Chapter 6

Using Mobile Cloud Computing for Developing Context-Aware Multimedia Applications

- A Case Study of the CAOS Framework

Fernando A. M. Trinta[1], Paulo A. L. Rego[1], Francisco A. A. Gomes[1], Lincoln S. Rocha[1] and José N. de Souza[1]

[1]Federal University of Ceará. GREat Research Group

{trinta,almada,lincoln}@great.ufc.br {pauloalr,neuman}@ufc.br

Abstract

Mobile cloud computing (MCC) and Context-Aware Computing (CAC) are research topics in growing evidence. The former seeks to leverage cloud computing features to improve the performance of mobile applications and reduce the energy consumption of mobile devices, while the latter seeks effective ways to build applications that react to changes in its context environment. This short-course aims at presenting main concepts, solutions, and technologies related to the integration of MCC and context-aware applications. We will present different motivational scenarios, examples of applications, as well as a practical guide to the development of a context-aware multimedia Android application using the framework CAOS. In addition, we will highlight research challenges and opportunities that come with such integration.

Resumo

Mobile Cloud Computing (MCC) e a Computação Sensível ao Contexto são tópicos de pesquisa em crescente evidência. O primeiro procura utilizar recursos da computação em nuvem para melhorar o desempenho de aplicações móveis e reduzir o consumo de energia dos dispositivos, enquanto o último busca formas eficazes de criar aplicações que reajam às mudanças de contexto do ambiente. Este minicurso tem como objetivo apresentar os principais conceitos, soluções e tecnologias relacionadas à integração de MCC e sensibilidade ao contexto. Serão apresentados diferentes cenários motivacionais, exemplos de aplicações, bem como um guia prático de como desenvolver uma aplicação multimídia sensível ao contexto utilizando o framework CAOS. Além disso, serão discutidos desafios e oportunidades de pesquisa relacionados à integração entre os dois tópicos.

6.1. Contextualization

Mobile Devices such as smartphones and tablets have become important tools for daily activities in modern society. These devices have improved their processing power due to faster processors, increased storage resources and better network interfaces. Mobile devices have also been gradually equipped with a plethora of sensors that gather data from the user's environment (e.g., location and temperature). One challenge for mobile and distributed computing is to explore the changing environment where mobile devices are inserted with a new kind of mobile application that take benefits from features of their dynamic environment. These new types of systems are called context-aware applications.

Nowadays, context management and inferences are becoming complex processes, as the amount of data increases and new algorithms are being proposed [Gomes et al. 2016]. In this scenario, cloud services can offer an interesting option by taking on more intensive context management tasks and performing these tasks only once for multiple users. However, since most contextual information is sensed and captured by mobile devices themselves, it is not always clear whether it is wise to send all the sensor data to the cloud for a remote processing or perform the whole processing of contextual data locally by the mobile device. These trade-offs depend on factors such as the amount of data being transferred and the type of processing that is required. According to [Naqvi et al. 2013], the Mobile Cloud Computing paradigm, from a context-sensitive perspective, can be seen as a promising field of research that seeks to find effective ways of doing service in Cloud-aware applications and clients.

Offloading is the main research topic in MCC [Fernando et al. 2013] and represents the idea of moving data and computation from mobile devices with scarce resources to more powerful machines [Rego et al. 2016]. There are several opportunities where computation and data offloading can bring improvements to context-aware mobile applications as well as multimedia applications. Some scenarios include: (i) devices with low processing power can use the cloud to act on their behalf (e.g., rendering images or videos) [Costa et al. 2015], (ii) it would be possible to leverage cloud resources to save energy by delegating tasks away from mobile devices [Barbera et al. 2013], and (iii) to save data storage from mobile devices.

In order to understand different approaches, concepts, and challenges about the integration between Mobile Cloud Computing (MCC) and Context-Aware Computing (CAC), this short-course is proposed as an opportunity to disseminate and improve the understanding of the target audience regarding the union of these two themes. To the best of our knowledge, until the writing of this document, there were no similar initiatives on the subject in any major symposium or conference in Brazil. We will use a framework developed by the GREat research group[1], called CAOS - Context-Aware and Offloading System [Gomes et al. 2017], which is a software platform for the development of context-aware mobile applications based on the Android platform. CAOS supports both data and computing offloading (i.e., it enables the migration of computing and contextual data from mobile devices to cloud platforms in a transparent and an automatic way).

The integration of cloud services is an increasing trend in several areas. This short-course aims to present how this integration for the domain of context-aware applications

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

can be done, focusing mainly on multimedia applications.

6.2. Theoretical Background

6.2.1. Context-Aware Computing

Mobile devices, such as smartwatches, smartphones, tablets, and ultrabooks, have become part of our everyday lives. They allow users to access a vast range of applications on-the-go, from personal agendas to existing social networks, from standalone applications running on the device, to distributed applications interacting with the environment [Herrmann 2010]. An important benefit of using an application on a mobile device is the possibility to use it anywhere and anytime. In fact, it can be seen as a step toward to achieve the Mark Weiser's Ubiquitous Computing [Weiser 1991] vision, embedding the computation in the user's devices and turning the user-interaction more soft and natural.

To achieve the calm interaction between users and computers is a complex task. Applications running on mobile devices have to interact with a dynamic environment, where available users, devices, and resources change over time. Therefore, the software running on these devices, interacting with users and the environment, must be designed from scratch taking into account these changes and adapting itself in order to achieve the desired behavior. The ability to perceive changes in the environment and adapt its behavior to meet these changes is called context-awareness. [Preuveneers and Berbers 2007].

Before precisely define context-aware we must first understand what context is. In that sense, we introduce in the following the most spread definition of context and context-awareness given by Anind K. Dey [Dey 2001]. Next, we introduce the Viana's *et al.* [Viana et al. 2011] context-aware definition adopted in this document.

> "*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*" [Dey 2001]

> "*A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.*" [Dey 2001]

Viana *et al.* [Viana et al. 2011] extends Dey's definition taking into account the dynamic nature of the context structure and its acquisition. In that sense, not only the values of the context elements evolve over time, but also the number of context elements changes. According to Viana *et al.* [Viana et al. 2011], the elements composing the context depends on the system's interest and the chance to observe them. So, once the system execution evolves, the set of observed context elements also evolve. In a nutshell, the context can be determined by the intersection of two sets of information in an instant t. The first one is called Zone of Interest (ZoI), comprising the set of contextual elements relevant to the system. The second set is called Zone of Observation (ZoO) and it describes the set of contextual elements that can be collected by the system. Figure 6.1 shows a Venn diagram with these two sets of information. Their intersection is the Zone of Context (ZoC) that is composed of any information that can be described and observed. All

three zones may change over time. For instance, ZoO may change when a sensor becomes unavailable; the ZoI may change when the system changes its interests, not requiring specific contextual information any longer. At last, the ZoC changes when changes on ZoI and ZoO affect their intersection.

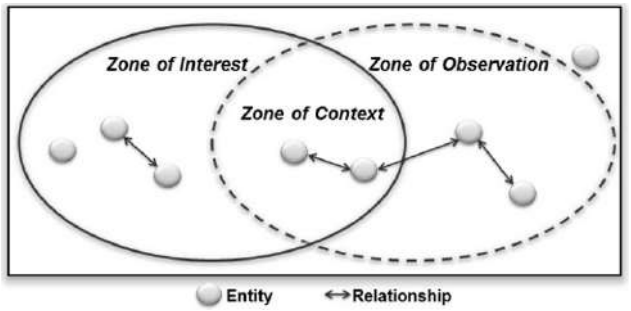


Figure 6.1. Context definition proposed by Viana *et al.* [Viana et al. 2011].

In short, context-awareness is the ability of the computer system to discover and react to its context changes. Mobile applications can benefit from context-awareness to provide personalized services to users and adapt their structure and behavior accordingly in order to save or optimize the mobile device resources usage. The mobile devices constrains (e.g., battery power, bandwidth, and storage capacity) and issues related to the context management (e.g., capture, process, and delivery), security, privacy, and trust turn the design and development of context-aware mobile applications a challenging task.

A considerable research effort has been done to provide solutions to support the development of context-aware systems. In particular, context management infrastructures to support the development and execution of context-aware applications received significant attention [Baldauf et al. 2007, Bettini et al. 2010]. These infrastructures are typically implemented as a middleware platform or a horizontal framework, providing support to capture, aggregate, store, infer, and delivery context information. Usually, such infrastructure follows a thin client/server architecture style, where the application running on the mobile device (client) performs no or few tasks related to the context management and all or most of the context management processing is performed on the server side. As a drawback, these infrastructures are not adaptable, in a sense they are not able to adapt its own context management mechanism [Da et al. 2011].

6.2.1.1. Solutions

The solutions found in the literature review were classified according to the following taxonomy divided into seven aspects: (i) **Research Subject** - this aspect concerns to the kind of software infrastructure is provided to support development of context-aware application (e.g., framework and middleware); (ii) **Target-Platform** - represents the mobile platform and development technology on which the solution was evaluated/implemented (e.g., Java, Android, Titanium, RESTful); (iii) **Interaction Paradigm** - this aspect captures the kind of interaction paradigm used between applications and the context management infrastructure (e.g., tuple-space, publish/subscribe, and request/response); (iv) **Context Type** - this aspect comprises the type of context each solution supports, following

the Yurur *et al.* [Yurur et al. 2016] classification: device context (e.g., available network interfaces, battery load, CPU and memory usage); physical context (e.g., temperature, noise level, light intensity and traffic conditions) user context (e.g., personal profiles, location and people surrounding them, social situation); and temporal context (e.g., time, day, week, month, season and year); (v) **Dependency** - this aspect indicates if the solution has any software dependence, such as an external framework or library; (vi) **Modularity** - this aspects indicates if the context management concerns implementation are clearly separated from the application business logic; and (vii) **Cloud Interaction** - this aspect concerns to the usage of Cloud Computing concepts to manage context information. Table 6.1 summarized the review and classification according proposed taxonomy.

All papers report solutions that were implemented as a framework (S02, S04, S05, S07, S10, and S12-S15) or as middleware platform (S01, S03, S08, S09, and S11) to manage context information. Only one solution (S06) combines both framework and middleware strategies. Most of them are implemented on top of Android platform (S01-S05, S08, S12-S15). The other three solutions focus on RESTful services (S07 and S09) and cross-platform technology (S10). These solutions implement different coordination models to perform interaction between context management infrastructure and the mobile applications. Most of them support both synchronous request-response interaction (S01, S02, S03, S07-S011) and asynchronous publish-subscribe interaction (S02-S07, S10, S11, S13-S15). Only three solutions provide support for tuple-space interaction model (S06, S08, and S12). All these solutions support different context data, which vary from raw sensor data to personal user information extracted from social networks. Most solutions do not have third-parties dependencies, except S06, S08, S11, and S13 solutions. These four solutions depend on specific implementations of OSGi specification[2]. Once most of the solutions are implemented as a framework or a middleware platform, the context management concerns are well modularized. Finally, regarding the cloud integration aspect, only a few solutions (S10, S12, and S15) use cloud services to manage context information, but none of them mention to have support for offloading techniques.

6.2.1.2. LoCCAM

The LoCCAM (Loosely Coupled Context Acquisition Middleware) [Maia et al. 2013] is a context management infrastructure that adopts the mobile device as the center point of context acquisition and decision. LoCCAM provides a transparent and self-adaptive context data acquisition approach, gathering context information both locally and remotely from the device. It also uses a novel mechanism to support adaptation in the way context information is collected and inferred, following the Viana *et al.*'s definition [Viana et al. 2011] (see Section 6.2.1). An overview of LoCCAM's software architecture is given in the Figure. 6.2. Basically, LoCCAM can be divided into two main parts: (1) the SysSU (System Support for Ubiquity) module; and (2) the CAM (Context Acquisition Manager) framework.

The SysSU [Lima et al. 2011] module provides a coordination mechanism based on tuple spaces and event-based notifications. Such mechanism enhances the coupling

[2]https://www.osgi.org/developer/specifications/

Table 6.1. The Context-Awareness Solutions Survey Summary.

ID	Paper	Research Subject	Target-Platform	Interaction Paradigm	Context Type	Dependency	Modularity	Cloud Interaction
S01	[Mane and Surve 2016]	Middleware	Android	Request/Response	Device Context Physical Context User Context Temporal Context	–	Yes	No
S02	[Da et al. 2014a]	Framework	Android Java SE	Request/Response Publish/Subscribe	Device Context Physical Context User Context Temporal Context	–	Yes	No
S03	[Da et al. 2014b]	Middleware	Android Java SE	Request/Response Publish/Subscribe	Device Context Physical Context User Context Temporal Context	–	Yes	No
S04	[Williams and Gray 2014]	Framework	Android	Publish/Subscribe	Device Context	–	Yes	No
S05	[Ferroni et al. 2014]	Framework	Android	Publish/Subscribe	Device Context Physical Context User Context Temporal Context	–	Yes	No
S06	[Maia et al. 2013]	Framework Middleware	Android	Publish/Subscribe Tuple-Space	Device Context Physical Context User Context Temporal Context	OSGi	Yes	No
S07	[Chihani et al. 2013]	Framework	RESTful	Request/Response Publish/Subscribe	Physical Context	–	Yes	No
S08	[Punjabi et al. 2013]	Middleware	Android	Request/Response Tuple-Space	Device Context Physical Context	OSGi JSON	Yes	No
S09	[Dogdu and Soyer 2013]	Middleware	Java ME RESTful	Request/Response	Physical Context	– –	Yes	No
S10	[Doukas and Antonelli 2013]	Framework	Titanium	Request/Response Publish/Subscribe	Device Context Physical Context User Context Temporal Context	–	Yes	Yes
S11	[Curiel and Lago 2012]	Middleware	RESTful	Request/Response Publish/Subscribe	Device Context Physical Context User Context	OSGi	Yes	No
S12	[Buthpitiya et al. 2012]	Framework	Android	Tuple-Space	Physical Context	–	Yes	Yes
S13	[Carlson and Schrader 2012]	Framework	Android	Publish/Subscribe	Device Context Physical Context User Context Temporal Context	OSGi	Yes	No
S14	[Lee et al. 2012]	Framework	Android	Publish/Subscribe	Physical Context	–	Yes	No
S15	[Mitchell et al. 2011]	Framework	Android	Publish/Subscribe	Device Context Physical Context User Context Temporal Context	–	Yes	Yes

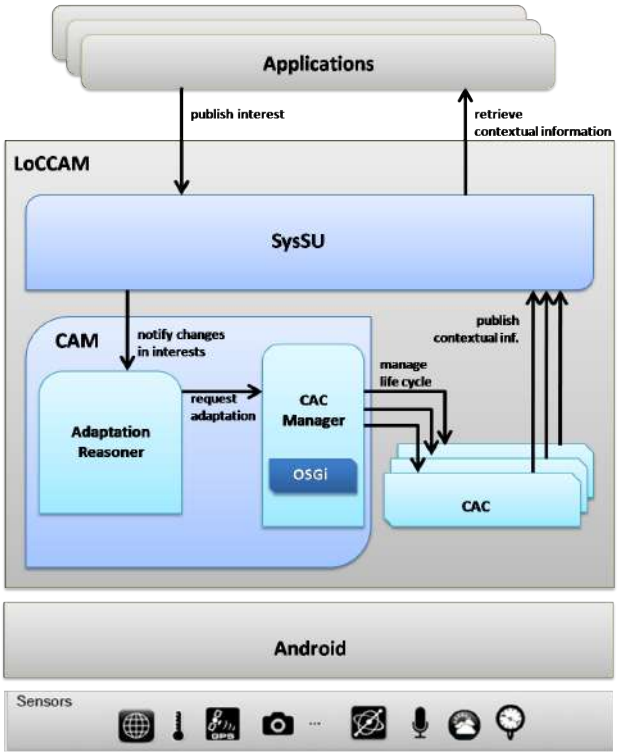


Figure 6.2. The LoCCAM Architecture [Duarte et al. 2015].

between applications and the context acquisition layer. This acquisition layer is based on software-based sensors called CAC (Context Acquisition Component) and can be classified into physical or logical ones. The physical CACs are those that only encapsulate the access to mobile device sensor information (e.g., accelerometer, temperature, and luminosity). On the other hand, the logical CACs are those that can use more than one mobile device sensor and information from other sources (e.g., social networks and weather service on the Internet) to provide high-level context information. Furthermore, LoCCAM offers a common vocabulary that allows CACs and applications exchange context information. Applications subscribe SysSU for the contextual information access by publishing their interests using its common vocabulary. Then, LoCCAM tries to find the more suitable CAC to provide such information. Such communication is based on the concept of Context Keys. They serve as a shared vocabulary to represent each type of contextual information that can be accessed. Each Context Key provides a unique name that must be used by CAC to determine the contextual information it publishes. This key is used by applications to make subscriptions on SysSU. For example, a Context Key that represents the ambient temperature in Celsius scale could be "`context.ambient.temperature.celsius`". To improve the context data selection, applications can use the concept of a filter, which defines a set of fine-grained selection criteria to precisely select the desired context information.

The CAM framework is divided into two modules: Adaptation Reasoner and CAC (Context Acquisition Component) Manager. The Adaptation Reasoner is responsible for maintain the applications subscription list of interest and keep it updated. When any change occurs in this list, the Adaptation Reasoner builds a reconfiguration plan to adapt

the context acquisition layer and sends it to the CAC Manager responsible for performing it. The CAC Manager controls the CAC's lifecycle using a specific implementation of OSGi framework[3]. The CAC Manager can install, uninstall, activate, and deactivate CACs at runtime, according to the application's interest regarding a contextual information access and the reconfiguration plan. A CAC is only activated when an application subscribes SysSU asking for specific contextual information provided by such CAC.

6.2.2. Mobile Cloud Computing

Mobile cloud computing is a new paradigm that incorporates three heterogeneous technologies (mobile computing, cloud computing, and networking) and aims to reduce the limitations of mobile devices by taking advantage of ubiquitous wireless access to local and public cloud resources. Such resources are used to augment mobile devices computing capabilities, conserve local resources, extend storage capacity, and enrich the computing experience of mobile users.

Several authors have also defined mobile cloud computing:

> "*Mobile cloud computing is an integration of cloud computing technology with mobile devices to make mobile devices resource-full in terms of computational power, memory, storage, energy, and context awareness. Mobile cloud computing is the outcome of interdisciplinary approaches comprising mobile computing and cloud computing.*" [Khan et al. 2014]

> "*Mobile cloud computing is a rich mobile computing technology that leverages unified elastic resources of varied clouds and network technologies toward unrestricted functionality, storage, and mobility to serve a multitude of mobile devices anywhere, anytime through the channel of Ethernet or Internet regardless of heterogeneous environments and platforms based on the pay-as-you-use principle.*" [Sanaei et al. 2014]

According to these definitions, the mobile cloud computing model is composed of mobile devices, wireless networks, and remote servers, where mobile devices use wireless technologies to leverage remote servers to execute their compute-intensive tasks or storage data.

Computation offloading is a popular technique to increase performance and reduce the energy consumption of mobile devices by migrating processing or data from mobile devices to other infrastructure, with greater computing power and storage. The concept of offloading, also referred as cyber foraging, appeared in 2001 [Satyanarayanan 2001] and was improved in 2002 [Balan et al. 2002], in order to allow mobile devices to use available computing resources opportunistically.

Migrating computation to another machine is not a new idea. The traditional client-server model is also widely used for this purpose. In fact, the ideas behind the

[3]Apache Felix distribution - http://http://felix.apache.org/

concept of computation offloading date back to the era of dumb terminals that used mainframes for processing. With the adoption of personal computers (e.g., desktops and notebooks), the need to migrate computation has decreased. Nevertheless, with the advent of portable devices, a new need for remote computing power has emerged [Dinh et al. 2011].

According to [Verbelen et al. 2012], offloading can be executed on a remote virtual machine-based environment (e.g., public clouds) or on any machine that is in the same WLAN in which mobile devices are connected to. In the latter case, the remote execution environment is known as cloudlet [Satyanarayanan et al. 2009], and its primary goal is to deliver a better quality of service since Wi-Fi networks are less congested (i.e., they have higher speeds and lower latency) than mobile networks.

It is important to highlight that offloading is different from the traditional client-server model, in which a thin client *always* migrates computation to a remote server. Depending on the availability and the condition of the network, which is highly influenced by users' mobility, offloading may not be possible or advantageous. Therefore, it is important to state that, in mobile cloud computing, the concept of computation offloading is usually implemented by a special program structure, or a design pattern, that enables a piece of code to execute locally on the mobile device or remotely, without impacting on the correctness of the application [Zhang et al. 2012].

In the next subsections we present a further discussion about the types of mobile applications and explore the concept of offloading, by addressing the questions *How?*, *When?*, *Where?* and *Why?* to perform computation offloading.

6.2.2.1. Types of Mobile Applications

According to [Kovachev et al. 2011], applications for mobile devices can be classified as offline, online and hybrid. Offline applications, which are also called native applications, act as fat clients that process the presentation and business logic layer locally on mobile devices, usually with data downloaded from remote servers. Also, they may periodically synchronize data with a remote server, but most resources are available locally, rather than distributed over the network.

Some advantages of native applications are: good integration with features of the device, optimized performance for specific hardware and multitasking, always available capabilities, even without Internet access. On the other hand, the main disadvantages are: they are not portable to other platforms and dependent exclusively on storage and processing power of the device, which may not be enough to execute certain types of computation.

Online applications assume that the connection between mobile devices and remote servers is available most of the time. They are usually based on Web technologies and present a few advantages when compared to offline applications, such as the fact that they are multiplatform and are accessible from anywhere. Nevertheless, they also present some disadvantages: the typical Internet latency can be a problem for some types of applications (e.g., real-time applications), difficulty in dealing with scenarios that require keeping the communication session opened for extended periods of time, and no access

to device's sensors such as camera or GPS.

Hybrid applications are targets of mobile cloud computing and offloading researchers. They can execute locally (as native applications) when there is no connection to remote servers. However, when there is connectivity between mobile devices and remote servers, they can migrate some of the computation to be performed out of the mobile device or access web services (as online applications). The idea is to combine the advantages of online and offline applications.

In the rest of this short-course, we consider only hybrid applications, which are the ones that can benefit from offloading techniques.

6.2.2.2. Where to Perform Offloading?

Mobile devices use remote resources[4] to improve application performance by leveraging offloading techniques. These remote resources are public cloud, cloudlets or another mobile device.

Public Cloud

The execution of services in the public cloud is common among mobile application developers since they can leverage features such as elasticity and connectivity to social networks to improve services. Applications such as Gmail and GoogleDocs are examples of online applications that require smartphones to be connected to the Internet all the time to be able to access data.

In order to connect to the public cloud, mobile devices can use mobile networks (e.g., 2G, 3G e 4G) or a Wi-Fi hotspot. Figure 6.3 illustrates a mobile device accessing an application that relies on Internet connection.

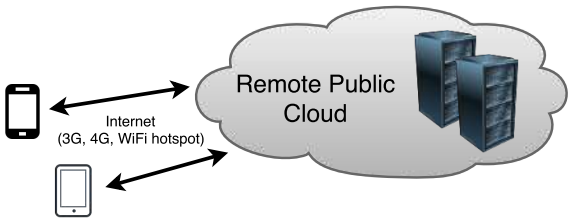


Figure 6.3. Public cloud as remote execution environment.

Cloudlet

The idea of using nearby servers to reduce the connection latency and improve users' quality of experience is being used since the emergence of the term cyber foraging, introduced by [Satyanarayanan 2001]. On that time, the authors used servers in the vicinity to handle the limited capabilities of mobile devices.

The concept of cloudlet is newer and was proposed in [Satyanarayanan et al. 2009]

[4]In this short-course, when we use terms such as remote servers, remote resources or remote execution environments, we refer to any real or virtual remote equipment where the computation of mobile devices can be migrated to.

also to use resources of servers that are close to mobile devices. The difference is that the authors have used virtual machines, in a trusted environment, as remote servers. Today, several studies have used the terms offloading or cyber foraging to indicate that there is a migration of data and/or computation of mobile devices to another location; as well as the terms cloudlet and surrogate that have been used indiscriminately to indicate a computer or cluster of computers directly connected to the same WLAN of mobile devices. Figure 6.4 illustrates the concept of cloudlets by showing mobile devices and remote servers connected to the same wireless network.

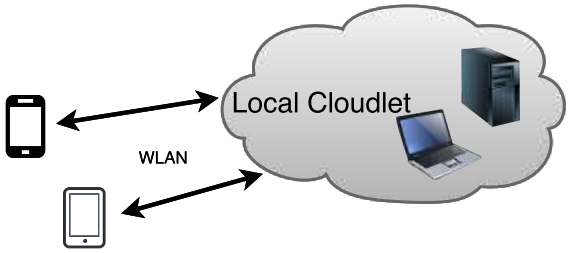


Figure 6.4. Cloudlet as remote execution environment.

The vision of [Satyanarayanan et al. 2009] is that cloudlets would be conventional equipment deployed such as access points, which would be located in public areas (e.g., cafes, pubs, and restaurants) so that mobile devices could connect via Wi-Fi networks and perform offloading without facing high latency and the typical variation of Internet bandwidth.

Other Mobile Device

There is another approach in the literature, commonly referred as "mobile cloud", that considers using other mobile devices as the source of resources, especially to perform a computation. Figure 6.5 illustrates this approach, in which mobile devices are usually connected using a peer-to-peer network and create a cluster (or cloud) of devices.

The idea behind this solution is to enable people, that are in the same place and share the same interests, to create a cloud of mobile devices and share their resources aiming to compute tasks more quickly or reduce energy consumption.

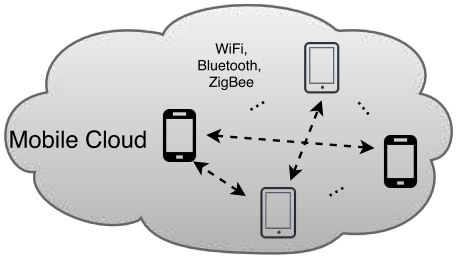


Figure 6.5. Cluster of mobile devices as remote execution environment.

Hybrid Environment

A hybrid environment is composed of two or more of the environments mentioned above. In Figure 6.6, we can see an example of this type of environment, where a mobile

device is part of a "mobile cloud" and can perform offloading by leveraging cloudlets and public cloud as remote execution environments.

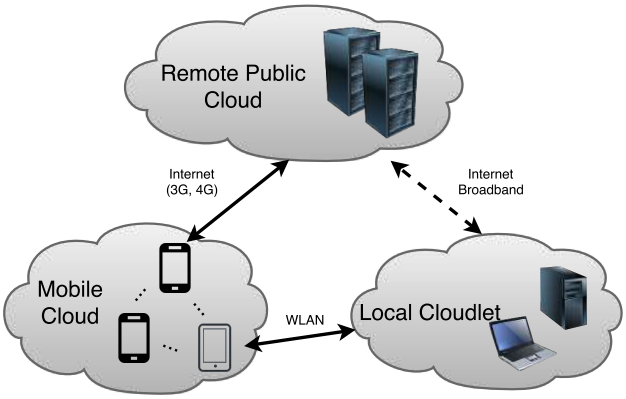


Figure 6.6. Hybrid remote execution environment composed of cloudlets, public cloud and a cluster of mobile devices.

6.2.2.3. Why to Perform Offloading?

Given the limited resources of mobile devices, researchers have mainly used computation offloading to enhance applications' performance, save battery power, and execute applications that are unable to run due to insufficient resources. Therefore, the main reasons for performing offloading are:

- **Improve Performance:** when performance improvement (i.e., reduce the execution time) is the main goal;
- **Save Energy:** when energy efficiency (i.e., reduce energy consumption) is the main goal;
- **Other:** when energy and performance are not the main reasons for performing offloading. Instead, the main reason may be to improve collaboration, extend storage capacity or reduce monetary costs.

6.2.2.4. When to Perform Offloading?

The reasons for performing offloading presented in the previous section are directly related to the decision of when to offload.

[Kumar et al. 2013] present an analytical model to answer the question "When the offloading technique improves the performance of mobile devices?". The model compares the time to process an application task on a mobile device ($\frac{W}{P_m}$) and the time to transfer the data and perform the computation out of the device ($\frac{D_u}{T_u} + \frac{W}{P_c} + \frac{D_d}{T_d}$), either on a public cloud instance or on a cloudlet. The model considers the following parameters: W is the total computation to be performed, which may be expressed in MI (million instructions); P_m and P_c are, respectively, the processing power of the mobile device and the cloud

(or cloudlet), which may be expressed in MIPS (million instructions per second); D_u is the amount of data sent from the device to the cloud (bytes), while D_d is the amount of data received by the device (bytes); and T_u and T_d are the upload and download rate (bytes/second) respectively.

$$\frac{W}{P_m} > \frac{D_u}{T_u} + \frac{W}{P_c} + \frac{D_d}{T_d} \tag{1}$$

Analyzing the model presented in (1), we can see that, to improve performance, the computation should be heavy (high value to W) and the communication between mobile device and the cloud should be brief (low value to $\frac{D_u}{T_u} + \frac{D_d}{T_d}$), either by transferring few data, or by having a high throughput.

[Kumar et al. 2013] highlight that increasing the difference between the processing power of mobile devices and clouds does not bring great impact to the decision. This fact can be observed in (2), when we consider a cloud K times faster than a smartphone ($P_c = K \cdot P_m$). Clearly, for large values of K, Equation (1) can be simplified to $\frac{W}{P_m} > \frac{D_u}{T_u} + \frac{D_d}{T_d}$, which means that the time required to transfer the data between mobile device and cloud has a key role in deciding when to perform offloading. Figure 6.7 depicts the trade-off between the amount of communication and computation for deciding whether or not to perform offloading.

$$\begin{aligned} \frac{W}{P_m} - \frac{W}{K \cdot P_m} > \frac{D_u}{T_u} + \frac{D_d}{T_d} &\Rightarrow \frac{W}{P_m} \cdot \left(\frac{K-1}{K}\right) > \frac{D_u}{T_u} + \frac{D_d}{T_d} \\ &\Rightarrow \frac{W}{P_m} > \frac{D_u}{T_u} + \frac{D_d}{T_d} \end{aligned} \tag{2}$$

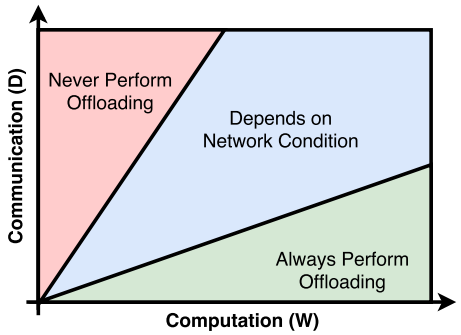


Figure 6.7. Offloading decision trade-off. [Kumar and Lu 2010]

Other researchers focus on different objectives such as throughput maximization [Xia et al. 2013], energy saving [Kharbanda et al. 2012], and cost reduction [Kosta et al. 2012].

Regarding energy savings, [Kumar and Lu 2010] list four basic approaches to save energy and extend the battery lifetime of mobile devices:

1. **Adopt a new generation of semiconductor technology.** Unfortunately, as transistors become smaller, more transistors are needed to provide more functionalities and better performance; as a result, the power consumption actually increases;

2. **Avoid wasting energy.** Whole systems or individual components may enter standby or sleep modes to save power;

3. **Execute programs slowly.** When a processor's clock speed doubles, the power consumption nearly octuples. If the clock speed is reduced by half, the execution time doubles, but only one-quarter of the energy is consumed. Therefore, executing applications more slowly is a good option to save energy. That is indeed done in some smartphone with DVFS to put devices in energy-saving mode. However, it is important to assess the trade-off between performance and energy saving.

4. **Eliminate computation altogether.** A mobile device does not perform computations; instead, the computation is performed somewhere else, thereby extending the battery lifetime of mobile devices.

The last approach can be realized by using offloading techniques to migrate computations from mobile devices to remote servers. We refer to [Magurawalage et al. 2014] for an energy savings analytical model for answering the question "When the offloading technique saves energy of mobile devices?".

Regardless of the reasons that motivate the use of offloading mechanisms, the solutions also differ regarding the **Offloading Decision**, which is related to how an offloading solution decides when to perform offloading. In short, the offloading is called:

1. **Static:** when the developer or system defines prior to execution (at design or installation time) what parts of the application should be offloaded and to where;

2. **Dynamic:** when the framework/system decides at runtime which parts of the application should be offloaded and where to offload, based on metrics related to the current condition of the network, mobile devices, and remote server.

6.2.2.5. What to Offload?

When the application is not available on a remote server, there is a need to migrate parts of (or the whole) application to the server along to the computation request and input data. In order to separate the intensive mobile application components that operate independently in the distributed environment, a partitioning procedure can be used to partition the application at different levels of granularity [Liu et al. 2015].

Partitioning of an application can be done automatically by the offloading system, or it can be provided by developers using a code markup (e.g., annotations on Java programming language). In the latter case, developers add some kind of syntactic metadata to the application source code to identify the components that are candidates to be offloaded. This markup process is usually done by applications' developers at the design phase and involves examining the complexity and dependency of methods.

Several strategies to perform offloading were proposed in the literature, and they differ in relation to which parts of the application are sent to be executed out of mobile devices. Thus, the parts of applications that are most commonly offloaded are as follows:

- **Methods:** when methods are used to partition the application (e.g., remote procedure calls);
- **Components/Modules:** when entire modules or components of an application are executed on another resource execution environment. It involves using specific frameworks for designing and developing modular applications (e.g., OSGi) or just a group of classes that may or may not be coupled;
- **Threads:** when threads of an application are migrated between mobile devices and remote execution environments. It usually involves performing changes on the Android virtual machine (DalvikVM);
- **Whole Application:** when virtualization techniques are used to run clones of mobile devices and the entire state of the application process is migrated and executed out of mobile devices (e.g., in a virtualized clone device). In this case, a synchronization module is usually required for keeping the applications running on mobile device and clone updated, by replicating all changes.

It is common for mobile applications to interact with sensors embedded in mobile devices (e.g., GPS and camera), thus, offloading the whole application may be impractical. That is why [Cuervo et al. 2010] defend that a fine-grain strategy leads to large energy savings as only the parts that benefit from remote executions are offloaded.

6.2.2.6. How to Perform Offloading?

There is no unique answer to the question "How to perform offloading?". In fact, several offloading frameworks/systems/middlewares have been proposed to address such question, and they usually differ regarding *What?*, *Where?*, *When?* and *Why?* to perform offloading.

Existing offloading solutions have applied various strategies and mechanisms to handle steps of the offloading process, such as cloudlet discovery, resources profiling, application partitioning and offloading decision [Sharifi et al. 2012]. Since the strategies used are quite varied, we have performed a literature review that helped us to identify the common approaches used for performing offloading and also have allowed us to create a taxonomy to assist in the classification of related works.

The categories of the taxonomy related to the question "How to perform offloading?" are discussed below:

Method Annotation: when an offloading solution supports granularity of methods and some type of syntax for marking methods is used to identify the methods prone to be offloaded:

1. **Yes:** when syntactic metadata is added to the application source code. The annotation is usually done by developers at design phase, and involves examining the complexity and dependency of methods;

2. **No:** when there is no annotation or when the framework uses a profiler component to collect information and annotate the relevant methods in an automatic fashion.

Decision Module Location: the module responsible for decision-making usually executes compute-intensive operations to decide when and where to perform offloading, which inevitably consumes resources of the mobile device when the module is executed locally. Despite the computation cost reduction when executing the decision module out of a mobile device, this solution imposes more communication costs, which is a clear trade-off that must be considered.

1. **Mobile Device:** when compute-intensive operations of the decision module are executed on mobile devices;
2. **Remote Execution Environment:** when compute-intensive operations of the decision module are executed out of mobile devices.

Decision Module Features: regards approaches and techniques used by a decision module.

1. **Online Profiling:** when the decision module uses measurements (collected at runtime) of different metrics of the environment, network, and application to improve the offloading decision;
2. **Historical Data:** when the solution collects and uses historical data to improve the decision module.

Metrics Used for Decision: regards the metrics considered by a decision module for deciding when and where to offload.

1. **Based on hardware:** when the solution uses metrics related to the hardware of mobile devices and remote servers (e.g., memory, CPU, battery);
2. **Based on software:** when the solution uses metrics related to the software of mobile devices and remote servers (e.g., size of data that will be transferred, execution time, code size or complexity, interdependence between modules or components);
3. **Based on network:** when the solution uses metrics related to network conditions (e.g., connection type, latency, jitter, packet loss, Wi-Fi signal strength, throughput).

Supported Platform/Programming Language: regards mobile platforms and programming languages supported by a solution.

1. **Android**;
2. **Windows Phone**;
3. **iOS**.

Discovery Mechanism: considers whether a solution uses any mechanism for discovering remote execution environments (usually cloudlets or other mobile devices).

1. **Yes:** when a solution automates the remote execution environment discovery;
2. **No:** when the address of a remote execution environment is somehow provided in advance by developers or when a DNS-based service discovery is used.

6.2.2.7. Taxonomy

During the literature review that we have performed, we identified several approaches used in offloading solutions. Then we categorized such approaches in groups based on the questions *What?*, *Why?*, *When?*, *Where?* and *How?* to perform offloading. Inspired by those questions and also in [Sharifi et al. 2012] and [Liu et al. 2015] taxonomies, we propose a taxonomy for offloading solutions.

Figure 6.8 depicts the proposed taxonomy, which is based on the aforementioned questions about offloading. Therefore, *What?* is related to the offloading granularity discussed in Section 6.2.2.5, *Why?* is related to the offloading objective discussed in Section 6.2.2.3, *When?* is related to the offloading decision discussed in Section 6.2.2.4, while *Where?* is related to the remote execution environments discussed in Section 6.2.2.2 and *How?* is related to all categories discussed in Section 6.2.2.6.

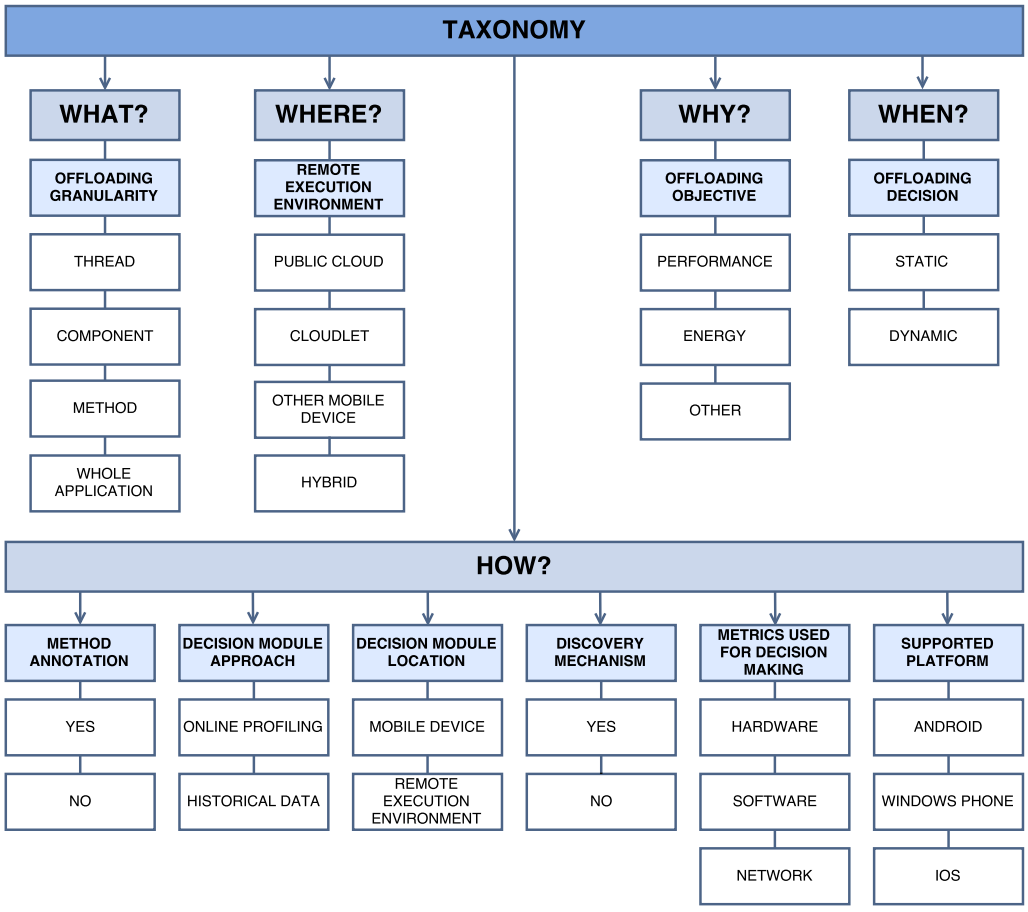


Figure 6.8. Taxonomy for offloading solutions.

6.2.2.8. Solutions

Several solutions have been developed in response to the challenges offered by MCC. In this section, we present some of these solutions, and we use the proposed taxonomy to classify them.

Regarding the offloading granularity, solutions like eXCloud [Ma et al. 2011] and MECCA [Kakadia et al. 2013] perform offloading of the whole application by leveraging virtualization techniques to run a clone of the mobile device. eXCloud uses a compact version of the Java virtual machine, called JamVM, and migrates the state of the application by transferring the Java virtual machine stack from the mobile device to its clone, which is running on the cloud. MECCA supports the Android platform and migrates an entire application process to a clone and allows the user to access the screen of the device using VNC (Virtual Network Computing).

Other solutions like μCloud [March et al. 2011] and MACS [Kovachev et al. 2012] perform offloading of modules/components of the application. Such solutions usually require the use of specific frameworks for designing and developing modular applications (e.g., OSGi and Android Services). For instance, to use MACS, developers have to design their applications using standard Android services, where the compute-intensive components must be implemented as services. Since Android services use inter-process communication channels to perform remote procedure call, MACS intercepts the requests sent to the services and decide whether the request must be executed on local or cloud services.

CloneCloud [Chun et al. 2011] and COMET [Gordon et al. 2012] are examples of frameworks that perform offloading of threads, which usually involves modifying the Android virtual machine to allow the seamless transfer of application's threads. Both solutions require a modified version of the Android operation system and also a clone of the mobile device running on the cloud.

The works that implement offloading of methods usually leverage techniques like Java Annotations to allow developers to identify the methods that are candidates to be offloaded. MAUI [Cuervo et al. 2010] is a framework developed in .NET for Windows Phone 6.5 that uses remote procedure calls to execute methods outside of mobile devices. MAUI uses an approach based on a proxy to intercept methods and redirect them to a server running on the cloud. On the other hand, Scavenger [Kristensen and Bouvin 2010] is a framework developed in Python that uses annotation of methods (using the concept of Python Decorators), and its decision module uses online profiling and historical data analysis to decide whether a method must be executed on mobile device or cloud.

The solutions AIOLOS [Verbelen et al. 2012], ThinkAir [Kosta et al. 2012], and ARC [Ferrari et al. 2016] use methods as offloading units, and they were developed for the Android platform. AIOLOS relies on code refactoring to generate OSGi components for a mobile application at building time. Despite executing OSGi components on the cloud, the decision of when to offload is based on methods provided by the components. On the other hand, ThinkAir relies on Java annotations to identify the methods that can be offloaded and allow such methods to be executed in parallel on multiple servers; while ARC is a framework where a method can be opportunistically offloaded to any available device that can be accessed through the wireless LAN (i.e., other mobile device or a dedi-

cated server, such as in the cloudlet concept). MpOS [Costa et al. 2015][Rego et al. 2016] is a framework that also uses methods as offloading units, but it can be used to develop Android and Windows Phone applications. The framework was developed in our research group and is discussed in more detail in the next section.

Regarding the remote execution environment, most works perform offloading to cloudlets or cloud environments (e.g., eXCloud, MpOS, ThinkAir, μCloud, CloneCloud, AIOLOS, MAUI, Scavenger). ARC and DroidCloudlet [El-Derini et al. 2014] are some of the solutions that use mobile devices as remote execution environments. In DroidCloudlet, the authors propose the creation of a mobile cloud of Android devices that can be used to improve performance and save energy of a device by offloading methods to other devices.

Regarding the offloading decision, μCloud uses static decision, while the other mentioned solutions support dynamic decision. Besides that, the solutions have different offloading objectives. While MpOS, COMET and eXCloud only focus on improving the performance of mobile applications; and Scavenger and μCloud only care about saving energy, ThinkAir, CloneCloud, DroidCloudlet, and AIOLOS consider both objectives (e.g., by solving multi-objective optimization problems).

6.2.2.9. MpOS

MpOS (Multiplatform Offloading System) is a framework for developing mobile applications that support offloading of methods for Android and Windows Phone platforms. MpOS was developed to address the lack of an offloading solution for multiple platforms, and its main goal is to improve performance.

To use MpOS, developers must mark the methods that can be executed out of the mobile device, and they can choose whether the offloading decision will be static or dynamic. If the dynamic offloading is defined, the Offloading System (illustrated in the Figure 6.9) checks if it is worthwhile to perform the offloading operation before sending the method to the remote server. Otherwise, when the static offloading is defined, MpOS verifies only if the remote server is available before performing offloading. In order to decide when to offloading, the decision module uses metrics (e.g., latency, download, and upload rate) that are online collected by the Network Profiler module.

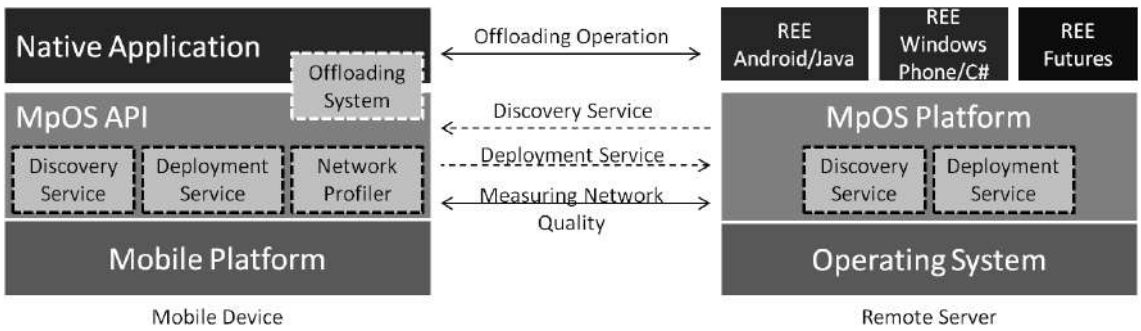


Figure 6.9. Overview of MpOS main components

MpOS has a Discovery Service that uses a multicast-based discovery mechanism

to identify nearby servers (cloudlets) and applications already running on the remote server. If the application is not running on the server, MpOS automatically sends the application and all dependencies to the remote server by means of the Deployment Service. All deployed applications are handled by the Remote Execution Environment (REE) module, which instantiates the applications in different endpoints. In order to support multiple platforms, MpOS provides Android and Windows Phone REEs, which run Java and C# code respectively. Once an application is instantiated on the REE, the mobile device can directly access the application's endpoint and perform offloading. More details about MpOS can be found in [Costa et al. 2015] and [Rego et al. 2016].

6.2.3. Context-Aware and MCC Integration

The advances in mobile computing have made the use of contextual information increasingly present in today's mobile applications. The adaptation of the applications according to contextual situations will improve the users' quality of experience while running the most diverse applications. However, as mentioned earlier in this document, the inference process of the user's contextual situation requires processing and data resources that may sometimes be unsuitable for processing locally on mobile devices.

This issue also occurs when the multimedia data is required to infer users' contextual situation. The use of complex data such as video, audio or images for determining the current context is pointed out as a future trend for mobile applications, but also brings processing requirements that become a problem for mobile computing environments with uncontrolled characteristics due to resource scarcity, low storage capabilities, intermittent connectivity, and power constraints.

In this sense, computing and data offloading fits like a glove to allow the integration of more advanced techniques for contextual inference (such as the use of machine learning algorithms, among others), while also allowing performance gains and energy savings for mobile devices.

It is important to notice that many research projects and reports that address the integration between Context-aware Mobile Computing and Mobile Cloud Computing use "context" as the situation in which a mobile application is running (network condition, remaining battery and/or size of parameters data) to decide whether is worthy to migrate data or processing to the cloud. This document presents a different point of view. We focus on how the management of contextual information may be enhanced (for instance, better or faster inference procedures) with the help of cloud services.

According to [Naqvi et al. 2013], the use of cloud services by context-aware mobile applications is a future trend with no turning back, even with the lack of reports about how this integration should be made. Once adaptation processes are becoming more and more complex, context analysis requires appropriate and robust services.

According to a brief literature review, a few solutions seek to integrate the management of contextual information with concepts of Mobile Cloud Computing, or even just cloud computing.

One of these is the CARMiCLOC [Aguilar et al. 2015], a reflexive middleware architecture based on autonomic computing, which uses cloud services to provide scal-

ability, self-adaptability, integration, and interoperability of context-aware applications. CARMiCLOC uses reflection to inspect its state and ensures a dynamic behavior of its operation. However, the use of cloud resources is restricted only to the storage of the contextual data obtained by sensors. Despite its merit, the cloud potential is underutilized in the proposal, because cloud resources are not used for data processing remotely in the Cloud.

[OSullivan and Grigoras 2016] presents CAMCS (Context Aware Mobile Cloud Services), a mobile cloud middleware solution that has been designed to deliver cloud-based services to mobile users while respecting the goal of providing an integrated user experience of mobile cloud applications and services. CAMCS supports the application partitioning, where some functionalities run on cloud services. Integration with contextual management services uses a component called Context Processor. This component can promote the customization of actions to be taken by the applications, which is performed by other component called Cloud Personal Assistant (CPA). CPA can perform task processing in the cloud on behalf of the user, asynchronously. CAMCS allows the use of historical data and ontologies, which are still prohibitive on mobile devices, due to their intensive processing.

6.2.3.1. Motivating Scenarios

In order to present some examples where context-aware mobile computing and MCC may improve the user experience, this subsection presents some possible scenarios for such integration.

Crowdsensing. Mobile crowdsensing refers to an approach where a large group of people uses mobile devices capable of sensing and computing (such as smartphones, tablet computers, wearables) to collectively share data and extract information to measure, map, analyze, estimate or infer (predict) processes of common interest[Liu et al. 2016]. This situation may enable to detect some contextual situations based on shared data from multiple users. MCC can improve this aggregation and processing of contextual situations based on hierarchies, for example, by aggregating contextual data from users connected to the same cloudlet, and infer some contextual situations in a collaborative way[Xiao et al. 2013][Gomes et al. 2016].

Healthcare and Well-being. With the advancements and increasing deployment of microsensors and low-power wireless communication technologies, the studies conducted on healthcare domain have grown, particularly the studies regarding the recognition of human activities. In this case, information corresponding to human postures (e.g., lying, sitting, standing, etc.) and movements (e.g., walking, running, etc.) can be inferred in order to provide useful feedbacks to the caregiver about a patient's behavior analysis [Yurur et al. 2016]. Recognizing such activities depends on monitoring and analyzing contextual data such as vital sign (e.g., heart rate, pressure level, and respiration rate), which might be aggregated by mobile devices. MCC can improve the execution and save energy of mobile devices when offloading the compute intensive operations required to recognize complex events.

Augmented Reality. A typical mobile augmented reality system comprises mo-

bile computing platforms, software frameworks, detection and tracking support, display, wireless communication, and data management [Chatzopoulos et al. 2017]. Contextual data is important for providing information about the user's environment, which might be used to present personalized content and improve user's experience. The problem is that vision-based applications are almost impossible to run on wearables and very difficult on smartphones since they require capable GPUs [Pulli et al. 2012]. MCC can, therefore, be used to offload the execution of heavy computations to a powerful remote device.

6.3. Context-Aware and Offloading System (CAOS)

CAOS is a software platform that assists developers to create context-aware applications for the Android platform and provides offloading mechanisms to delegate the migration of methods and contextual data from mobile devices into cloud platforms. CAOS is based on two solutions: (1) LoCCAM (Loosely Coupled Context Acquisition Middleware) [Maia et al. 2013], a middleware that helps context-aware applications developers to get information from sensors, as well as to separate questions related to contextual information acquisition from the applications business logic; and (2) MPoS (Multiplatform Offloading System) [Costa et al. 2015][Rego et al. 2016], a service-oriented architecture that enables developers to mark methods on their applications using annotations, in order to identify which methods can be transferred to cloud servers instead of being executed on the mobile device.

In order to conceive CAOS, we surveyed frameworks and middlewares that support both context-aware and offloading features. This survey provides us an insight of good design decisions that were used to build CAOS. According to these design principles, we designed a software architecture that supports both method and data offloading into cloud infrastructures. We implemented a prototype of CAOS, and we conducted experiments to evaluate its impact on performance and energy consumption of Android applications [Gomes et al. 2017].

Before showing how to use CAOS to develop context-aware Android applications, the next section introduces CAOS architecture and main components.

6.3.1. Architecture and Components

CAOS rests on a traditional client/server architecture, where mobile devices act as clients for services running on the top of cloud/cloudlets infrastructures. Figure 6.10 shows the CAOS architecture where its main components are divided into two groups: mobile side and cloud side components.

6.3.1.1. The Mobile Side

CAOS mobile side is composed by 10 (ten) components: CAOS (which is responsible for synchronizing the startup of other components of the mobile side), *Offloading Monitor*, *Offloading Client*, *Discovery Client*, *Authentication Client*, *Profile Monitor*, *Offloading Reasoner Client*, *Context-Acquisition Middleware*, *Tuple-Space Based Context-Bus* and *Context Synchronizer*. Each one of these components is detailed as follows.

The *Discovery Client* component uses a mechanism based on UDP/Multicast com-

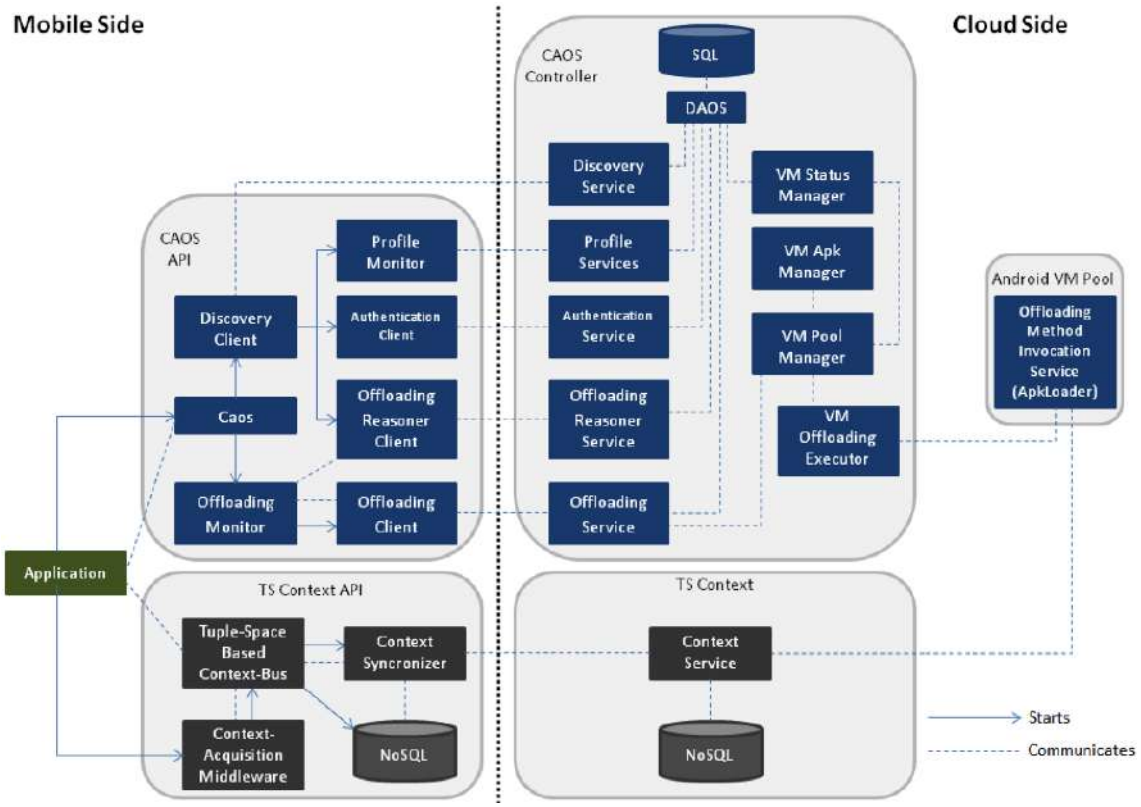


Figure 6.10. Overview of CAOS main components

munication to discover CAOS Controllers running in the user's local network (i.e., cloudlets). Once the *Discovery Client* detects a CAOS Controller, the *Authentication Client* authenticates the mobile application and sends device data to the cloud side to keep the list of devices attached to a specific CAOS Controller.

In CAOS, application's methods can be marked by developers with a Java annotation @*Offloadable*, which denotes that those methods can be executed out of the mobile device. The *Offloading Monitor* is the components responsible for monitoring the application execution and intercepting the execution flow whenever an annotated method is called. After intercepting the method call, the *Offloading Monitor* asks the *Offloading Reasoner Client* module whether it is possible to start the offloading process or not.

The *Offloading Reasoner Client* assists the offloading decision using a decision data-structure that is asynchronously received from the *Offloading Reasoner Service*. In CAOS, the decision whether is worthy or not to offload a method is performed in two steps: one at the cloud side, and another on the mobile side. The cloud side keeps receiving profiling data from each mobile device connected to its infrastructure and creates a two-class decision tree [Rego et al. 2017] with the parameters that should be considered to decide if it is worth to offload a method, such as latency, parameters types, and so on. Once the decision tree is created or updated, it is sent to the mobile device, so the *Offloading Reasoner Client* only has to enforce the decision based on current values of the monitored data and the decision tree structure.

Figure 6.11 presents an example of offloading decision tree for a dummy applica-

tion. In this example, if the upload rate is equal to 200 Kbps and RTT is equal to 100 ms, the method `Bar` must be executed locally. Moreover, in all cases the method `Foo` must be executed on the remote server.

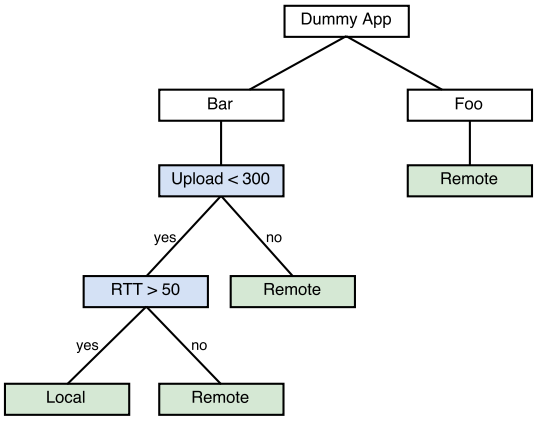


Figure 6.11. Example of offloading decision tree. [Rego et al. 2017]

By using the received data structure, the *Offloading Monitor* can decide locally when it is worth performing offloading. If the answer is negative, the method execution flow is resumed, and its execution is performed locally. Otherwise, when the answer is positive, the *Offloading Monitor* requests the *Offloading Client* to start the method offloading process, which in turn transfers the method and its parameters to the *Offloading Service* in the cloud side.

The *Profile Monitor* component is responsible for monitoring the mobile device environment (e.g., network bandwidth and latency, power and memory status) and sends such information periodically to the *Profile Services*, which will be used in the *Offloading Reasoner Service* (cloud side), that generates the offloading decision tree based on the mobile device information and then sends it back to the *Offloading Reasoner Client*.

The *Context-Acquisition Middleware* component is a new version of the former CAM component in the LoCCAM framework, which has been adapted for this project. The original component has been extended to provide a better integration with cloud/-cloudlets. A new optimized manager was built from the scratch to control CAC's lifecycle, removing OSGi dependency.

This component manages the software-based sensors lifecycle (i.e., search, deploy, start and stop) that encapsulates how context information is acquired. These sensors are called *Context Acquisition Component - CAC* and can be classified into physical or logical ones. The physical CACs are those that only encapsulate the access to mobile device sensor information (e.g., accelerometer, temperature, and luminosity). On the other hand, the logical CACs are those that can use more than one mobile device sensor and information from other sources (e.g., social networks and weather service on the Internet) to provide high-level context information. This middleware also provides an API to build new CACs and incorporate them into new and existing applications.

All context information acquired by CACs is stored in the *Tuple-Space Based Context-Bus module*, which is a new version of the former SysSU module. The *Tuple-*

Space Based Context-Bus stores the context information in a tuple-based format and deliver such information to applications using event notification via a contextual event-bus. This new version has a new feature, the Tuple-Space Based Context-Bus that stores more than one tuple for a particular sensor, creating a list of samples. The initial version of SysSU stored only the last collected sample, overlapping the previous one. This feature may improve inferences using a larger amount of contextual data.

All context information of each mobile device connected to a CAOS Controller is sent to the Context Service (cloud side) to keep a database of contextual information history. The idea is to explore the global context (i.e., the context of all mobile devices) to provide more accurate and rich context information. The *Context Synchronizer* exchanges contextual data between the mobile and the cloud sides. This data migration is important because, in CAOS, filters can be performed in both local context information repository (*Tuple-Space Based Context-Bus*) and global context repository (*Context Service*). If an application has an @*Offloadable* marked method that accesses context information using the filter concept; it can benefit itself from the offloading process to access the global context repository. This can be done using two filters: one to be executed locally (on the mobile device) and another that runs in the global context repository when the method is offloaded to the cloud. The decision of which filter will be executed is performed automatically by CAOS.

6.3.1.2. The Cloud Side

The CAOS cloud side is composed by 11 (eleven) components: *Discovery Service*, *Profile Services*, *Authentication Service*, *Offloading Reasoner Service*, *Offloading Service*, *VM Pool Manager*, *VM Apk Manager*, *VM Status Manager*, *VM Offloading Executor*, *Context Service*, and *Offloading Method Invocation Service*.

The *Discovery Service* provides correct endpoints information so that clients can access CAOS services. The *Authentication Service* is responsible for saving device information and controlling which devices are currently connected to the CAOS Controller. The *Profile Services* is a set of services which receives device monitored data related to the network quality and local execution time of offloadable methods, in order to keep historical records of executed methods. These records will be used during the offloading decision tree creation process.

The *Offloading Reasoner Service* is responsible for processing the mobile device profile data and creating the decision tree that will be used by the *Offloading Reasoner Client* component (on the mobile side) to decide about the offloading process execution. The *Offloading Service* receives offloading requests directly from *Offloading Client* and redirects them to the *VM Pool Manager*. When the offloading process finishes, the *Offloading Service* returns the result to the *Offloading Client* and persists offloading information.

In CAOS, offloaded methods are executed on Android Virtual Machines running on traditional x86 machines. The *VM Pool Manager* component is responsible for providing an environment that redirects offloading requests to a proper Android Virtual Machine where the offloading execution happens. In order to run a method from a specific applica-

tion, CAOS requires that the corresponding deployment packages (a.k.a. APK) of CAOS compliant applications must be stored in a special folder on the CAOS platform. The *VM Apk manager* pushes all APK files to all reachable Android Virtual Machines listed in this special folder. The *VM Status Manager* is responsible for monitoring and maintain information (e.g., the current number of offloading executions) about each virtual machine managed by the VM Pool Manager in a repository. The *VM Offloading Executor* component is responsible for requesting the offloading execution in an Android Virtual Machines calling the *Offloading Method Invocation Service*, which runs on the virtual machine and performs the offloading method execution.

The *Context Service* acts like a global context repository, and stores all context information data sent from all mobile devices connected to the CAOS Services. It maintains this context information in a NoSQL database and provides a proper interface that can be used by logical CACs to access this information to generate high-level context information for an application running on the mobile device.

Related to our architecture, it is possible to store the private contextual data in special cloudlet, called *Gateway*. In short, a Gateway is a cloudlet with privacy policies where its data are chosen by each user to be sent to Cloud or not. This approach filters all information that the user does not want to share with other users, but wants to use for personal purposes. This information is stored in a gateway, but not sent to the cloud.

6.3.1.3. Implementation Details

The technologies employed in the CAOS reference implementation are described in the following. In the mobile side, the CAOS modules were implemented using the Android SDK on Android Studio IDE. The Couchbase (a NoSQL database) was adopted to maintain a local cache of context information history to be offload to a remote side following the CAOS context data offloading strategy. The Bouncy Castle encryption API was chosen to improve the CAOS wireless communication security. The JCoAP (Java Constrained Application Protocol) was adopted as a protocol to achieve the standardization level requested by the project sponsor.

In the cloud/cloudlet side, the CAOS Controller modules were implemented using Java SE/EE and deployed on Apache TomCat web container. All server side services were implemented as RESTful Web services using the Jersey framework. All data in CAOS Controller are stored in a PostgreSQL database using the Hibernate framework. The Context Service stores the context tuples on a MongoDB (a NoSQL database). The VirtualBox hypervisor technology was employed in the Android environment virtualization in the server side using an ISO Android x86 version (4.4.2) Kitkat.

6.3.2. CAOS Study Case

This section presents the steps required to develop a context-aware multimedia Android application with offloading capability, called MyPhotos. Such application allows users to apply filters to their photos and share them on social networks along with *hashtags*, to make it easier to search for them. Also, photos can be tagged with contextual information, such as location, date and time at which they were captured. MyPhotos uses CAOS to

offload methods related to image filters execution and contextual data into CAOS servers. The CAOS server will then enable MyPhotos to recommend *hashtags* for photos with similar context, by leveraging contextual data offloaded by other users.

MyPhotos has two image filters that can be applied to a photo: Simple and Complex. The former applies a RedTone filter. The latter is more compute intensive than Simple because produces a cartoonized version of the image, where the new photo looks like a pencil sketch, as depicted in Figure 6.21. MyPhotos's main screen also shows the time elapsed for the last filter execution.

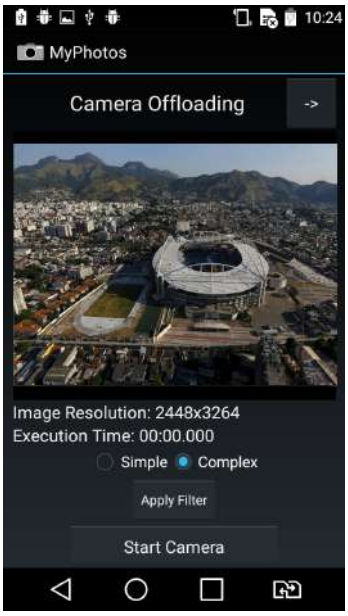


Figure 6.12. MyPhotos main screen

6.3.2.1. Preparing CACs for MyPhotos

The first step in the MyPhotos development process it to implement the CACs that provide the contextual data needed for the application. Two CACs need to be developed: the first CAC provides the location of the mobile device, while the second encapsulates a logical sensor. This second CAC uses the contextual data from the first CAC and provides two values: the hashtags produced by the user and the location/time at which they were produced.

Figure 6.13 illustrates the steps required to create a CAC.

Firstly, we need to configure the development environment in a Linux operating system, by downloading a *shell script* that was developed to create the main structure of a CAC project. This *script* generates a Maven [5] project, which can be edited in any development environment, and has the required settings for a CAC and all methods that need to be implemented.

[5]https://maven.apache.org

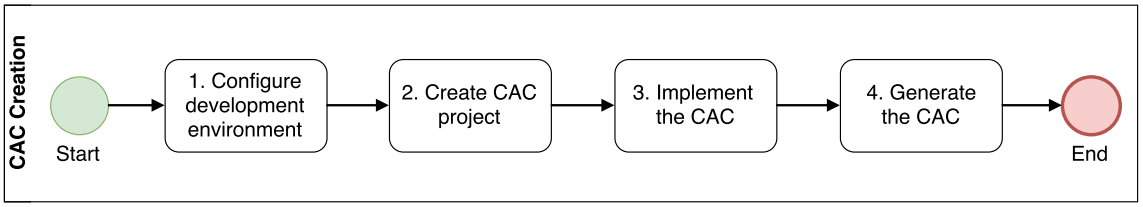


Figure 6.13. Overview of the CAC creation process

The *script* is called `mavenbuild.sh`. Ideally, it should be in the *PATH* setting of the operating system, so that developers can create CACs easily in any folder. The *script* requires two parameters: the CAC name and the project package.

The second step is to run the *script* and create the CAC project. Figure 6.14 shows an example of how to use the *script* to create the first CAC.

Figure 6.14. Example of how to create a CAC project.

Figure 6.15 shows a tree view of the CAC project folder. We can see that there is a file called `pom.xml`, which is where Maven project developers should place the project dependencies and other parameters (e.g., the project version). Two folders, `main` and `test`, have been created, and a package with the name that was passed as a parameter when launching the *script*. In the `main` folder, the CAC main class has the same name as the project, and it is in such file that the CAC must be configured and its required methods must be implemented. In the `test` folder, the main class is used to perform unit tests in the project.

In CAOS, once the CAC project is created, the developer must write the necessary codes (step 3). To configure a CAC, it is necessary to configure some parameters that will be used by CAOS. The main parameter of the CAC is the *ContextKey* (CK), which informs the type of contextual data that it provides. For the first CAC, CK was defined as "context.device.location". This CAC provides contextual data about latitude and longitude of the mobile device's location. For the second CAC, CK was defined as "context.ambient.hashtags".

The hashtags CAC is designed so that whenever the user of the MyPhotos application enters with *hashtags*, they produce contextual data containing the location of the mobile device and *hashtags*. Thus, the *hashtags* CAC performs a subscript of the produced *hashtags*, reads the device's location and provides a new contextual data with both information.

The last step is to generate the CAC, which consists of generating a JAR file containing all class files. This JAR file must be sent to a specific folder on the mobile device so that the CAC begins to provide contextual data according to applications' interest (see

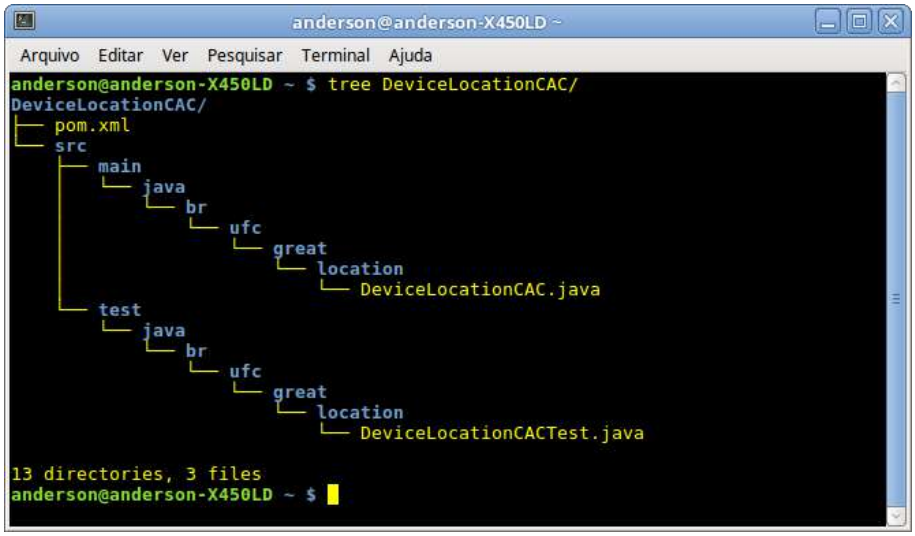


Figure 6.15. Tree view of the CAC project folder.

Figure 6.16).

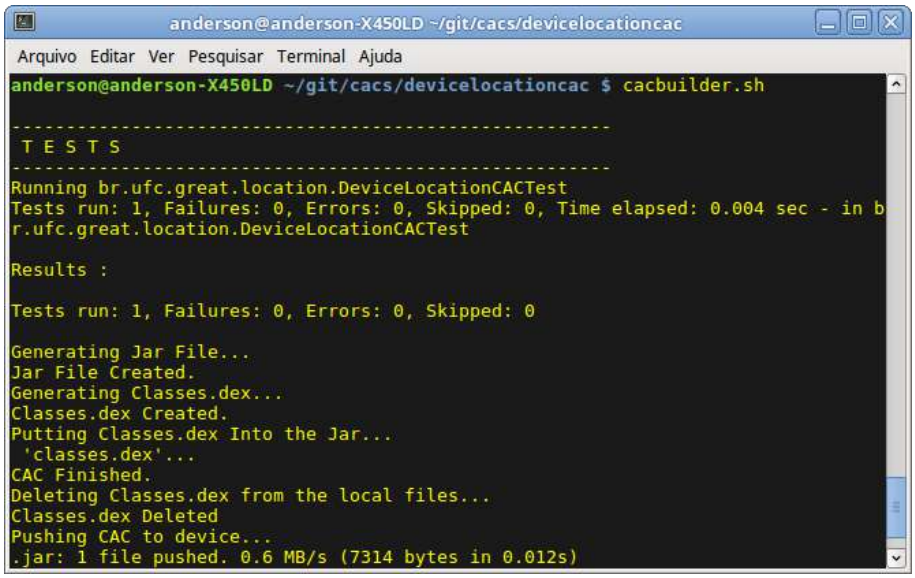


Figure 6.16. Generate the CAC

6.3.2.2. Developing MyPhotos

Once the CACs are built, we can implement MyPhotos. However, due to space constraints, we cannot show all steps and codes required for developing the application. Thus, we will consider that the basic code is already developed[6] and that we still need to configure MyPhotos, so it is able to perform offloading using CAOS. Figure 6.17 illustrates all steps a developer must follow to configure MyPhotos (or any application) to use CAOS.

[6]MyPhotos source code is available at: http://caos.great.ufc.br.

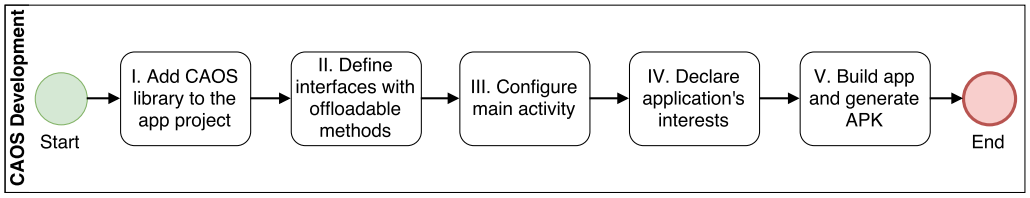


Figure 6.17. Overview of the CAOS development process

The first step (I) is to add the CAOS library to the set of libraries of the Android application project. The library contains CAOS API and TS Context API, which will be used at build time.

After including the CAOS library to the set of libraries of the MyPhotos project, we need to mark the methods that can be offloaded with `@Offloadable` annotation (step II). The `Effect` interface has two methods to apply different image effects to pictures. The methods `simple` and `complex` (respectively, lines 3 and 6 in Listing 6.1) are marked with the offloading markup (respectively, lines 2 and 5 in Listing 6.1), so the Offloading Monitor component will intercept the execution flow of these methods and decide whether the methods must be executed locally or out of the mobile device.

```
1  public interface Effect {
2      @Offloadable
3      public byte[] simple(byte source[]);
4
5      @Offloadable
6      public byte[] complex(byte source[]);
7      ...
8  }
9  public class ImageEffect implements Effect {
10     public byte[] simple(byte source[]){ ... }
11     public byte[] complex(byte source[]){ ... }
12 }
```

Listing 6.1. MyPhotos Interface Effect Markup.

The markup process also has to be performed for the interface responsible for offloading contextual data. The `Read` interface contains the method that retrieves data, and the concrete class `ReadHashtags` implements this interface and all processing related to the data.

```
1  public interface Read {
2      @Offloadable
3      public String readHashTags(double latitude, double longitude,
           double intervalTime, double intervalLocation);
4  }
5
6  public class ReadHashtags implements Read {
7      public String readHashTags(double latitude, double longitude,
           double intervalTime, double intervalLocation){ ... }
8  }
```

Listing 6.2. MyPhotos Interface Read Markup.

In the next step (III), we need to mark the main class of the MyPhotos application (`MyPhotosActivity.java`) with the `@Caos` markup (e.g., Listing 6.3, line 1) and we configure CAOS services to start and stop along with the application. We start

CAOS in the `onCreate` method and stop it in the `onDestroy` method, as illustrated in Listing 6.3, lines 16 and 24 respectively.

It is also necessary to declare the interests of the application regarding contextual data (step IV). Thus, we define MyPhotos interest in Location in Listing 6.3 line 31, while in line 32 we define the interest in Hashtags.

```
1  @Caos
2  public class MyPhotosMainActivity extends Activity implements CaosContextListener {
3
4      @Inject(ImageEffect.class)
5      private Effect effect;
6      @Inject(ReadHashtags.class)
7      private Read read;
8      ...
9      String CK_TAGS = "context.ambient.noise";
10     String CK_LOCATION = "context.device.location";
11     ...
12     @Override
13     protected void onCreate(Bundle bundle) {
14         ...
15         Caos caos = Caos.getConfig().getInstance();
16         caos.start(this, this);
17         ...
18     }
19     ...
20     @Override
21     protected void onDestroy(Bundle bundle) {
22         ...
23         Caos caos = Caos.getConfig().getInstance();
24         caos.stop(this);
25         ...
26     }
27     ...
28     @Override
29     public void onServiceConnected(ISysSUService service) {
30         try {
31             caos.putInterest(CK_LOCATION);
32             caos.putInterest(CK_TAGS);
33         } catch (RemoteException e) {
34             e.printStackTrace();
35         }
36     }
37 }
```

Listing 6.3. Application initialization code for Android.

Finally, when all steps are done, we can build the MyPhotos project and generate the APK file (V). Then, the APK can be installed in the mobile device and also sent to the CAOS server, in order to support offloading of MyPhotos methods (such step will be explained in the next section).

6.3.2.3. Configuring CAOS Services

In this section, we present how to configure CAOS Services on mobile and cloud sides.

On mobile side

In order to run CAOS on the mobile side, we need to the install the application CAOS. This application acts as an service, in which contextual data synchronization policies are configured as well as the privacy settings of such data. Figure 6.18 shows screen-

shots of the CAOS application.

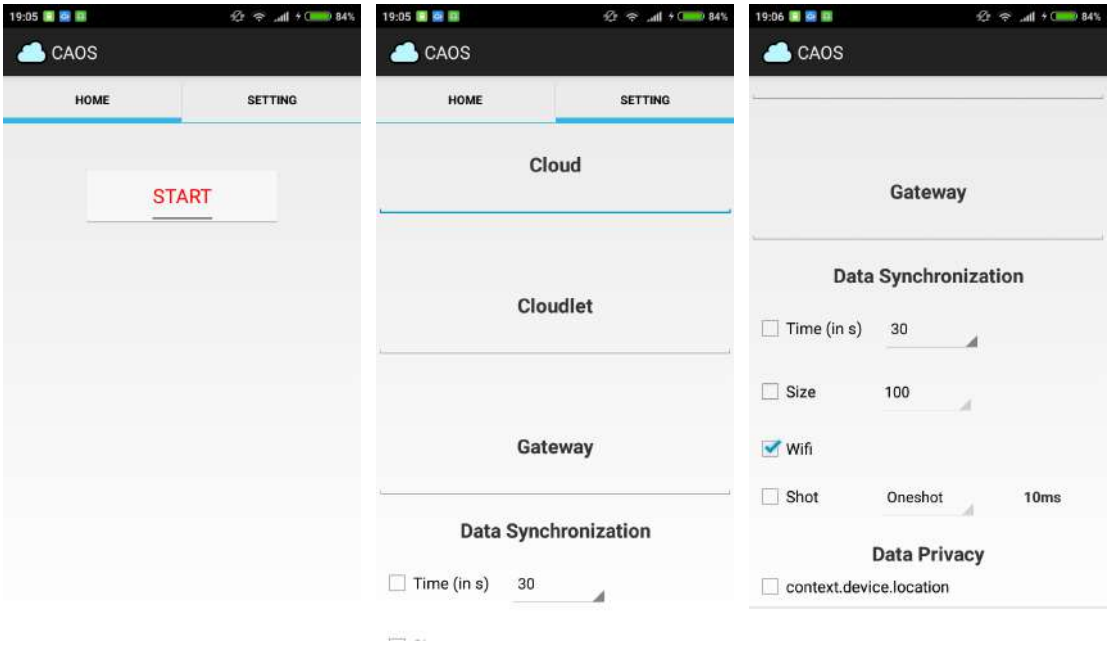


Figure 6.18. Screenshots of the CAOS application running on mobile device

The JAR file of each CAC must be sent to a specific mobile device folder, so the CAOS application can manage the CACs and control their use. In sum, all CACs stored in the `/storage/emulated/0/Android/data/br.ufc.great.caos/files/components` folder of the mobile device are loaded and their privacy setting are managed by CAOS. The user can then use the application to configure the privacy level of each CAC.

On cloud side

In order to run CAOS Services, we need to the install Android SDK, MongoDB, TomCat server, PostgreSQL server, and VirtualBox in a machine running any operating system. After installing the Android SDK, we need to include the ADB (Android Debug Bridge) tool into the PATH variable of the system. Thus, the CAOS controller can manage the pool of Android x86 virtual machines using ADB.

After installing the mentioned applications, the next step is to install the Android x86 operating system in a virtual machine with at least 512 Mb of RAM and 1 GB hard disk. Android x86 is a project that aims to port the Android open source project to the x86 platform. In their website, we can find ISO images for different Android versions, but we recommend to use the 4.4.3 version. Once the Android x86 is installed in a virtual machine, we need to install the ApkLoader application on it, in order to run the Offloading Method Invocation Service.

We need to configure a few properties before executing the CAOS controller. First, the database credentials have to be included in the `persistence.xml` file (as illustrated in Listing 6.4). We need to set IP, database name, username and password in the file according to the PostgreSQL configuration.

```
<persistence ... version="2.0">
  <persistence-unit name="caos" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <properties>
        <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://<IP>/<DB>"/>
        <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver"/>
        <property name="javax.persistence.jdbc.user" value="<USER>"/>
        <property name="javax.persistence.jdbc.password" value="<PASS>"/>
        ...
      </properties>
  </persistence-unit>
</persistence>
```

Listing 6.4. Example of the persistence.xml file.

After that, we need to configure the `vms` file, which is located in the properties folder. In this file, we have to include the endpoint of the ADB server running on the Android x86 virtual machines. As by default ADB listens to the port 5555, the vms file should look like the Listing 6.5. In the example, we are setting two Android x86 virtual machines with IPs 192.168.56.110 and 192.168.56.120 to our VMs pool.

```
192.168.56.110:5555
192.168.56.120:5555
```

Listing 6.5. Example of the vms file.

The last file we need to edit is the `net.properties`, which is also located in the properties folder. In this file, we have to set the network interface of the machine that will listen for discovery messages, as well as the ports that will be used by several services. We recommend to use all default ports and change only the network interface name. The Listing 6.6 presents an example of the `net.properties` file. In the example, we use the wireless interface (`wlan0`) to listen for mobile devices discovery requests.

```
prop.server.networkInterfaceName=wlan0
prop.server.port.authentication=8300
...
prop.server.port.discovery.reply=31002
```

Listing 6.6. Example of the net.properties file.

Before launching the CAOS Controller, we need to put in the `apks` folder a copy of all applications (APK files) that we want to support offloading. Figure 6.19 shows the `apks` folder with several APK files.

When running, the CAOS Controller will distribute the files to the Android x86 virtual machines that will run the offloading requests. Figure 6.20 shows how to run the CAOS controller.

6.3.2.4. Running MyPhotos

MyPhotos allows users to apply filters to their photos, use *hashtags*, share and tag them with contextual information. Users can use two image filters to apply effects to photos: Simple and Complex. The Figure 6.21(a) shows the cartoonized version of the Maracanã Stadium image after applying the filter Complex.

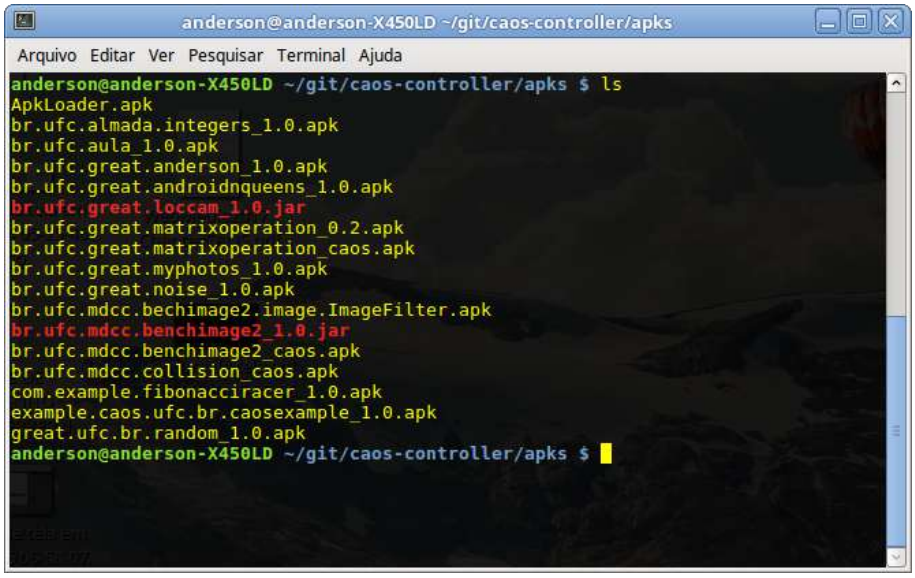


Figure 6.19. APKs Folder

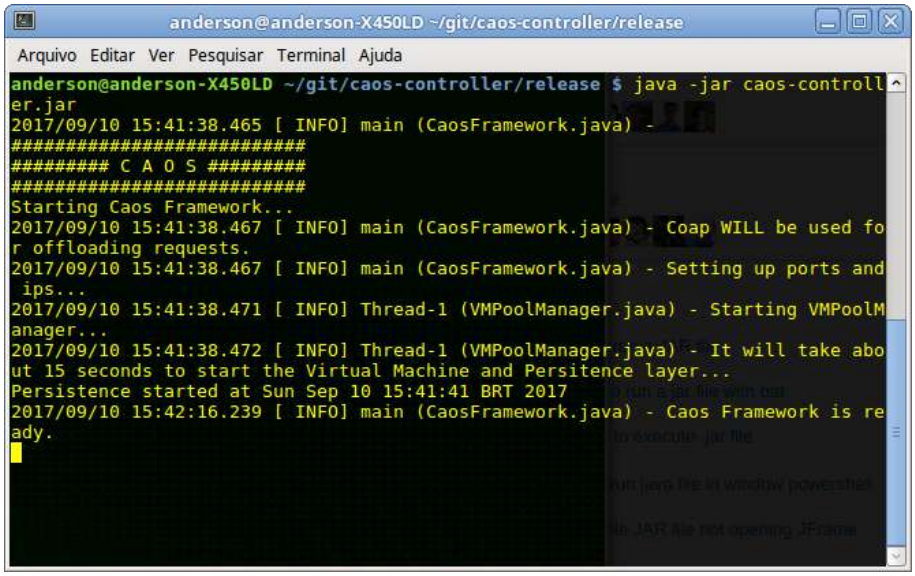
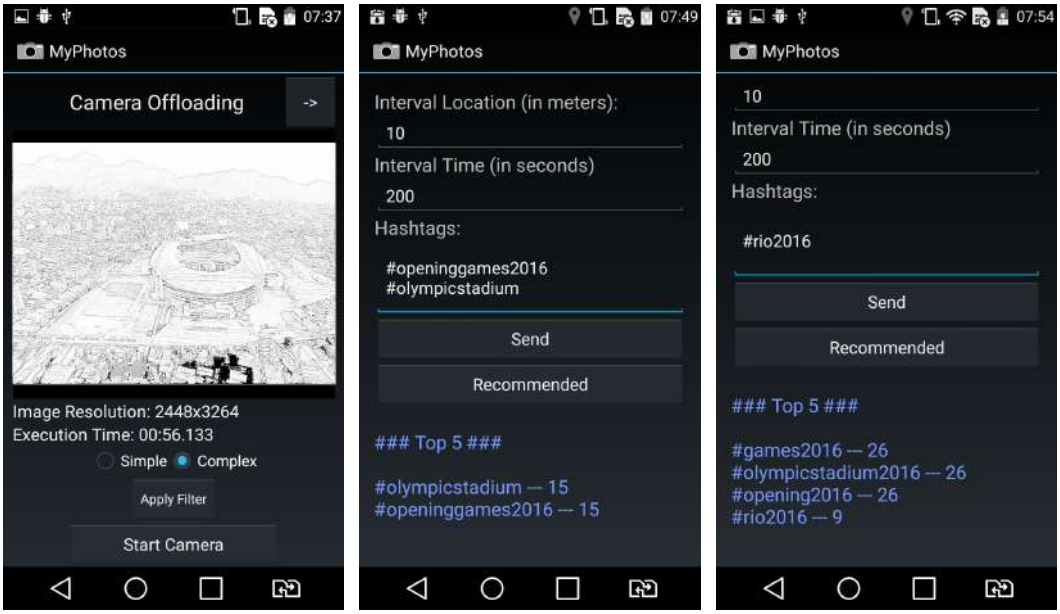


Figure 6.20. Run the CAOS Controller

In Figure 6.21(b), the settings available for the use of *hashtags* are presented. Two parameters are required: the location range (in meters) and the time interval (in seconds). Thus, the user defines how far and how late these data are according to his/her geographical position and time. The "Recommended" button retrieves *hashtags* used previously on images with the same context of the current photo. The "Send" button publishes the image.

First, MyPhotos was executed with the device disconnected from the network, which forces the image filter and tags recommendation to run locally. Figure 6.21(a) shows that it takes more than 56 seconds to apply the filter Complex.



(a) Applying the filter Complex (b) Recommended *hashtags* when running locally (c) Recommended *hashtags* when connected to the CAOS server

Figure 6.21. MyPhotos Screenshots

The Figure 6.21(b) shows that when the mobile device is not connected to the CAOS server, it only retrieves *hashtags* stored in the local Tuple-Space Based Context Bus. Since the user labeled the photo with two *hashtags*, #olympicstadium and #openinggames2016, these two tags are also the only *hashtags* that could be recommended to the user. In Figure 6.21(c), the device is connected to the CAOS server, and when the *hashtags* recommendation is requested, the application retrieves up to the five most popular *hashtags* with a context similar to the current user.

6.4. Trends and Research Challenges

Although the combination of context-aware Computing and MCC has great potential for future mobile applications, it is also important to understand that there are issues that raise new challenges to enable such integration. Some of these issues are mentioned in this section.

6.4.1. Costs for raw data transfer to compute the context inference on cloud services

For some scenarios, the amount of data required to process a contextual situation may be a barrier for the use of MCC. Sometimes, multiple sources of information are needed to derive a Context information, i.e., a multi-sensory process. This is a common scenario for data-intensive applications, in which some cases of context sensing may fit. [Naqvi et al. 2013] cites that step counting or falling detection algorithms from sensor samples such as accelerometer and gyroscope may require a reasonable amount of data to produce good results. However, transferring a huge amount of data very frequently may bring side effects related to energy consumption or charging fees while using mobile operator network.

6.4.2. High Latency between mobile devices and cloud resources

According to [Naqvi et al. 2013], certain applications such as live gaming, augmented reality do not perform efficiently on most mobile devices nowadays. Although the use of cloud resources can improve the processing of these applications, they are also sensitive to latency in communication between mobile devices and the cloud. Some context-aware mobile applications may suffer from the delay in processing contextual information, make it even unpractical. As pointed out in [Khan et al. 2012], "by the time the data makes its way to the cloud system for analysis, the opportunity to act on it might be gone". Studies conducted by our research group show how current mobile infrastructures in Brazil do not guarantee adequate QoS to transfer processing between mobile devices and the cloud [Costa et al. 2014]. This is mainly due to the distance between mobile devices and the centralized resources in cloud provider data centers. In these situations, cloud-based approaches or fog computing appear as strong candidates for meeting low latency requirements, since they typically involve one-hop communication. However, the large-scale dissemination of cloudlets or similar infrastructures is still in its infancy. There is also the fact that there are no formed business models that indicate how to encourage the popularization of these devices.

6.4.3. Security and Privacy

Cloud computing is often criticized for its centralized model of information storage on third-party machines. When thinking about the integration of context-aware computing with mobile cloud computing, this problem also rises, since the inference of contextual situations typically makes use of sensitive information from users (such as their location), which for many people, it can be seen as a breach of privacy. Traditional techniques such as information cryptography or data anonymization may be employed, but they also impose additional processing, and consequently delays and additional power consumption.

6.4.4. Power Consumption

Extending the device autonomy time is one of the biggest (if not the biggest) concerns for mobile application applications. While there has been a significant increase in the processing capacity of mobile devices each year, the same advancement is not seen in relation to the device's batteries. As a result, there are still barriers to processing or even intense data exchange, since radio transmission is one of the biggest villains for the power consumption of a mobile device. With this, the decision to process the contextual situation locally or remotely becomes even more difficult, making it a trade off between performance and energy savings. It is also possible that according to the decision, the inference of the contextual situation can be carried out in different ways, according to the context of the device. If the current situation allows to offload the inference of context to the cloud, more complex processing with more accurate results may be performed, based on larger datasets. If it is not possible, a simpler treatment can be offered, but decreasing the quality of the result of the inference. Finally, some scenarios can lead to situations where context management tasks can be divided between mobile device and cloud. This decision balances a choice between improving the accuracy of contextual inference or preserving device features.

6.4.5. Large-scale availability of MCC infrastructures

Despite the benefits that infrastructures such as cloudlets offer, having such services spread across multiple locations globally requires considerable investment, which is prohibitive for application providers. There is currently no commercial deployments that use MCC technology [Balan and Flinn 2017], probably due to the lack of appropriate business models. Even if we think of specialized companies that support MCC, or even users who want to offer resources spontaneously, aspects such as security and privacy can be obstacles to popularizing these initiatives.

6.4.6. Missing Killer applications

Experiments reports with MCC include topics such as image processing, recommendation systems, face recognition and language translation. However, according to [Balan and Flinn 2017], the popularization of the use of MCC faces two main problems. First, more advanced deep learning techniques typically used in these applications require large data sets, which impacts on the overall response time on these applications. Second, advances in the processing capability of mobile devices suppress the need of offloading tasks out of the mobile device. While this statement is true, our particular view is that only a category of mobile devices (high-end devices) meets processing capabilities for these applications. [Balan and Flinn 2017] claims that real-time video processing applications have a good potential to become a killer application in MCC field, since processing video streams does not require a huge amount of data, but demands an intensive processing to obtain data from a scene (such as subtitles on certain locations).

6.5. Conclusion

Contextual-Aware features are increasingly present in current mobile applications. Even with advances made over the past few years in mobile devices' capabilities, scenarios envisioned for context-aware mobile applications indicate the use of large datasets and machine learning techniques that will require resources not found (at least on most) in current mobile devices. This mini-course focuses on this problem, by presenting how Mobile Cloud Computing (MCC), especially offloading techniques, may improve the support for future context-aware mobile applications.

The use of "context" is already widely used in MCC integration. However, previous research works found on a literature review use this term as the situation for decision making if data or tasks offloading between mobile devices and cloud infrastructures are worthy or not, regarding performance or energy savings. In this mini-course, our approach focus on how context management services can be improved with MCC concepts. This view is presented in a practical way through the framework CAOS, an Android-based platform that supports both data and computing offloading. A case study illustrating how CAOS can be used to implement a context-aware multimedia application is described. To the best of our knowledge, this is one of the few initiatives where context management integration with cloud services is supported.

As future work, we plan to keep evolving our framework. We currently study a complete refactoring of CAOS architecture, where its services will be designed using MicroServices approach. We aim at improving CAOS' scalability and flexibility by mod-

eling its services as independent units.

References

Aguilar, J., Jerez, M., Exposito, E., and Villemur, T. (2015). Carmicloc: Context awareness middleware in cloud computing. In *2015 Latin American Computing Conference (CLEI)*, pages 1–10.

Balan, R., Flinn, J., Satyanarayanan, M., Sinnamohideen, S., and Yang, H.-I. (2002). The case for cyber foraging. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, EW 10, pages 87–92, New York, NY, USA. ACM.

Balan, R. K. and Flinn, J. (2017). Cyber foraging: Fifteen years later. *IEEE Pervasive Computing*, 16(3):24–30.

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

Barbera, M. V., Kosta, S., Mei, A., and Stefa, J. (2013). To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In *Proc. of IEEE INFOCOM*, volume 2013.

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., and Riboni, D. (2010). A survey of context modelling and reasoning techniques. *Pervasive Mob. Comput.*, 6(2):161–180.

Buthpitiya, S., Luqman, F., Griss, M., Xing, B., and Dey, A. (2012). Hermes – a context-aware application development framework and toolkit for the mobile environment. In *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, pages 663–670.

Carlson, D. and Schrader, A. (2012). Dynamix: An open plug-and-play context framework for android. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 151–158.

Chatzopoulos, D., Bermejo, C., Huang, Z., and Hui, P. (2017). Mobile augmented reality survey: From where we are to where we go. *IEEE Access*, 5:6917–6950.

Chihani, B., Bertin, E., and Crespi, N. (2013). Decoupling context management and application logic: A new framework. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a*, pages 1–6.

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., and Patti, A. (2011). Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 301–314, New York, NY, USA. ACM.

Costa, P. B., Rego, P. A. L., Coutinho, E. F., Trinta, F. A. M., and d. Souza, J. N. (2014). An analysis of the impact of the quality of mobile networks on the use of cloudlets. In *2014 Brazilian Symposium on Computer Networks and Distributed Systems*, pages 113–121.

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

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. (2010). Maui: Making smartphones last longer with code offload. In *MobiSys 2010, Proceedings ACM*, pages 49–62, New York, NY, USA. ACM.

Curiel, P. and Lago, A. (2012). Context management infrastructure for intelligent-mobile-services execution environments. In *Information Systems and Technologies (CISTI), 2012 7th Iberian Conference on*, pages 1–6.

Da, K., Dalmau, M., and Roose, P. (2011). A survey of adaptation systems. *International Journal on Internet and Distributed Computing Systems*, 2(1):1–18.

Da, K., Dalmau, M., and Roose, P. (2014a). Kalimucho: Middleware for mobile applications. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 413–419, New York, NY, USA. ACM.

Da, K., Roose, P., Dalmau, M., Nevado, J., and Karchoud, R. (2014b). Kali2much: A context middleware for autonomic adaptation-driven platform. In *Proceedings of the 1st ACM Workshop on Middleware for Context-Aware Applications in the IoT*, M4IOT '14, pages 25–30, New York, NY, USA. ACM.

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

Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2011). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*.

Dogdu, E. and Soyer, O. (2013). Morecon: A mobile restful context-aware middleware. In *Proceedings of the 51st ACM Southeast Conference*, ACMSE '13, pages 37:1–37:6, New York, NY, USA. ACM.

Doukas, C. and Antonelli, F. (2013). Compose: Building smart amp; context-aware mobile applications utilizing iot technologies. In *Global Information Infrastructure Symposium, 2013*, pages 1–6.

Duarte, P. A., Silva, L. F. M., Gomes, F. A., Viana, W., and Trinta, F. M. (2015). Dynamic deployment for context-aware multimedia environments. In *Proceedings of the 21st Brazilian Symposium on Multimedia and the Web*, WebMedia '15, pages 197–204, New York, NY, USA. ACM.

El-Derini, M., Aly, H., El-Barbary, A.-H., and El-Sayed, L. (2014). Droidcloudlet: Towards cloudlet-based computing using mobile devices. In *Information and Communication Systems (ICICS), 2014 5th International Conference on*, pages 1–6.

Fernando, N., Loke, S. W., and Rahayu, W. (2013). Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84 – 106.

Ferrari, A., Giordano, S., and Puccinelli, D. (2016). Reducing your local footprint with anyrun computing. *Computer Communications*, 81:1 – 11.

Ferroni, M., Damiani, A., Nacci, A. A., Sciuto, D., and Santambrogio, M. D. (2014). coda: An open-source framework to easily design context-aware android apps. In *Proceedings of the 2014 12th IEEE International Conference on Embedded and Ubiquitous Computing*, EUC'14, pages 33–38, Washington, DC, USA. IEEE Computer Society.

Gomes, F. A., Viana, W., Rocha, L. S., and Trinta, F. (2016). A contextual data offloading service with privacy support. In *Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web*, pages 23–30. ACM.

Gomes, F. A. A., Rego, P. A. L., Rocha, L., de Souza, J. N., and Trinta, F. (2017). Caos: A context acquisition and offloading system. *COMPSAC*.

Gordon, M. S., Jamshidi, D. A., Mahlke, S., Mao, Z. M., and Chen, X. (2012). Comet: Code offload by migrating execution transparently. In *USENIX 2012, Proceedings ACM*, pages 93–106, Berkeley, CA, USA. USENIX Association.

Herrmann, K. (2010). Self-organized service placement in ambient intelligence environments. *ACM Trans. Auton. Adapt. Syst.*, 5(2):6:1–6:39.

Kakadia, D., Saripalli, P., and Varma, V. (2013). Mecca: Mobile, efficient cloud computing workload adoption framework using scheduler customization and workload migration decisions. In *Proceedings of the First International Workshop on Mobile Cloud Computing & Networking*, MobileCloud '13, pages 41–46, New York, NY, USA. ACM.

Khan, A., Othman, M., Madani, S., and Khan, S. (2014). A survey of mobile cloud computing application models. *Communications Surveys Tutorials, IEEE*, 16(1):393–413.

Khan, R., Khan, S. U., Zaheer, R., and Khan, S. (2012). Future internet: The internet of things architecture, possible applications and key challenges. In *2012 10th International Conference on Frontiers of Information Technology*, pages 257–260.

Kharbanda, H., Krishnan, M., and Campbell, R. (2012). Synergy: A middleware for energy conservation in mobile devices. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pages 54–62.

Kosta, S., Aucinas, A., Hui, P., Mortier, R., and Zhang, X. (2012). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953.

Kovachev, D., Cao, Y., and Klamma, R. (2011). Mobile cloud computing: A comparison of application models. *CoRR*, abs/1107.4940.

Kovachev, D., Yu, T., and Klamma, R. (2012). Adaptive computation offloading from mobile devices into the cloud. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 784–791.

Kristensen, M. D. and Bouvin, N. O. (2010). Scheduling and development support in the scavenger cyber foraging system. *Pervasive Mob. Comput.*, 6(6):677–692.

Kumar, K., Liu, J., Lu, Y.-H., and Bhargava, B. (2013). A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140.

Kumar, K. and Lu, Y.-H. (2010). Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56.

Lee, Y., Iyengar, S. S., Min, C., Ju, Y., Kang, S., Park, T., Lee, J., Rhee, Y., and Song, J. (2012). Mobicon: A mobile context-monitoring platform. *Commun. ACM*, 55(3):54–65.

Lima, F. F. P., Rocha, L. S., Maia, P. H. M., and Andrade, R. M. C. (2011). A decoupled and interoperable architecture for coordination in ubiquitous systems. In *Proceedings of the 2011 Fifth Brazilian Symposium on Software Components, Architectures and Reuse*, SBCARS'11, pages 31–40, Washington, DC, USA. IEEE Computer Society.

Liu, J., Ahmed, E., Shiraz, M., Gani, A., Buyya, R., and Qureshi, A. (2015). Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions. *Journal of Network and Computer Applications*, 48:99–117.

Liu, J., Shen, H., and Zhang, X. (2016). A survey of mobile crowdsensing techniques: A critical component for the internet of things. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6.

Ma, R. K. K., Lam, K. T., and Wang, C.-L. (2011). excloud: Transparent runtime support for scaling mobile applications in cloud. In *Proceedings of the 2011 International Conference on Cloud and Service Computing*, CSC '11, pages 103–110, Washington, DC, USA. IEEE Computer Society.

Magurawalage, C. M. S., Yang, K., Hu, L., and Zhang, J. (2014). Energy-efficient and network-aware offloading algorithm for mobile cloud computing. *Computer Networks*, 74, Part B:22 – 33. Special Issue on Mobile Computing for Content/Service-Oriented Networking Architecture.

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

Mane, Y. V. and Surve, A. R. (2016). Capm: Context aware provisioning middleware for human activity recognition. In *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, pages 661–665.

March, V., Gu, Y., Leonardi, E., Goh, G., Kirchberg, M., and Lee, B. S. (2011). μcloud: Towards a new paradigm of rich mobile applications. *Procedia Computer Science*, 5(0):618 – 624. The 2nd International Conference on Ambient Systems, Networks and Technologies (ANT-2011) / The 8th International Conference on Mobile Web Information Systems (MobiWIS 2011).

Mitchell, M., Meyers, C., Wang, A.-I., and Tyson, G. (2011). Contextprovider: Context awareness for medical monitoring applications. In *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE*, pages 5244–5247.

Naqvi, N. Z., Preuveneers, D., and Berbers, Y. (2013). *Cloud Computing: A Mobile Context-Awareness Perspective*, pages 155–175. Springer London, London.

OSullivan, M. J. and Grigoras, D. (2016). Context aware mobile cloud services: A user experience oriented middleware for mobile cloud computing. In *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 67–72.

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

Pulli, K., Baksheev, A., Kornyakov, K., and Eruhimov, V. (2012). Real-time computer vision with opencv. *Commun. ACM*, 55(6):61–69.

Punjabi, J., Parkhi, S., Taneja, G., and Giri, N. (2013). Relaxed context-aware machine learning midddleware (rcamm) for android. In *Intelligent Computational Systems (RAICS), 2013 IEEE Recent Advances in*, pages 92–97.

Rego, P. A. L., Cheong, E., Coutinho, E. F., Trinta, F. A., Hasan, M. Z., and de Souza, J. N. (2017). Decision tree-based approaches for handling offloading decisions and performing adaptive monitoring in MCC systems. In *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*.

Rego, P. A. L., Costa, P. B., Coutinho, E. F., Rocha, L. S., Trinta, F. A., and de Souza, J. N. (2016). Performing computation offloading on multiple platforms. *Computer Communications*.

Sanaei, Z., Abolfazli, S., Gani, A., and Buyya, R. (2014). Heterogeneity in mobile cloud computing: Taxonomy and open challenges. *Communications Surveys Tutorials, IEEE*, 16(1):369–392.

Satyanarayanan, M. (2001). Pervasive computing: vision and challenges. *Personal Communications, IEEE*, 8(4):10–17.

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

Sharifi, M., Kafaie, S., and Kashefi, O. (2012). A Survey and Taxonomy of Cyber Foraging of Mobile Devices. *IEEE Communications Surveys & Tutorials*, 14(4):1232–1243.

Verbelen, T., Simoens, P., De Turck, F., and Dhoedt, B. (2012). Aiolos: Middleware for improving mobile application performance through cyber foraging. *J. Syst. Softw.*, 85(11):2629–2639.

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

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, 265(3):94–104.

Williams, E. and Gray, J. (2014). Contextion: A framework for developing context-aware mobile applications. In *Proceedings of the 2Nd International Workshop on Mobile Development Lifecycle*, MobileDeLi '14, pages 27–31, New York, NY, USA. ACM.

Xia, Q., Liang, W., and Xu, W. (2013). Throughput maximization for online request admissions in mobile cloudlets. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 589–596.

Xiao, Y., Simoens, P., Pillai, P., Ha, K., and Satyanarayanan, M. (2013). Lowering the barriers to large-scale mobile crowdsensing. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, HotMobile '13, pages 9:1–9:6, New York, NY, USA. ACM.

Yurur, O., Liu, C. H., Sheng, Z., Leung, V. C. M., Moreno, W., and Leung, K. K. (2016). Context-awareness for mobile sensing: A survey and future directions. *IEEE Communications Surveys Tutorials*, 18(1):68–93.

Zhang, Y., Huang, G., Liu, X., Zhang, W., Mei, H., and Yang, S. (2012). Refactoring android java code for on-demand computation offloading. *SIGPLAN Not.*, 47(10):233–248.

Authors' Biographies

Fernando A. M. Trinta holds a PhD degree from Federal University of Pernambuco (2007). He is an adjunct professor at the Computer Science Department - Federal University of Ceará since 2011. He works mainly in the areas of Multimedia, Software Engineering and Distributed Systems. Currently, his research is focused on Mobile Cloud Computing, Context-Aware Computing and Fog Computing.

Paulo A. L. Rego holds a PhD in Computer Science from the Federal University of Ceará (UFC) since 2016. He is currently an adjunct professor at UFC - Campus Quixadá. He works in the area of Computer Science, with emphasis on Computer Networks and Distributed Systems. His main research interests include Mobile and Cloud Computing, Edge and Fog Computing.

Francisco A. A. Gomes holds a Master degree from the Federal University of Ceará. He is currently a PhD student. He works in the area of Software Engineering applied to Ubiquitous Computing. His main research interests are: Mobile Computing and MCC.

Lincoln S. Rocha holds a PhD in Computer Science from Federal University of Ceará (2013). He is currently a professor at the same university and has experience in the area of Software Engineering. His man research interests include Mobile and Ubiquitous Computing, Context-Aware Computing and Internet of Things.

José N. de Souza holds a PhD degree at Pierre and Marie Curie University (PARIS VI/MASI Laboratory), France, since 1994. He is currently working as a researcher full professor at the Federal University of Ceará in the Computer Science Department, and is IEEE senior member. Since 1999, he has been the Brazilian representative at the IFIP TC6. His main research interests are Cloud Computing and Network Management.