cODA: An Open-Source Framework to Easily Design Context-Aware Android Apps

M. Ferroni, A. Damiani, A. A. Nacci, D. Sciuto, M. D. Santambrogio
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)
Politecnico di Milano, Milano, Italy
{matteo.ferroni, alessandro.nacci, marco.santambrogio, donatella.sciuto}@polimi.it
{andrea3.damiani}@mail.polimi.it

***Abstract*—Mobile devices take an important part in everyday life. They are now cheaper and widespread, but still a lot of time is spent by the users to configure them: users adapt to their own device, not vice versa. Can our smartphones do something smarter? In this work, we propose a framework to support the development of context aware applications for Android devices: the goal of such applications is to reduce as much as possible the interaction with the user, making use of automatic and intelligent components. Moreover, these components should consume as less power and computational resources as possible, being them part of a mobile ecosystem whose battery and hardware are highly constrained. The work implies the study of a methodology that fits the Android framework and the design of a highly extensible software architecture. An open source framework based on the proposed methodology is then described. Some use cases are finally presented, analyzing the performances and the limitations of the proposed methodology.**

I. Introduction

In the last few years, mobile devices completely changed our lives, our social interactions and our culture: they became the most used electronic devices, giving us the possibility to be always fully connected with other people. In fact, a generic mobile device (like a notebook, a smartphone, a tablet, a smart-watch, or a smart-glass) has no reason to exist without its connectivity capabilities: said in other words, mobile devices are meant to let us reach the information we need while we are in mobility.

Moreover, the evolution of these devices highly affected the way in which we communicate and interact. At the beginning, we were used to intend the communication as *on-request and synchronous*: mobile devices were used to perform phone calls from a caller to a receiver, as soon as the caller had the need to start this interaction. At this stage, mobile devices were just mobile phones, without any other remarkable capability. A famous example is the Motorola StarTAC, the first clamshell mobile device based on the TAC protocol [1].

Later, the introduction of the *Short Message Service* (SMS) led to a *on-request and asynchronous* communication paradigm: the caller did not expect to receive an immediate answer to his/her interaction. Therefore, mobile devices gained a more powerful screen and some limited computation capabilities. The Nokia 3210 and 3310 are probably the most important pieces of this generation of mobile devices [2].

Then came the World Wide Web and the possibility for everyone to be the author of Internet contents: suddenly, mobile devices were pushed to become the main technology to access the Internet, thanks to their portability and widespread diffusion. Consequently, new communication paradigms like Instant Messaging (IM) and social networking were introduced along with phone calls and SMS, making mobile devices the biggest and most important information hub for every customer. Mobile devices had then a larger and powerful screen and processor. Common examples are today's Apple iPhone, Samsung Galaxy and Google Nexus.

At this point, mobile phones started replacing laptops and desktops in a variety of functionalities, since they were cheaper and simpler to use: they became indispensable in our everyday life, simplifying many daily operations. Moreover, they moved from an *on-request* communication model to an *always-connected* one: Internet information publishing and retrieving became possible anytime and anywhere.

Up to now, the communication was only between the mobile device and the Internet. In the last few years, a new trend became clear: people are interested in making their own mobile device a data collecting hub with respect to their environment (Figure 1). As a consequence, mobile devices started integrating more and more sensors (accelerometers, gyros, etc.) or connecting with external ones (healthcare sensors, sport sensors, etc.). This led to the definition of the so called *context aware mobile computing* [3]. Context aware mobile computing aims at using the information about the user behavior and environment that is possible to retrieve through mobile devices. Following the definition expressed in [3], context aware systems can *collect 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*. In other words, *context-aware computing refers to a general class of mobile systems that can sense their physical environment, and adapt their behavior accordingly* [4] [5]. Generally, the final goal of such context aware computing systems is to create devices able to anticipate the users needs, in order to simplify their life.

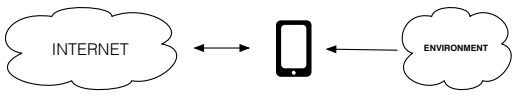


Fig. 1. Internet and enviroment connected mobile device

Following this trend, many examples of context aware applications for mobile devices have been developed: as shown

in [6], for instance, applications to assist users while shopping, traveling or working have been proposed and implemented. Despite these examples, there is still a lack of shared and well defined design methodologies to create such applications in the constrained environment of mobile devices. As a consequence, the implementation of such systems is commonly performed from scratch, with a high development cost [7]. This is the reason why we propose here a methodology to enable a rapid design and development of context aware applications.

II. Contribution

The idea of creating a methodology that is shared among different use cases came out from the observation that all these applications have something in common. In order to understand this concept, let us consider two examples: an application that starts tracking the user performance while making a sport (e.g., jogging) and another that automatically suggests the user with the best time to wake up with respect to the appointments on his/her schedule.

In the first case, we need to understand that the user is actually jogging, in order to start tracking the user performance. For such a purpose, we need to **collect** data from motion sensors (e.g., the accelerometer); these data must be then **elaborated** in other to find a pattern which is typical for the current user behavior. Finally, when a jogging pattern is recognized, the **action** of measuring the user performance starts. In the second example, we need to **collect** data about the user routine in order to identify his home and work locations; as soon as the user will set the alarm clock for the following morning, the system has to **elaborate** the time it takes to reach the place of the first appointment of the day he/she has in his/her schedule. Finally, the system has to perform an **action** (i.e., warn the user) if the alarm time does not guarantee him/her to reach the place of the appointment on time.

Some common aspects have been highlighted in both the examples: a collecting phase, an elaboration (decision) phase and an action phase are typical in context aware applications. Generalizing the examples, we inferred that the idea of context aware computing is strictly related with the so called Observe, Decide and Act (ODA) loop [8]. In fact, everytime we want to implement such a system, we need an **observation phase** on the environment, a **decision phase** to analyze data gathered in order to choose which are the actions to be performed, and finally an **action phase** to actually perform those actions.

Within this paper, we want to answer a simple question: how can context-aware concepts fit the current available mobile devices technologies? We then propose a design methodology, implemented in the lightweight *context Observe Decide Act* (cODA) framework for Android devices[1]. We decided to work on Android since it is the most widespread operating system for mobile devices (79% of mobile devices market [9]). An explanation of how the cODA methodology is implemented in the Android SDK will be given: differently from all the state of the art approaches, we present a ready-to-use open source framework, meant for Android developers without requiring any particular modification to the mobile device operating system.

[1]The cODA framework is open source and can be downloaded from https://github.com/mattferroni/cODA

The paper is organized as follows: in Section III a state of the art analysis of similar works is presented; then in Section IV the implementation details are explained, while in Section V we propose two case studies in order to show the potential of the cODA framework. Finally some concluding remarks are presented in Section VI.

III. Related works

Many research groups explored the opportunities of context-aware applications on mobile devices, since the first vision of ubiquitous computing [10]. One of the foundational work was an architecture proposed by Bill N. Schilit to support context-aware computing [11], in which the user's location, nearby people, the presence of smart devices or other kinds of objects could trigger applications adaptation over time.

That implementation made use of a *blackboard pattern* to share context information; all the components of the infrastructure can read or write new data (i.e., sensor-based or inferred context information) on a shared "board", thus creating a loosely coupled extensible component interface. The project was implemented on the Xerox PARCTab, whose hardware was proprietary: as a consequence, this solution is not available on other devices. The same blackboard approach has been used in subsequent works [12] [13], while other works implemented a *widget based* approach [14] [15]. However, all those solutions had the strong limitation of running on proprietary hardware or they affect solutions that are no more supported (e.g., Symbian OS [16]).]

Other remarkable implementations have brought a context-aware approach inside the Operating System, with the main focus of making the OS energy aware. Some notable examples of such operating systems are CondOS [17], ErdOS [18], FALCON [19], ContextOS [20]. Since customizing an operating system or installing a custom OS is an operation affordable only by power users, these solutions are not widespread and cannot be adopted by common people.

We introduced the context awareness at the OS level in a previous work, [8]. Anyway, nowadays, everyone has a mobile device with a pre-loaded OS and generally applications can be easily downloaded form a marketplace. As a consequence, a developer who wants to develop context aware applications needs an applicative framework that lets him/her to develop such application since the current available operating system does not support such capability. Being constrained by the OS capability, we need to bring the context awareness into the applications themselves to deliver the context awareness to everyone. As a consequence, with this paper, we want to propose a methodology and a framework able to work on the largest amount of currently available mobile devices.

The information about the context are obviously made available by the operating system but this information is too raw to be directly used to build the context status. In order to extract the needed degree of information, some computation and aggregation need to be performed to create context-level data. Many frameworks have been proposed in the past to accomplish this task. For instance, the Java Context-Awareness Framework (JCAF) is proposed in [21], SOLAR is proposed in [22] and Context Toolkit is proposed in [14]. While the first one is a Java-based context-awareness infrastructure and

programming APIs for creating context-aware computer applications, the second one is an open platform which employs a graph-based abstraction for context aggregation and dissemination, the third introduced a structured design approach that helps with identifying context abstractions and leads to a software design. Anyway, all of them are generic and not bound to the Android OS components, so, as a consequence, they do not have to deal with such a constrained software and operating system (in the case of mobile devices, for instance, the energy consumption is a major deal).

In fact, the need of sensing the environment to understand the context cannot disregards the power consumption, since we have to sense while consuming the less energy possible and saving only relevant data [23] [24]; more information on the topic can be found in a survey [25].

IV. Methods And Framework Design

As stated in Section I, the cODA framework implements and revises the concept of ODA Loop. An ODA Loop comprises three steps: the Observe phase, dedicated to raw data acquisition about the environment and the system; the Decide phase, that leverages the data retrieved in the previous step to identify complex situations; the Act phase, that modifies the system in order to respond to what has been previously discovered.

The ODA Loop concept was originally introduced for simple devices (e.g., embedded systems) to perform control mechanisms on physical systems [26]. These electronic systems were generally based on serial processors without any possibility to perform parallel tasks. On the contrary, today's mobile devices are far more powerful than the aforementioned ones, being equipped with a full multi-tasking operating system like Android OS [27].

In order to efficiently bring the ODA concepts on modern Android devices, the features provided by the Android SDK have to be taken into account. The proposed cODA methodology affords this situation splitting the ODA Loop in all its steps and implementing each one as a bunch of independent pieces of software. These components communicate via an Event-Based paradigm and exchange data using a *shared blackboard* approach.

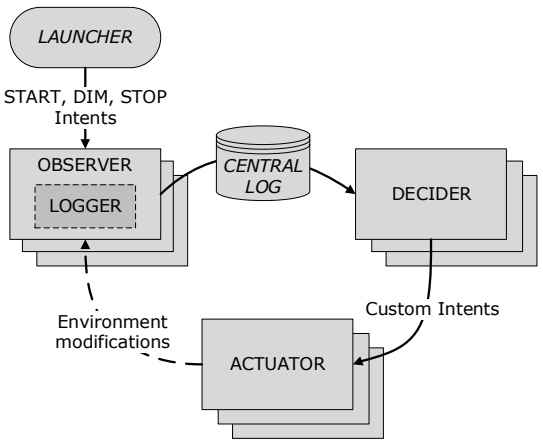


Fig. 2. cODA Loop

Fig.2 shows the overall system architecture. cODA.Observer components are mapped on Android Broadcast Receivers and are used to monitor environmental changes only when needed, so that they consume the least energy possible. cODA.Loggers are components used by cODA.Observers when we need to store and keep track of a continuously evolving state. The shared blackboard approach is mapped on the central cODA.Log component, that is an Android Content Provider: it provides the connection between the cODA.Observers and the cODA.Deciders, that may implement different policies to recognize specific patterns. As soon as a pattern is identified, a custom Android Intent is fired, thus connecting the cODA.Deciders to any listening cODA.Actuator component: these may even be external Android apps, that make the framework easily extendible. Finally, the feedback loop is implicitly obtained: as a cODA.Actuator modifies the environment conditions, the interested cODA.Observers will eventually sense the change.

The cODA framework itself needs to be aware of the state of the system: for instance, as soon as the battery is low, the cODA.Observers must limit their monitoring operations in order to reduce the overall consumed power. For this reason, we introduced the **cODA.Launcher** component: it is an Android Broadcast Receiver that starts with the system and sends commands to all the cODA.Observers via Android Intents. A first START command is used to start all the components. In order to make the framework power aware, the cODA.Launcher intercepts also all the events related to battery level changes. When the battery is low, the launcher sends the DIM command to ask the cODA.Observers to activate all the needed procedures to reduce their respective power consumption. Finally, we defined a STOP command, that pauses all the cODA.Observers: in this way, the flow of the cODA Loop is cut and, thus, the overall process is suspended.

The **cODA.Observers** acquire the information from the system and the environment, independently from how this will be used. Each cODA.Observer is implemented as a Broadcast Receiver that listens for the aforementioned START, DIM and STOP commands. When the observation phase requires a long running job, a **cODA.Logger** has to be used. A practical example is when a hardware sensor (e.g., the accelerometer) is involved in the observation: the incoming data must be sampled for a long period of time and usually they are not stored permanently till the end of the sampling period, mainly not to stress the permanent memory but also to be able to accomplish parsing and filtering that may be impossible to be carried out at a single sample level. Such a logger is then implemented as an Android Service that, as soon as it starts, begins to sample the sensor data by registering a callback and storing each sample into a local data structure. It guarantees that the sampling rate is constant by discarding too fast samples and using cache for too slow rates (this is necessary since Android gives no guarantee on stability of a sensor's output rate). This component is *optional*, since it is just an utility to perform the observation phase when a cache of the historical value under observation is required: this kind of caching operation cannot be performed directly by a *cODA.Observer* since it is an Android Broadcast Receiver and its life-cycle ends in a very short time (specifically when the *onReceive* method returns).

As said, cODA.Loggers are components used by

cODA.Observers when we need to store and keep track of a continuously evolving state. The shared blackboard approach is mapped on the **central cODA.Log** component, that is an Android Content Provider: it provides the connection between the cODA.Observers and the cODA.Deciders, that may implement different policies to recognize specific patterns. In the present implementation the data is persisted via a SQLite database that consists of a single table with timestamp, observer name, sampled values and expiry timestamp fields. The last field is used by the Content Provider for garbage collection, in order to avoid an uncontrollable growth of the database.

cODA.Deciders are in charge of finding patterns and valuable complex information in the middle of the huge amount of raw data recorded into the central cODA.Log. A cODA.Decider is a component that either is triggered by specific events occurrence - in this case it will be implemented as an Android Broadcast Receiver - or, it performs periodical checks on the data present in the central cODA.Log - in this other case it will be implemented as an Android Intent Service listening to the cODA.Launcher commands (START, DIM, STOP), in a way that is similar to cODA.Observer/cODA.Logger pattern but more compact, since the cODA framework enforces no restriction on the cODA.Deciders class type. cODA.Deciders signal potentially interesting situation via Broadcast Intents. A cODA.Decider is an Android Intent Service because it must execute periodical checks; this repeating behavior is achieved by letting the cODA.Decider listen to the cODA Launcher Intents (START, DIM and STOP) and schedule a repeating Android Alarm that triggers itself after constant delays. This pattern is very similar to the Observer/Logger one, but, in this case, since the framework do not impose any restriction on the cODA.Decider class type, it is possible to collapse the Android Broadcast Receiver and the Android Service parts into a single element: the same Android Intent Service stated above. cODA.Deciders signal potentially interesting situation via Broadcast Intents.

The actions to be performed are finally executed by the **cODA.Actuators**. These are only logical components from the cODA prospecting and may represent totally independent applications that are activated when an Android Intent from a cODA.Decider is received. These applications are suited for effectively acting on the user's environment.

V. Case Studies

We used the cODA framework to create different case studies inspired to the examples cited in Section II: the jogging and the wake up time detection. The goal of these case studies is to demonstrate that, with the proposed framework, it is possible to design complex context aware applicatiosn in an easy way. In the next paragraph, the implementation details of the developed case studies will be presented. The code of these Apps is open source and can be download from https://github.com/mattferroni/cODA.

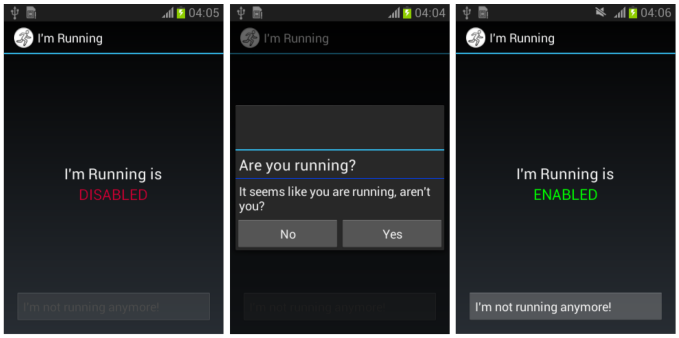
A. I'm running

"I'm Running" is an Android App that automatically silences the smartphone while the user is jogging. The App pops-up (Fig. 3.a) when the cODA framework detects that the user may be running and prompts him/her to confirm it

	Observer	Logger	Decider	Actuator
I'm running	**Broadcast Receiver** Schedules the repeated activation of the Logger.	**Service** Samples Accelerometer data at fixed rate.	**Intent Service** Periodically joins and synchronizes data from the Loggers, filters and classifies them.	**Broadcast Receiver and Activity** Silences phone and set a text auto-responder when running.
	Broadcast Receiver Schedules the repeated activation of the Logger.	**Service** Samples Gyroscope data at fixed rate.		
You may be late	**Broadcast Receiver** Register the Logger to receive location updates from the system.	**Intent Service** Parses location updates coming from the system.	**Broadcast Receiver** When an alarm clock is set, checks if the user has enough time to reach the successive appointment, based on past locations.	**Broadcast Receiver and Notification** Notifies if the user may be late for his/her next appointment.

TABLE I. Case studies framework component implementation

(Fig. 3.b). Since it is highly probable that the user will not be able to see the dialog if he/she is really training, after a timeout, the application automatically excludes the possibility of a false positive and enables the silencer/auto-responder (Fig. 3.c). When "I'm Running" is active, the smartphone is totally silenced (no vibration, no ringer). Every incoming call will be answered with a text message telling the caller that, at the moment, the callee is in a training session and asking him/her to recall later. When the user stops jogging, he/she will have to disable the App (that in the meanwhile remained on foreground) by simply tapping a button. All the missed calls will be present in the telephone log as usual.



(a) (b) (c)

Fig. 3. I'm running GUI

To properly detect whether the user is running or not, it is necessary to gather the data coming from both the **accelerometer** and the **gyroscope** sensors available on the smartphone. This is accomplished by two separate *cODA.Observers*, one for each sensor. Both these components do not directly record the sensors data, but they simply schedule the activation of a *cODA.Logger* that executes it on their behalf, managing all the related issues discussed further; this scheduling is accomplished by means of repeating Android Alarms. In order to perform jogging detection, some peculiar patterns have to be recognized on the data coming from the sensors over a temporal window. As a consequence, *cODA.Loggers* are used in order to perform an efficient recording of the input data.

The applicative logic that detects whether the user is running or not is encapsulated into a *cODA.Decider*, called

cODA.MotionDecider. When the *cODA.MotionDecider* starts, it retrieves the sensors data from the central *cODA.Log* within a definite temporal window (in the current implementation, the last 3 minutes), synchronizes them (since there is no guarantee that both the samples coming from the accelerometer and the gyroscope have been acquired at the same time instant), splits the entire temporal window into smaller and overlapping time slices (in the current implementation, half-overlapped 2 seconds slices), extracts some features from each one of them via a Principal Component Analysis and labels them via a 2-nearest neighbors classifier trained with a static dataset; the methodology that underlays this algorithm is fully analyzed in [28]. If the number of slices labeled "running" are enough to exceed a threshold, that is currently statically defined after the results of some experiments carried out without demanding completeness, the user is considered to be running and the RUNNING Intent is fired. For the entire process it was decided not to imply any GPS data, because when a user is running the smartphone is probably in his/her pocket and so the GPS fix may become very difficult and power consuming. Accelerometers and gyroscopes instead are less power demanding and their effectiveness is not influenced by the position of the device. The only factor that can affect power consumption is the sampling rate, which in the present implementation is set at 50Hz.

The only *cODA.Actuator* in this case is the "I'm Running" App itself. It is made of an Android Broadcast Receiver that listens to the RUNNING Intent fired by the *cODA.MotionDecider* and an Android Activity that is started by the previous component and allows the user to see the state of the silencer/auto-responder and to disable it.

The current demonstrative implementation is based on the results stated in [28]. It is important to notice that the goal of this case study is not to create the most precise application for the jogging detection but to demonstrate that the proposed framework allows any designer to create complex context aware application in an easy way. The main limitation this case study is affected by is related to the *cODA.MotionDecider* algorithm. There is an important trade-off in designing it. A more complex algorithm would be able to detect if the user is running in a more precise way and would require higher sampling rates, but the impact of this improvement on resources utilization and power consumption is hard to be modeled and is out of the scope of this work. A secondary limitation is about the user interaction: Android OS no longer exposes any API to hang up incoming calls directly. This forced the decision to implement the "I'm running" App as a silencer and not as a proper text auto-responder.

B. You may be late

"You may be late" is an Android App that alerts the user with a notification if he/she has set the alarm clock too late to reach the subsequent appointment on his/her device calendar. The app estimates and suggests the correct time to set the alarm clock to be on time.

To be able to estimate how long it will take for the user to get to his/her next appointment, it is necessary to know where he/she may be at the time when the alarm clock is set. This is accomplished by reading the user location and comparing it with the location of the appointment.

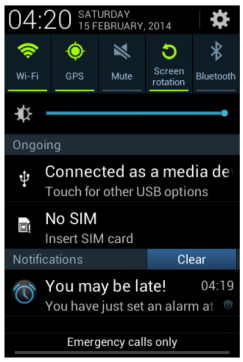


Fig. 4. You may be late GUI

The *cODA.LocationObserver* is a standard *cODA.Observer* that registers a *cODA.LocationLogger* for receiving Android OS LOCATION_STATUS_CHANGED Intents at repeating intervals in space and time. The *cODA.LocationLogger* is an Android Intent Service that parses the information present in the LOCATION_STATUS_CHANGED Intents broadcasted by the system and records them into the central *cODA.Log*.

The *cODA.LateDecider* is a standard *cODA.Decider* in charge of understanding whether the user may be late or not when he/she sets a new alarm clock. Since this check is strictly consequent to a user action that causes the system to broadcast a SET_ALARM Intent, this component is an Android Broadcast Receiver that listens only to this specific intent. When triggered, this *cODA.LateDecider* retrieves the user location at the new alarm clock time in a defined temporal window (in the current implementation, the last 2 weeks), compares them in order to detect how many conflicting locations have been found and, if the number of conflicting records is less than the maximum number of matching records, this last location is set to be the start location to reach the subsequent appointment; then, the *cODA.LateDecider* retrieves the first appointment subsequent to the alarm-clock time and sets its location as the end location, if any; finally, the *cODA.LateDecider* connects to the Google Directions Web API to compute the journey duration and compares it with the difference between the appointment start time and the alarm clock time. If the journey duration is greater than the difference, the cODA.Decider computes the delay and minimum alarm-clock time and fires a LATE Intent.

The cODA.Actuator is the couple Android Broadcast Receiver/Notification present in the "You may be late" App. The Android Broadcast Receiver, as imposed by the cODA framework, listens to the LATE Intent and triggers the Android Notification visualization (Fig. 4). Since the Android preinstalled Alarm Clock App does not fires the SET_ALARM Intent when a new alarm clock is set, the "You may be late" App includes a simple Alarm Clock App that overcomes this issue. This same App is started when the user taps the "You may be late" notification.

C. Further Examples

The cODA methodology is meant to propose a standard design pattern to support any kind of application that involves context awareness in a mobile environment. There are

numerous examples of application fields that have not been explored in the case studies. If we think about sensors, one interesting application field may be health care. Wearable sensing elements with on board diagnostic capabilities, ECG test units for example, and a wireless connection, can easily become a point from where to start to build an ODA Loop that can timely deliver life saving alerts, or compute calories consumption to support users on diet, and so on. It is possible to envision a single central Health Decider able to broadcast sticky intents stating an abstraction of the user health state in every moment, and a suite of apps implementing smart behaviors, such as forbidding the user to open a navigator app when the sensor have detected that the user is not able of driving, or tuning the pacemaker parameter to react at a sudden burst of stress.

VI. Conclusion And Future Works

We presented a methodology and a framework based on it, namely cODA (*context Observe Decide Act*), targeting the great variety of Android devices. The proposed methodology aims at providing the guideline to develop context aware applications on the selected family of mobile devices, exploiting the features of the selected platforms. An asynchronous communication pattern among the Observe-Decide-Act loop components, on which the context aware applications are based, is presented by means of Android Broadcast Receivers and Android Intent. The methodology has then been implemented into the aforementioned open source cODA framework and all the implementation details have been shown. We finally built different case studies using the cODA framework, showing the effectiveness of the proposed approach.

Some improvements for the cODA framework are left as future works. At first, it is possible to introduce some artificial intelligence libraries to provide a better decision phase. All the cODA.Deciders may have an enormous advantage leveraging on machine learning techniques for a better adaptation to the user's habits and needs. This could be partially implemented on a remote server and integrated via periodical updates of the parameters on the device. Secondly, in order to let the user access a large variety of cODA.Actuators, a marketplace could be implemented, from where the user could download and install them. To do this in an optimized way and in order to avoid code or functionality duplication, a packet manager that keeps track of installed software and package dependences should be deployed.

References

[1] motorola.com, "Motorola startac3000," http://www.motorola.com/mdirect/manuals/StarTAC3000_User_Manual_E.pdf.

[2] nokia.com, "Nokia 3210," http://nds1.nokia.com/phones/files/guides/3210_usersguide_it.pdf.

[3] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Handheld and ubiquitous computing*. Springer, 1999, pp. 304–307.

[4] R. J. Robles and T.-h. Kim, "Review: Context aware tools for smart home development." *International Journal of Smart Home*, vol. 4, no. 1, 2010.

[5] G. Thyagaraju and U. P. Kulkarni, "Design and implementation of user context aware recommendation engine for mobile using bayesian network, fuzzy logic and rule base," *International Journal of Pervasive Computing and Communications*, vol. 8, no. 2, pp. 133–157, 2012.

[6] G. Chen, D. Kotz *et al.*, "A survey of context-aware mobile computing research," Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, Tech. Rep., 2000.

[7] J. Pascoe, N. Ryan, and D. Morse, "Issues in developing context-aware computing," in *Handheld and ubiquitous computing*. Springer, 1999, pp. 208–221.

[8] A. Nacci, M. Mazzucchelli, M. Maggio, A. Bonetto, D. Sciuto, and M. Santambrogio, "Morphone.os: Context-awareness in everyday life," in *Digital System Design (DSD), 2013 Euromicro Conference on*, Sept 2013, pp. 779–786.

[9] canalys.com, "Android on 80http://www.canalys.com/newsroom/android-80-smart-phones-shipped-2013.

[10] M. Weiser, "The Computer for the 21st Century," *Scientific American*, vol. 265, no. 3, pp. 94–104, Sep. 1991.

[11] W. N. Schilit, "A System Architecture for Context-Aware Mobile Computing," Ph.D. Thesis, Columbia University, 1995.

[12] I. MacColl et al., "Shared visiting in EQUATOR city," in *Proceedings of the 4th international conference on Collaborative virtual environments - CVE '02*. New York, New York, USA: ACM Press, 2002, pp. 88–94.

[13] P. Korpipää and J. Mäntyjärvi, "An ontology for mobile device sensor-based context awareness," *Modeling and Using Context*, pp. 451–458, 2003.

[14] A. Dey, G. Abowd, and D. Salber, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," *Human-Computer Interaction*, vol. 16, no. 2, pp. 97–166, Dec. 2001.

[15] M. Raento, A. Oulasvirta, R. Petit, and H. Toivonen, "ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications," *IEEE Pervasive Computing*, vol. 4, no. 2, pp. 51–59, Apr. 2005.

[16] Techcrunch, "Nokia confirms the pureview was officially the last symbian phone," *Nokia Corporation Q4 and full year 2012 Interim Report*, January 2013.

[17] D. Chu, A. Kansal, J. Liu, and F. Zhao, "Mobile apps: It's time to move up to condos," in *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, ser. HotOS'13. Berkeley, CA, USA: USENIX Association, 2011, pp. 16–16.

[18] N. Vallina-Rodriguez and J. Crowcroft, "Erdos: achieving energy savings in mobile os," in *Proceedings of the sixth international workshop on MobiArch*. ACM, 2011, pp. 37–42.

[19] T. Yan, D. Chu, D. Ganesan, A. Kansal, and J. Liu, "Fast app launching for mobile devices using predictive user context," *Proceedings of the 10th international conference on Mobile systems, applications, and services - MobiSys '12*, p. 113, 2012.

[20] K. Ariyapala, M. Conti, and C. Keppitiyagama, "ContextOS: A Context Aware Operating System for Mobile Devices," in *IEEE GreenCom 2013*. IEEE, Aug. 2013, pp. 976–984.

[21] J. Bardram, "The Java Context Awareness Framework (JCAF)a service infrastructure and programming framework for context-aware applications," *Pervasive Computing*, pp. 98–115, 2005.

[22] G. Chen and D. Kotz, "Solar: An open platform for context-aware mobile applications," DTIC Document, Tech. Rep., 2005.

[23] B. Priyantha, D. Lymberopoulos, and J. Liu, "Little Rock: Enabling Energy Efficient Continuous Sensing on Mobile Phones," *IEEE Pervasive Computing*, vol. 10, no. 2, pp. 12–15, 2011.

[24] N. Xiong and P. Svensson, "Multi-sensor management for information fusion: issues and approaches," *Information Fusion*, vol. 3, no. 2, pp. 163–186, Jun. 2002.

[25] N. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. Campbell, "A survey of mobile phone sensing," *IEEE Communications Magazine*, vol. 48, no. 9, pp. 140–150, Sep. 2010.

[26] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, "Engineering self-adaptive systems through feedback loops," in *Software Engineering for Self-Adaptive Systems*. Springer, 2009, pp. 48–70.

[27] T. Wooley, "A comparative study of the android and iphone operating systems," *University of central Florida*, 2010.

[28] T. T. N. HO, "Activity recognition using smartphone based sensors," *Master Thesis Politecnico di Milano*, 2013.