobile Cloud Computing (MCC) is a novel paradigm based on three pillars: mobile computing, cloud computing and wireless networks [Fernando et al. 2013]. MCC aims at overcoming limitations of mobile devices regards their performance and energy consumption through the use of cloud resources to increase both computing and storage capacity of these devices. According to [Shiraz et al. 2013], MCC is "the latest practical computational paradigm that extends the vision of utility computing to cloud computing to increase the limited resources of mobile devices". There are several research topics related to MCC, but the main theme among them is *offloading* [Fernando et al. 2013], which is a technique that aims at increasing performance and reducing power consumption of mobile devices through processing or data migration to other infrastructure, typically with more computing power and storage capacity. There are several research questions related to offloading, such as: how, when, where and why it should be performed. Regarding "where", the literature mentions three alternatives: a public cloud, a *cloudlet* or other mobile device. A *cloudlet* can be seen as an execution environment (e.g., desktop or laptop) that interacts with mobile devices in the same WLAN in which the devices are connected to [Satyanarayanan et al. 2009]. The advantage of running on cloudlet is because, in general, WLANs have higher speeds and lower latency over other wireless networks.

Typically, there are two main types of offloading: data and computing offloading [Fernando et al. 2013]. *Computing* offloading is the migration of computer processes from a mobile device to another execution environment in order to extend battery life and/or increase the application performance. *Data* offloading expands mobile device storage capacity by migrating data into cloud repositories, where these data could also be used by other users to refine their applications.

2.4 MpOS Framework

MpOS (Multiplatform Offloading System) is a framework for the development of mobile applications for multiple mobile platforms (Android and Windows Phone), which performs the method offloading technique through RPCs (Remote Procedure Call) [Costa et al. 2015; Rego et al. 2017]. The mobile application interacts with a client library provided by MpOS framework and enables offloading operations to be performed for a particular remote server. MpOS consists of services that run on both mobile device and remote server. These services are responsible for various networking activities, such as discovery and deployment of a service, network profiling, and suitable environment to perform offloading services [Costa et al. 2015]. This last component called Remote Execution Environment (REE), which is where the required offloading operations are performed. During application development, developers must explicitly point out each method that can be executed in a remote environment, when it is advantageous. The developer can also choose whether the offloading will be triggered statically (i.e., always will be offloaded) or dynamically (i.e., the offloading process is triggered based on some environment variables, such as network latency and free memory).

3. MOTIVATING SCENARIO

Based on the concepts of context-aware and MCC, this article seeks to improve the migration of contextual data between the mobile device and the cloudlet, as well as cloudlets to a cloud. The relevance of COP becomes clearer in a motivating scenario. In this way, the functional requirements (FR), non-functional requirements (NFR) and the motivating scenario are presented below.

Requirements List:
FR1 - Agglutinate the contextual data of all users
FR2 - Control the privacy of contextual data
FR3 - Control the sending of contextual data
NFR1 - Contextual data must be accurate

Alexandre is at the opening ceremony of the Rio 2016 Olympic Games. Alexandre and thousands of users are posting photos and tagging them with hashtags with an application running on their mobile devices and suggesting hashtags of photos with similar context (location and time) (NFR1). Whenever possible, the application performs image filter processing in each photo and data offloading on the cloudlet running at the Olympic Stadium (FR3). When Alexander captures a photo, five most talked-about hashtags among the application users during the opening party are recommended (FR1). Thus, data migration allows data sharing to occur. After the ceremony, Alexander goes to a shopping center that also has a cloudlet. He keeps using the application, but fearing that other users will know your location, Alexandre no longer wants to divulge your contextual data. So, he disables sending his data (FR2).

In this scenario, there are two distinct moments. First, Alexander is recommended on the most commented hashtags when opening ceremony is happening. In the second moment, Alexander does not want to make his location public. This scenario aims at showing how interesting it is to use contextual data from more than one user, because it allows data sharing, as well as the need to provide a privacy mechanism capable of filtering only user public data of a particular application.

4. COP: THE PROPOSED SOLUTION

Based on motivation scenario and in order to mitigate problems presented previously, we propose a solution to enable to offloading of contextual data. This solution is a service named COP (Context Offloading service with Privacy support), which is based on: (*i*) a context model; (*ii*) a privacy policy; and (*iii*) synchronization policies.

4.1 Architecture

The proposed solution has a three-tier architecture, shown in Figure 3. The first tier is represented by the mobile device, where the contextual data is captured from its sensors and stored locally. The second tier is represented by a cloudlet, which receives contextual data from mobile devices connected to the same WLAN. The last tier represents a tuple space hosted in a cloud service, which aggregate data from different cloudlets and users who share their contextual data. On the mobile device side, a NoSQL database has been added, which acts as a local cache of context information, until the data offloading process occurs.

In the mobile side, *Synchronizer* component is responsible for retrieving contextual data from the database running on the mobile device and sending it to the remote environment. *Synchronizer* uses different strategies defined by synchronization policies (for more details see Section 4.4). Whenever tuples are sent to the remote environment, the local database is emptied to prevent data overflow on the mobile device. Both cloud and cloudlet tier provide components responsible for managing shared contextual data. In the case of cloudlet, this component is called *Cloud SysSU*, whereas the service

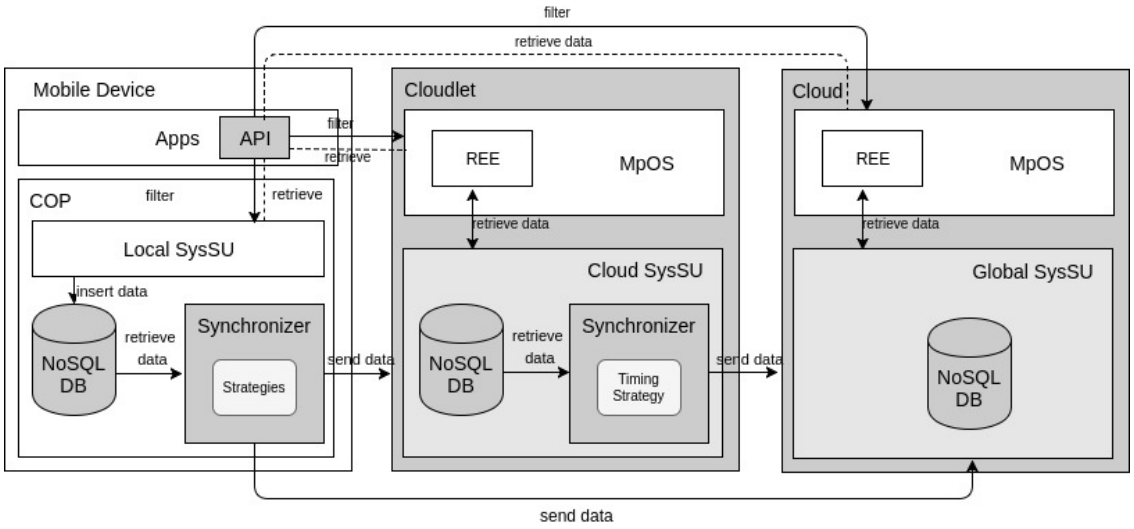


Fig. 3. The COP Architecture.

that aggregates data from all cloudlets and users is called *Global SysSU*. Both have a NoSQL database, similar to the one that runs on the mobile device, which aims at persisting contextual data of mobile devices that use the system.

COP enables mobile devices offload tuples into cloudlets, or into the *Global SysSU*, allowing contextual data from multiple mobile devices to be brought together in one place. In the cloudlet side, *Synchronizer* component send its data periodically to the *Global SysSU*. In the cloud side, there is not *Synchronizer* component, because it is the centralizer. It is important to note that if no cloudlet is found, the contextual data can be sent directly from the mobile device to *Global SysSU*.

COP accesses contextual data through contextual filters in a transparent way with an API (Application Programming Interface). The API is used to integrate the COP with the application. It is responsible, among other features, by directing the filter for on the mobile device (*Local SysSU*), on the cloudlet (*Cloud SysSU*) or on the cloud (Global SysSU). In order to accomplish this task, COP uses MpOS offloading support [Costa et al. 2015]. Among other features, MpOS decides whether offloading processing should be performed or not in the remote environment to improve application performance. Network metrics are used for decision-making, specifically regarding the quality of the connection between mobile devices and the remote server (e.g., connection latency).

COP inherits cloudlet discovery mechanism developed by MpOS, which performs a broadcast on the device's local wireless network to find by an active cloudlet. Even with a cloudlet present, the decision to migrate or not to process is made, as there may be a network degradation at any given time. If this decision indicates that the best location is the mobile device, the contextual filter will be executed only on tuples that are stored in the Local SysSU. It is worthy to run these filters in a cloudlet or a cloud, these filters are offloaded to those locations, specifically into the REE component of MpOS. When these filters run at the REE component, it uses the aggregated context information stored in the *Cloud SysSU* or *Global SysSU*.

4.2 Context Model

COP extends the LoCCAM context model presented previously in section 2.2. This extension required modifications in data represented by tuples. At COP, contextual data from multiple users is stored in a shared tuple space. Thus, new information is required to individualize the data, which indicates

that COP tuples require more fields to guarantee the identification of the context data source, to deal with clock issues in a distributed environment and with data privacy, according to Definition 3.

Definition 3 COP. *A COP tuple is represented by nine elements, five being inherited from Definition 2 and four new fields. They are:* $t = (\texttt{iddevice}, \text{“?”}), (\texttt{idapp}, \text{“?”}), (\texttt{visible}, \text{“?”}), (\texttt{globaltimestamp}, \text{“?”})\rangle$, *where* `iddevice` *represents the ID of the mobile device;* `idapp` *represents the id of the application that is using the service;* `visible` *is used to identify data visibility (more details in Section 4.3); and* `globaltimestamp` *represents the instant, in milliseconds, in which the contextual data is entered into the remote environment.*

The structure presented in (3) characterizes a sample of the context data "ambient temperature" in COP tuple format. Underlined fields have been added to existing fields. In this example, mobile device has ID “0424033418” and uses an application with identification “br.casa.temp”.

$$
\begin{aligned}
t = \langle & (\texttt{contextkey}, \text{“}\texttt{context.ambient.temperature}\text{”}), \\
& (\texttt{source}, \text{“}\texttt{physical}\text{”}), \\
& (\texttt{values}, \text{“80”}), \\
& (\texttt{accuracy}, \text{“0”}), \\
& (\texttt{timestamp}, \text{“239459060969”}) \\
& (\underline{\texttt{iddevice}}, \text{“0424033418”}) \\
& (\underline{\texttt{idapp}}, \text{“}\texttt{br.casa.temp}\text{”}) \\
& (\underline{\texttt{visible}}, \text{“0”}) \\
& (\underline{\texttt{globaltimestamp}}, \text{“239459780932”})\rangle
\end{aligned}
\tag{3}
$$

4.3 Privacy Policy

For dealing with privacy, COP established two specific concepts: sensitive contextual data and reliable cloudlet. In COP, users can explicitly state which data they authorize (or not) to send to a remote environment. Contextual data privacy is related to its visibility, which is represented by field “visible” of COP tuple structure (see Definition 3). If a user has no problem with disclosing their contextual data with other system users, the value of this field should be set to “1” to indicate that visibility is public, otherwise the value is “0” to indicate that visibility is private. In structure shown in (3), the value of `visible` field is “0”. Thus, this contextual data is said to be sensitive and will not be sent to the remote environment.

Reliable cloudlet (also called “gateway” in our proposal) is a user-defined cloudlet which can receive all contextual data gathered by a user, including his/her sensitive contextual data. It could be, for example, a desktop in that user's home. However, private data stored in reliable cloudlets are not forwarded to *Global SysSU*. Thus, according to COP privacy policy, if a mobile device is not connected to a reliable cloudlet when contextual data migration occurs, only public tuples will be migrated. Otherwise, if a device is connected to a reliable cloudlet, all contextual data will be sent, regardless of its visibility.

4.4 Synchronization Policies

Another important point is to define when contextual data should be migrated from mobile devices to a remote environment. We have established different strategies for this proposal, which we called synchronization policies.

Three sending strategies have been specified, namely: (*i*) timing, (*ii*) amount of tuples, and (*iii*) Wi-Fi connection-oriented. Timing strategy means that periodically (e.g., every 30 seconds), tuples

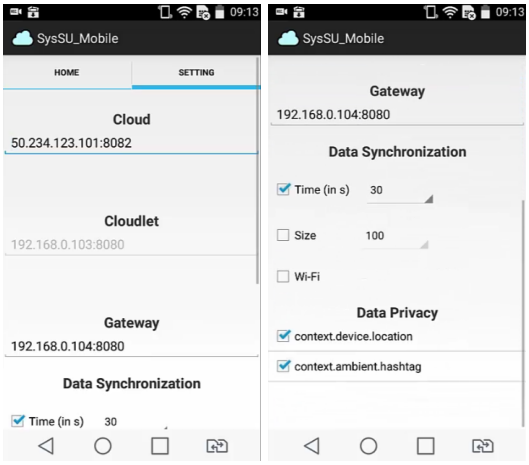
are sent to the remote environment. The period for sending the data is configurable, and if a user does not define explicitly which strategy to use, timing will be the default strategy. The value default is 30 seconds and it is empirical. The amount of tuples strategy implies that if a pre-established number of tuples is reached, these tuples will be sent to the cloud. This number is also configurable. Finally, Wi-Fi connection-oriented strategy defines that only when a Wi-Fi connection is established, the mobile device tuples will be sent to the cloud infrastructure. If a user establishes this policy but uses a 3G/4G connection, contextual data will not be offloaded until he/she connects to a Wi-Fi link.

These strategies can be combined simultaneously, where the one that happens first will be executed. For example, if a user defines that they want to use the timing and amount of tuples strategies at the same time, if the amount of pre-established tuples is reached before the fixed period defined by the Timing strategy, then tuples will be sent to the cloud/cloudlet and vice versa.

5. PROOF OF CONCEPT

An application called MyPhotos has been developed to illustrate the benefits of data offloading using COP. MyPhotos is based on the motivating scenario presented (see Section 3) and show how contextual data can be used to recommend items based on the user's interest in the application. MyPhotos uses COP API to implement two contextual filters for contextual data retrieval. Users can share photos and tag them with hashtags to enable their retrieval easier for other users or applications. Also, photos are also tagged with the GPS location and the instant (date/time) when they were captured by the users' mobile device (NFR1). Since COP stores data from multiple users, MyPhotos application suggests hashtags of photos with similar context (FR1). Two mobile devices were used. The cloudlet used in our experiment was set "unreliable" for both devices. A view of the synchronization and privacy setting screen for this service can be seen in Figure 4 (FR2 and FR3).

Figure 5 shows the MyPhotos architecture. Two CACs were developed to publish the contextual data: Hashtags and Location. This data is published in the COP Service, specifically in Local SysSU, and is sent to the cloudlet through the synchronization policies. The *ReadHashtags* component is responsible for requesting the contextual data from the COP. API performs query on the mobile device or cloudlet, depending on network conditions. Thus, if appropriate, the contextual data from cloudlet will be presented to the application user. The *ImageFilter* component is responsible for applying image filters to the captured photos.



(a) (b)

Fig. 4. Screenshots - COP Service.

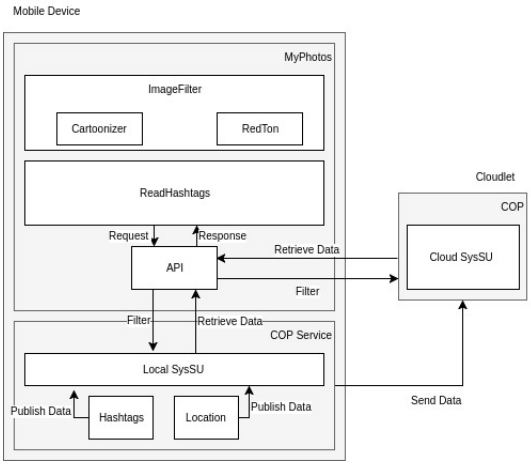
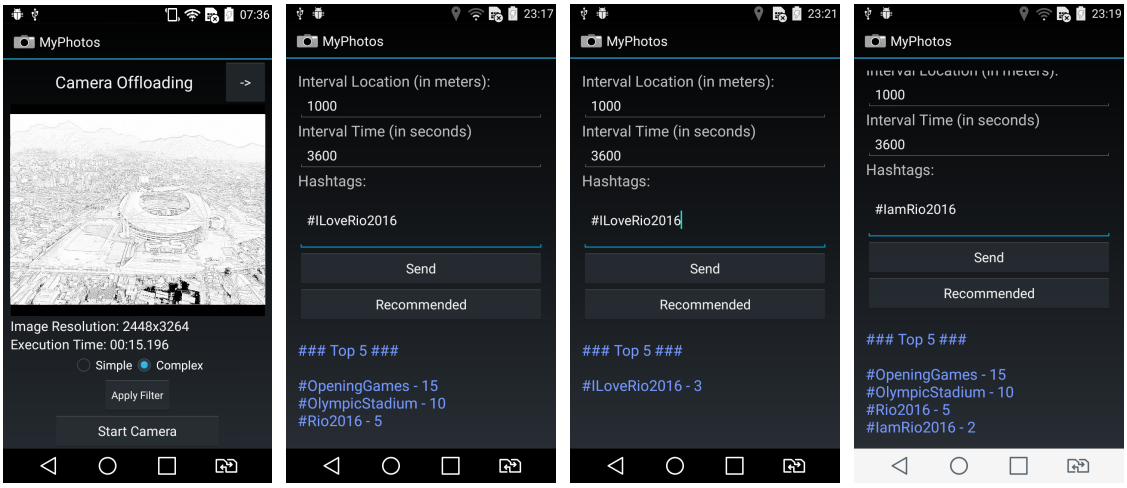


Fig. 5. The MyPhotos Architecture.



(a) D1 - Main Screen

(b) D1 - Recommendation Screen with Wi-Fi Connection

(c) D1 - Recommendation Screen without Wi-Fi Connection

(d) D2 - Recommendation Screen with Wi-Fi Connection

Fig. 6. MyPhotos Running on Two Mobile Devices

During MyPhotos execution, it is required to configure some parameters for the hashtags recommendation: (*i*) location threshold (in meters) and (*ii*) time delay (in seconds). In other words, the user defines how far and how much delayed the data regarding his current geographical position are. Figure 6 shows some screenshots during MyPhotos execution.

The first device (D1) has been configured to make its context data private, other than the second device (D2). In D1, a filter was applied which leaves image similar to a cartoon drawing (See Figure 6(a)). On the next screen (see Figure 6(b)), possible settings for filtering the hashtags are displayed. It is possible to see that some hashtags (#OpeningGames, #OlympicStadium and #Rio2016) are recommended. The user entered a new the hashtag: #ILoveRio2016. Because D1 was configured not to make its data public, this new hashtag was not sent to the cloud remote SysSU. Since there was no Wi-Fi connection, only D1 hashtags would be recommended (see Figure 6(c)). In D2, the hashtag #IamRio2016 was sent and the user received new recommended hashtags (those include by D1), plus #IamRio2016, as shown in Figure 6(d).

6. EVALUATION

This section presents experiments performed to evaluate the performance of mobile applications and energy consumption of mobile devices when using COP. We used two applications in our experiments, called *Integers* and *Sending*.

The *Integers* app was designed to provide contextual data containing integer values. It has no practical functionality and was implemented for test purposes only. For this application, a CAC called *IntegersCAC* was developed, which receives two thresholds, one lower and one upper, and generates contextual data tuples inside this interval. For example, if the CAC receives the value 0 (zero) as the lower limit and the value 100 (100) as the upper limit, it will generate tuples from a loop containing that range and increment the value until it reaches the high threshold (0, 1, 2, ..., 100). At the end of this loop, the COP, more specifically the SysSU, will have 101 contextual data tuples containing integer values. The user can control a number of tuples generated by COP, as well as control the number of tuples that will be retrieved from the contextual filters. Thus it is possible to know exactly the amount of data retrieved from the proposed service. From the contextual filters execution, processing time and energy consumption were measured.

Sending app was designed to provide contextual data from the accelerometer. For this application, three CACs with different publication rates was developed to provide mobile device accelerometer samples. The Android documentation on sensors was used to define how often such CACs published data[3]. This documentation defines constants that correspond to how long the device's sensors must take to publish the information. Constant "SENSOR_DELAY_NORMAL" sets the time of 200 ms. Thus, if this constant is used, the sensor should publish accelerometer values approximately at that frequency. Taking into account the value of this constant, three modes of publication were defined:

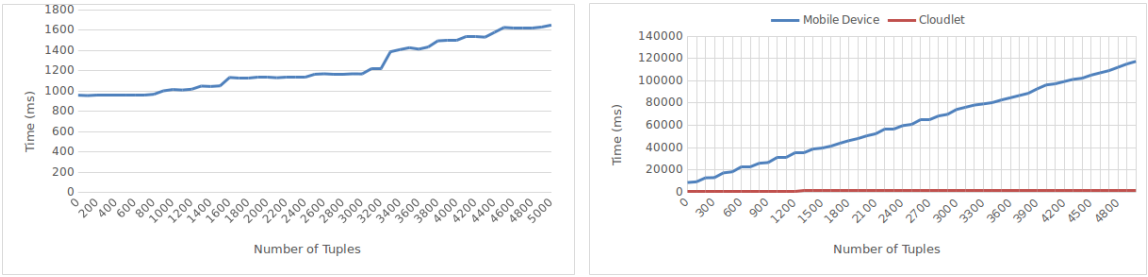
—**Slow**: all publication frequencies are greater than 200 ms;
—**Normal**: all publication frequencies equal to 200 ms;
—**Fast**: all publication frequencies are less than 200 ms.

Following these modes, an accelerometer CAC was developed for each mode. In the application, the user enters a value that defines the period in seconds in which each sending strategy defined in the COP service will be executed and the CAC that will provide the accelerometer tuples. The following subsections present more details about the experiments of each application.

6.1 Experimental Setup for Integers Application

The experiment was carried out in two stages. The first step consisted in measuring time and energy consumption spent on processing contextual filters in the mobile device. The second step performed the same experiment in a cloudlet. The mobile device used in our experiments was an LG G3 Beat smartphone with Qualcomm Snapdragon 400 MSM8226 Cortex-A7 1.2 GHz Quad Core, internal memory of 8GB and 1 GB RAM, running Android 5.0.2. The cloudlet was a laptop running Linux Mint 17.2 64 bit operating system, with 8 GB RAM and Core i5-4200U (1.6 GHz Quad Core) processor. In order to calculate contextual filters execution time, we measured the time spent from the method call responsible for the tuples recovery, until the return of its execution. This elapsed time was calculated in milliseconds. A specialized device called Power Monitor[4] was used to measure the energy consumption during these experiments. This device analyzes energy consumption of the mobile device, and ignores as much device resources as possible that might harm the experiment. One of the metrics analyzed by Power Monitor is power (in Watts). Energy consumption was calculated by multiplying the power and the processing time, which is equivalent to the energy in millijoules.

[3]https://developer.android.com/guide/topics/sensors/sensors_overview.html
[4]https://www.msoon.com/LabEquipment/PowerMonitor/



(a) Cloudlet (b) Mobile Device x Cloudlet

Fig. 7. Contextual Filter Processing Performance

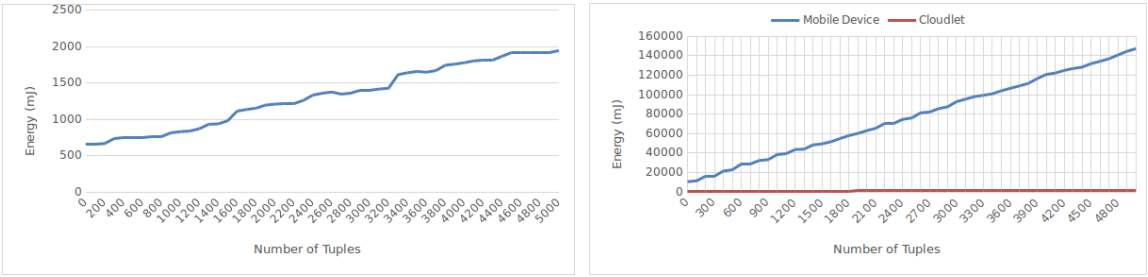
6.1.1 *Experiments for Integers Application.* In the first step, the LG G3 Beat smartphone was used. We requested the creation of 10000 tuples in the range of 1 to 10000. For contextual filters execution, filtering was performed in steps of 100, from the lower threshold 0 to the upper threshold of 5000. Thus, the filters were as follows: 1 to 100, 1 to 200, 1 to 300, ..., 1 to 5000. For each filter, processing time and energy consumption were measured. In the second stage, we used three LG G3 Beat smartphones and the same laptop used in the previous experiment as a cloudlet. In the first smartphone, the interval for generating tuples was set from 1 to 5000. In the second smartphone, the interval for generating tuples was set from 5001 to 10000. Thus, 10,000 tuples were generated as in the first step. For the execution of the contextual filters, the third smartphone was used, filtering steps of 100 from the lower threshold 0 to the upper threshold of 5000. Thus, filters were as follows: 1 to 100, 1 to 200, 1 to 300, ..., 1 to 5000. For each step, processing time and energy consumption were measured. The experiments were performed 30 times and the data collected were stored in a spreadsheet for further analyses.

6.1.2 *Integers Application Results.* Figure 7(a) shows a chart with the result of processing the contextual filter in the cloudlet. The chart shows processing time in milliseconds by a number of tuples retrieved from the generated ones. From this chart, it is possible to verify that the processing time has a little variation with the increased amount of recovered tuples. Thus, it can be stated that the quality of the connection has greater influence on the service than the number of processed tuples. The Figure 7(b) shows a comparative chart of the contextual filter processing time in the mobile device and cloudlet. From this chart, it is possible to see that the mobile device processing time is higher than the cloudlet.

Regarding energy consumption, Figure 8(a) shows the relationship between the energy spent in millijoules (mJ) and tuples retrieved from the tuples in the cloudlet. Since the energy depends on time, this chart has the same behavior as the chart of Figure 7(a). Thus, it is possible to verify that the energy consumption showed a smooth growth with the increase of the number of tuples, presenting only small oscillations. The Figure 8(b) shows a comparative chart of the contextual filter processing energy consumption in the mobile device and the cloudlet. From this chart, it is possible to verify that the energetic cost of contextual filter processing in the mobile device is quite superior to the cloudlet cost. Thus, it can be confirmed that data offloading from mobile devices into a remote environment has advantages regarding saving storage on the device. It also provides increasing speed and less energy consumption when retrieving information from the entire system.

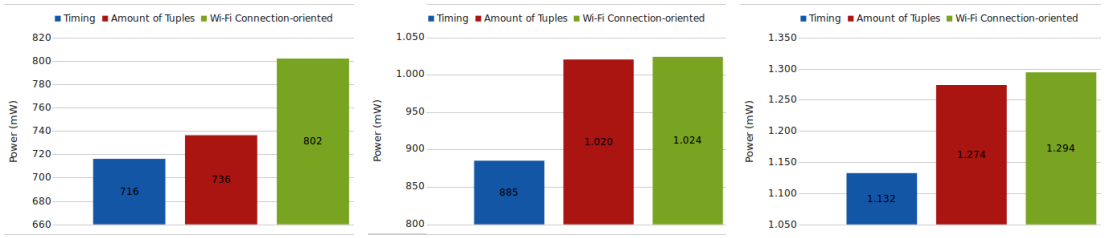
6.2 Experimental Setup for Sending Application

The experiment consisted of measuring energy consumption required to send contextual data from the mobile device to a cloudlet of each synchronization policy strategy. In this experiment, we used an LG G3 Beat smartphone and the same laptop of the previous experiment. Power Monitor was used to measure the energy consumption.



(a) Cloudlet (b) Mobile Device x Cloudlet

Fig. 8. Contextual Filter Processing Energy Consumption



(a) Slow Mode (b) Normal Mode (c) Fast Mode

Fig. 9. Energy Consumption of the Mobile Device Using the Three Accelerometer CACs

6.2.1 *Experimental for Sending Application.* The experiment was performed for each three contextual data submission strategies. For each strategy, three accelerometers modes were used: slow, normal and fast. The experiment consisted of measuring the energy consumption of sending contextual data from the mobile device to a cloudlet during 300 seconds for each of the three synchronization policy strategies. In this experiment, we used an LG G3 Beat smartphone and the laptop, and the Power Monitor measured energy consumption during 300 seconds of sending the contextual data from the mobile device for each of the three synchronization policy strategies. The experiments were performed 30 times and the data collected were stored in a spreadsheet for further analyses.

6.2.2 *Sending Application Results.* The Figure 9 shows three charts with the result of the energy consumption on the mobile device using the three accelerometer CACs (slow, normal and fast). Each chart shows the average power in milliwatts (mW) for each strategy of sending contextual data in 300 seconds and each CAC. The three charts in the Figure 9 show the similar behavior. It is possible to verify that the timing strategy has a lower energy consumption while sending by Wi-Fi connection oriented has greater consumption.

This behavior is justifiable, because in the Timing strategy, COP only tries to send data to the server periodically, reducing the number of requests. The amount of tuples strategy makes many requests to the server, so the power consumption is as poor as in the Wi-Fi connection oriented strategy, which is constantly monitoring if a Wi-Fi link has been enabled on the device and needs to send a larger amount of data than other strategies, because data is only sent when the connection is established.

7. RELATED WORK

Many solutions (e.g., frameworks and middleware platforms) address the development of mobile and context-aware applications [Yurur et al. 2014]. However, among the existing works, few are those that take into account the data migration issues such as data privacy and sensitivity.

Table I. Comparative Between Related Work

Work	Type	Platform	Communication Paradigm	Context Model	Synchronization Policy	Privacy	Requirements
Sensarena	Framework	Android	Request-Response	Key-Value	No	Yes	FR1 FR2
CUPUS	Middleware	Android	Publish-Subscribe	Key-Value	Yes	No	FR1 FR3
CAROMM	Framework	Android	Request-Response	Markup	Yes	No	FR1 FR3
Sahyog	Middleware	Android	Request-Response Publish-Subscribe	Key-Value	No	Yes	FR1 FR2
COP	Middleware	Android	Request-Response Publish-Subscribe	Key-Value	Yes	Yes	FR1 FR2 FR3 NFR1

For instance, Sensarena is a framework proposed in [Messaoud et al. 2016] to help collaborative application development for the Android platform, which consists of three primary entities: two types of mobile applications, and one central server. The first application is a mobile software responsible for context acquisition. The second application treats contextual data requests. The central server has a mechanism to assign context acquisition tasks and data storage. Sensarena provides users with a mechanism for filtering the sensors' data that will be shared. However, it does not offer a synchronization policy for contextual data migration from the mobile device to the remote environment. CUPUS is a cloud-based middleware, which focuses on collaborative applications on Internet of Things environments [Antonić et al. 2016]. Its most valuable property is elasticity. CUPUS implements a mechanism of self-organization of its processing components by adjusting the cloud resources according to the current workload. CUPUS does not treat the privacy of data sent to the cloud.

CAROMM is a framework for mobile applications that use data from multiple users [Sherchan et al. 2012]. This infrastructure aims at facilitating the gathering of sensor data from mobile devices, and correlate these data with social media information in real time. CAROMM design principles are: (*i*) capture distinct types of streaming data from mobile devices; (*ii*) processing, managing, and analyzing captured data with associated contextual information; and ((*iii*) enabling real-time queries from mobile devices. The framework is implemented on Android platform and allows the adjustment of parameters (e.g., the frequency of information sending, the list of enabled sensors). The cloud side uses the services of Amazon AWS [5]. CAROMM also neglects the privacy of contextual data. Sahyog is a middleware proposed by [Bajaj and Singh 2015] and developed for Android platform that supports the development of collaborative applications. Sahyog provides a mechanism to control the privacy of data, where the user can choose to share information with other users or keep it private. Sahyog uses JSON[6] in data exchange, which allows a high flexibility of the parameters for data search. In the representation of these data, there are mandatory parameters that can invalidate information and thus not return that information to the interested users (e.g., timestamp), being a deficiency of that work. Table I details a comparison between related work and the COP service.

8. CONCLUSION AND FUTURE WORK

This paper presents COP, a proposal of a contextual data migration service from mobile devices to a cloud, which deals with data privacy policies and contextual data sensitivity. Our research implements MCC offloading techniques by using a cloudlet environment. We reduced privacy challenges by modifying the context model of LoCCAM and by establishing a privacy policy, which treats the contextual data visibility and its migration to a reliable cloudlet. Even though COP is based on LoCCAM and MPOS, the problems discussed in this paper persist in other context-aware management middlewares.

As future work, we plan to improve the execution of the Global SysSU, so that contextual information present in the cloudlets will be sent to a single logical location, allowing an even greater storage capacity, as well as improve the performance and the quality of contextual filters. This article

[5]https://aws.amazon.com/
[6]http://www.json.org/

focused mainly on WLAN networks, it will be conducted experiments with mobile networks (3G / 4G). Also, we aim to implement a better system scalability management to allow systems to adapt to the increase of user's requests. Additionally, we will carry out new evaluation experiments, which will take into account more metrics (e.g., internal memory). The goal is to comprise COP advantages and limitations better.

REFERENCES

Antonić, A., Marjanović, M., Pripužić, K., and Žarko, I. P. A mobile crowd sensing ecosystem enabled by cupus: Cloud-based publish/subscribe middleware for the internet of things. *Future Generation Computer Systems* vol. 56, pp. 607 – 622, 2016.

Bajaj, G. and Singh, P. Sahyog: A middleware for mobile collaborative applications. In *New Technologies, Mobility and Security (NTMS), 2015 7th International Conference on*. pp. 1–5, 2015.

Baldauf, M., Dustdar, S., and Rosenberg, F. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing* 2 (4): 263–277, 2007.

Caceres, R. and Friday, A. Ubicomp systems at 20: Progress, opportunities, and challenges. *IEEE Pervasive Computing* 11 (1): 14–21, January, 2012.

Cisco, C. V. N. I. Global mobile data traffic forecast update, 2016–2021. *white paper*, 2017.

Costa, P. B., Rego, P. A. L., Rocha, L. S., Trinta, F. A. M., and de Souza, J. N. Mpos: A multiplatform offloading system. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. SAC '15. ACM, New York, NY, USA, pp. 577–584, 2015.

Dey, A. K. Understanding and using context. *Personal Ubiquitous Comput.* 5 (1): 4–7, Jan., 2001.

Duarte, P. A. S., Barreto, F. M., Gomes, F. A. A., Viana, W. C., and Trinta, F. A. M. Critical: A configuration tool for context aware and mobile applications. *In: 39th Annual International Computers, Software & Applications Conference* vol. 1, pp. 159–168, jul, 2015.

Fernando, N., Loke, S. W., and Rahayu, W. Mobile cloud computing. *Future Gener. Comput. Syst.* 29 (1): 84–106, Jan., 2013.

Gomes, F. A., Viana, W., Rocha, L. S., and Trinta, F. A contextual data offloading service with privacy support. In *Proceedings of the 22Nd Brazilian Symposium on Multimedia and the Web*. Webmedia '16. ACM, New York, NY, USA, pp. 23–30, 2016. doi: 10.1145/2976796.2976860.

Knappmeyer, M., Kiani, S. L., Reetz, E. S., Baker, N., and Tonjes, R. Survey of context provisioning middleware. *IEEE Communications Surveys Tutorials* 15 (3): 1492–1519, Third, 2013.

Lima, F., Rocha, L., Maia, P., and Andrade, R. A decoupled and interoperable architecture for coordination in ubiquitous systems. In *Software Components, Architectures and Reuse (SBCARS), 2011 Fifth Brazilian Symposium on*. pp. 31–40, 2011.

Maia, M. E. F., Fonteles, A., Neto, B., Gadelha, R., Viana, W., and Andrade, R. M. C. Loccam - loosely coupled context acquisition middleware. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. SAC '13. ACM, New York, NY, USA, pp. 534–541, 2013.

Messaoud, R. B., Rejiba, Z., and Ghamri-Doudane, Y. An energy-aware end-to-end crowdsensing platform: Sensarena. In *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*. IEEE, pp. 284–285, 2016.

Preuveneers, D. and Berbers, Y. Towards context-aware and resource-driven self-adaptation for mobile handheld applications. In *Proceedings of the 2007 ACM Symposium on Applied Computing*. SAC '07. ACM, New York, NY, USA, pp. 1165–1170, 2007.

Rego, P. A., Costa, P. B., Coutinho, E. F., Rocha, L. S., Trinta, F. A., and de Souza, J. N. Performing computation offloading on multiple platforms. *Computer Communications* vol. 105, pp. 1 – 13, 2017.

Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE* 8 (4): 14–23, Oct, 2009.

Schilit, B. N. and Theimer, M. M. Disseminating active map information to mobile hosts. *IEEE network* 8 (5): 22–32, 1994.

Sherchan, W., Jayaraman, P. P., Krishnaswamy, S., Zaslavsky, A., Loke, S., and Sinha, A. Using on-the-move mining for mobile crowdsensing. In *Mobile Data Management (MDM), 2012 IEEE 13th International Conference on*. pp. 115–124, 2012.

Shiraz, M., Gani, A., Khokhar, R., and Buyya, R. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. *Communications Surveys Tutorials, IEEE* 15 (3), Third, 2013.

Viana, W., Miron, A. D., Moisuc, B., Gensel, J., Villanova-Oliver, M., and Martin, H. Towards the semantic and context-aware management of mobile multimedia. *Multimedia Tools and Applications* 53 (2): 391–429, 2011.

Weiser, M. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.* 3 (3): 3–11, July, 1999.

Yurur, O., Liu, C., Sheng, Z., Leung, V., Moreno, W., and Leung, K. Context-awareness for mobile sensing: A survey and future directions. *Communications Surveys Tutorials, IEEE* PP (99): 1–1, 2014.