mophone.OS: Context-Awareness in Everyday Life

A. A. Nacci\textsuperscript{1}, M. Mazzucchelli\textsuperscript{1}, M. Maggio\textsuperscript{2}, A. Bonetto\textsuperscript{1}, D. Sciuto\textsuperscript{1}, and M. D. Santambrogio\textsuperscript{1}
\textsuperscript{1}DEIB, Politecnico di Milano, Italy
\textsuperscript{2}Department of Automatic Control, Lund University, Sweden
\{nacci, maggio, bonetto, sciuto, santambr\}@elet.polimi.it,
\{matteo1.mazzucchelli\}@mail.polimi.it

Abstract—Mobile devices, due to their wide distribution and to their increasing smartness and availability of computational power, can become the interaction point between users and their surrounding environments. However, current mobile devices OSes lack of the ability to anticipate and overcome internal and external changes. Integrating mechanisms of self-awareness and self-adaptability in nowadays smartphones is an attractive perspective to match with these requirements. Moreover, adaptive behaviors can enhance the management by the mobile device itself, of the available resources at its best, e.g., the battery life. This paper envisions various situations in which a self-aware mobile device can interact with the surrounding environment and support the user in performing everyday actions. A prototype of such an adaptive device, called mophone.os and based on the Android OS, has been designed and implemented to verify the reaction of the device in different situations providing convincing and promising preliminary results.

I. INTRODUCTION

Mobile phones are indispensable in nowadays life. They are now cheaper, smaller and much popular than laptops. Furthermore, they are substituting laptops and desktops in a variety of functionalities [11]. Due to this growth in functionalities and the constant presence in our life, mobile phones are one of the most important interaction points between people and the surrounding environment. It is however to be considered that resources in such devices are much more limited, e.g., computational power is reduced and disk space is limited. Nonetheless, both internal and external conditions are rapidly changing and may influence the behavior of the entire system, e.g. switching between network types causes an unpredictable power consumption and the network instability. Two representative examples are a person moving from an area where signal is very powerful to a low-connected one and a switch between network types, that may cause unpredictable power consumption and network instability. In other words, mobile phones need context-awareness, i.e., they need to be aware of the surrounding environment conditions [5] and of their internal state [7]. With this knowledge, they should be able to optimize and to meet their requirements taking advantage as much as possible of the context to pursue concurrent goals, like optimizing the quality of a voice call consuming less energy. Context-awareness can be generally defined as the ability to collect and understand the information coming from the context. More formally, it could 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 [1]. A context-aware mobile device is an active system, capable of reacting to simple and direct inputs as well as to infer its operating conditions and adapts accordingly. It is able to learn and understand the place in which it is operating and the user habits and to reconfigure itself to prove the best possible user experience, even anticipating changes and reactions. For the purpose of this work, the context of a mobile device is split in three different components: Environment, User and Self.

The Environment (E) contains characteristics of the ambient in which the device has to operate such as lighting, noise level and location. The User (U) represents information about interaction between the user and the device, not necessarily related to a direct interaction with the mobile device but also including user behavior data. The Self (S) component is related to the device itself, which is able to provide internal information relative to the past and current state, examples being battery level history and computing units frequencies. In this respect, the aim of this work is to describe and implement context-awareness considering all the three components at the same time. In this work, we will:

- analyze the requirements and the goals of the introduction of self-awareness in a mobile device;
- develop a first implementation of the mophone.OS, a self-aware enabled version of the Android OS;
- introduce an applications performance monitor within the OS, than can be accessed both by native C applications and by Java applications;
- propose test cases to prove the effectiveness of the mophone.OS in different situations.

The remaining of this paper is organized as follows. In Section II some background concepts about how Android is able to sense the environment are explained and this work is compared to the state of the art. In Section III, the design phase of our system is discussed, introducing a preliminary requirement analysis phase and then the structure of the system control loop. Various test cases and scenarios are illustrated In Section IV and finally Section V concludes with some future works.

II. CONTEXT DEFINITION

Mobile devices belong to a constrained and fluctuating environment. Resources are limited, both in terms of performance and power, and their availability can greatly vary over time. The battery energy is an example of a resource that influences the whole system functioning and it has to be
carefully managed. As mobile devices have evolved from low-function devices to complex systems, resources management cannot disregard the environmental conditions. There is a strong need to make mobile devices context-aware. As a short definition, “self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation” [15]: Next Sections will provide some background knowledge about adaptive systems and context-awareness, along with some related work in the field.

A. Background

Nowadays devices have the availability of several sensors, that can be used to sense both the internal and external device conditions. The Android OS makes possible to access those sensors with standard APIs, but often those sensors are not enough to retrieve all the necessary information, or the sensors data are not as precise as required. In this Section, at first we will explain how the standard Android APIs can be used to sense the environment. Then, it is described an external framework that has been integrated into the Android OS to extend its sensing capabilities.

1) Android as a sensor platform: The Android SDK includes a set of API that can be used to access the available sensors and observe the device context. The device geolocation [3] is ensured by the Location framework. The main component of the location framework is the LocationManager system service, which provides APIs to determine the position of the underlying device, if the device supports this feature. The GPS interface is the most accurate source to determine the user position, but it only works outdoors, it is highly battery consuming and it can be not always available. If an accurate position is not mandatory, the user location can also be determined using cell tower or Wi-Fi signals, consuming in this way less battery power. Widely used wireless technologies, such as Bluetooth and NFC, are supported by the Android platform and can be used to discover the nearest devices in the surrounding environment. API are provided to scan for other devices, query for paired devices and exchange data between paired devices. Through the SensorManager it is possible to access all the hardware sensors available on the device. Those sensors may include accelerometers, magnetometers, pressure and temperature sensors. Each sensor is identified through a type, a sampling rate and an accuracy. The touch screen itself is recognized as a sensor. Since Android 2.3, several new sensor fusion [19] APIs including quaternion, rotation matrix, linear acceleration and gravity have been added. Sensor fusion is used to convert the combined raw signals from multiple sensors to create new, more-useful sensor types. Sensors can interact directly with the Linux Kernel drivers that pass data through the hardware abstraction layer (HAL) and up into the application layer. However, some issues may arise when dealing with sensors in the Android platform. There is no standard availability of sensors across devices and the Android platform is not optimized for real-time sensor data acquisition, and any delay in servicing and using the data can disrupt motion tracking.

2) The Heartbeats framework: A simple way to observe current and target performance of applications is described in [9] and [8]. The framework proposed, named Application Heartbeats (or, more simply, Heartbeats), provides a simple yet effective infrastructure to measure and monitor application progress toward goals. Crucial points in the design of the framework are simplicity and portability, while keeping a standardized interface. A set of API is used to express the applications goals and to monitor their progresses toward them. Each application has to be monitored so that an heartbeat is issued whenever there is a significant point in which application progress wants to be controlled. During the initialization phase, goals can be expressed in terms of a window between a minimum and a maximum threshold, such as the lowest and highest heart rate the application should generate or the minimum and maximum latency between two heartbeats. An heartbeats monitor is created for each monitored application and its task is to collect generated heartbeats and provide information about the application history and its rate over time.

B. Related Work

Reasoning along the mentioned dimensions (Environment, User and Self) a few works discuss how it is possible to correctly sense the environment, i.e., consuming only the necessary power and saving only relevant data, such as [12], [17]. The interested reader can also refer to a survey on the matter for further information [10]. Also, the user dimension has been explored. A study conducted on two different dataset was aimed at finding patterns in smartphone usage [6]. The authors characterize the phone usage along different key dimensions, being those user interactions, applications use, network traffic and energy drain. They found a huge different among users in their device interaction patterns and concludes that optimization strategies have to be adjusted per single user to be effective. Much more has been done for enabling Self sensing and relative adaptations, especially in the power consumption domain. Carroll and Heiser studied the power consumption of a Openmoko Neo Freerunner mobile phone, the circuit schematics of which are publicly available [4]. The main purpose of their study was to effectively understand where and how the energy is used and as a main results, they produced a breakdown of power distribution to processing unit, memory, touchscreen, graphics hardware audio, storage and networking interface. The study relates on the phone schematics and it is therefore not portable to other devices but the contribution in terms of discovering where to intervene is crucial. Another key idea for modeling power consumption is to build and validate a power model based on the battery discharging rate [20]. PowerBooster and PowerTutor automatically build a model that uses built-in battery voltage sensors and knowledge of the battery discharging to monitor power consumption and control the power management of individual components. This work deals with the Self level but is related to correctly sensing the phone and not to react and adjust based on the obtained numbers. Another important contribution
modeled the device components power consumption based on a logging application released “into the wild” [16]. The authors used a regression-based power estimation model that relies on accessible measurements that could be collected by a logger with privileges to access the phone state. Traces were collected for twenty different users, demonstrating diversity in the phone usage and components contributions. The authors also propose a solution for reducing power consumption based on change blindness, or the inability to perceive very slow changes happening in the environment; more specifically they reduce the screen brightness whenever the device is used for a long period of time, in order to save battery. They verified experimentally that users are relatively unaware of this change or accept it with limited complains. This is an example of user and self-adaption, in the sense that the device is able to adapt itself based on the user experience, although in a very preliminary way. A resource provisioning system named Application Resource Optimizer [13] was presented, with the aim of profiling the cross-layer interaction among phone layers including radio resource channel state, transport layer, application layer and the user interaction one. Also in this case, a single adaptation level is enabled, while our work is more general and could include strong contributions like the mentioned ones and others like [2], [14] in a unitary framework capable of choosing the correct adaptation mechanism based on sensing capabilities. The contributions in the previous described works are generally limited to one specific domain, e.g., power consumption. This work aims instead to propose a general framework, integrated inside an existing operating system, to enable adaption at different levels and possibly to coordinate different adaptation types. In this work we will explain several applications that exploit our framework with different goals and sensing capabilities.

III. SYSTEM DESIGN

Many information about the surrounding environment can be obtained using a mobile device, for example the current position can be retrieved through the integrated GPS or, less accurately, using the network positioning. Sensing the device environment is therefore an easy task. Collecting data about user interactions and the internal state of the device seems instead to be more complex, due to the lack of enabling technologies. In this work an important step in this direction is made, introducing a powerful enabling solution for sensing mobile devices applications. In order to develop a complete self-aware system on a mobile device, we analyzed the requirements of such a system. In the following, we first highlight the targets that the system should optimize and the paradigms on which adaptivity can be applied. After this phase, we discuss a possible structure for the basic control loop that could manage adaptation from observations to actions.

A. Requirements: Targets and Paradigms

Performance has always been the primary issue for every kind of computing devices. The term performance is commonly used in the sense of *as-fast-as-possible execution* but this is not what we want in a mobile environment. In fact, since we are dealing with mobile phones, also power and energy consumption are playing a key role. These goals are obviously not isolated one from the others. Power reduction cannot be blindly performed, just to save as much energy as possible, because this may cause a loss in performance and a discomfort to the user. Contemporary, performance could reduce battery life and cause problems to the user. It is therefore necessary to manage performance and power objectives jointly, at the same time, to reach an adequate trade-off between the two.

To introduce this trade-off at system level, the performance evaluation is transformed in a Quality of Service (QoS) concept. For this purpose, we are using the Heartbeat framework, as will be explained in Section III-B1. The performance goal will ensure to provide a user an adequate QoS, with respect to the type of applications the user is demanding and the general system conditions. Efforts will be done to ensure the minimum QoS also in varying system conditions. Our goal is not to provide always the best possible performance for each application. A power goal is used to ensure the system is not consuming more power than required to meet its performance goals. The Android framework lacks mechanisms to retrieve a precise power consumption estimation of single components or of single applications, even if there is ongoing research on the matter [4], [13], [16], [20]. The research effort is distributing in various directions and different power models are envisaged, however it is not clear how to combine them. A precise power monitoring system is required, coming from the combination of literature strategies or from direct measurements. A work in progress in that direction is illustrated in Section IV-B.

Apart from these two main goals, many other smaller or composed goals can be optimized in a mobile device. Those goals are usually not isolated from the main two goals just described, but they are often parts of them or related to a specific problem. An example is the case of network managing [2], whose target goal can be a mixed goal composed by a power goal and a stability one. In this case, the primary goal is to avoid net fluctuation to achieve a better stability, but the side effect is a reduction of the power dissipation due to the continue change of network.

Two paradigms are possible and can coexists in this system, as shown in Figure 1.

![Diagram](image)

**Fig. 1:** (a) *morphone.app*: a self-optimizing application. (b) *morphone.OS*: optimization of machine parameters by an external observer.

In the first paradigm, applications are self-monitored and self-adaptive, while in the second one, they are monitored
by an external entity which can enable actuators to perform actions to modify global system conditions. Application knobs and global knobs belong to the same framework and are entities that can act on the system configuration in terms of objectives. The two paradigms are not mutually exclusive and the could live together with the only restriction that an application is denied to modify global parameters. This is due to the lack of knowledge, on the application side, of the system global conditions. On the other hand a global entity, if necessary, can modify single application parameters to orchestrate diverse components to reach a global target. In details:

- a morphone.app is a self-adaptive application which integrates an adaptation loop inside the application itself. Such an approach is used when actions are performed to tune application specific parameters, and a global vision of the system is not necessary. In this case, only application specific performance goals can be taken into consideration,
- a centralized approach uses an external observer to monitor all the monitorable applications and activate adaptations and actions on them. This approach relies on a global vision of the entire system, therefore system wide actions can be applied. For example, an external observer is necessary to monitor and act on power consumption, since changes at the global level affecting power management will reflect on all the applications running in the device.

The system should finally be able to react both to internal and external changes, since a mobile device is not isolated from the surrounding environment and also external conditions can modify the device behavior.

B. The Control Loop

The development of self-aware adaptive computing systems is strictly linked with the basic capabilities their decision and control loops provide. Design this feedback loops is the first step in building a self-aware and adaptive system. These loops are generally referred as ODA (Observe, Decide and Act) loops. In our case, the observe phase collects data from all the adaptive applications that expressed their intention to be monitored, and from various system components. The type of data collected in this phase strictly depends on the objectives that have to be optimized by the feedback loop. Those data are collected by a central element, that has a global vision of the system, knowing which applications are currently being monitored, their goals and which policies and actuators are available to adapt the applications behavior and the system itself in order to meet local and global goals. Participants in the decide phase are this central entity, named Services Coordinator (SC), a set of policies that could be activated to perform adaptation and a set of targets to be met. A target is a specific goal that an application wants to reach. The act phase is performed by one or more actuators, that are in charge of executing specific actions on the system to vary actual performance. Figure 2 shows the combination of the different phases.

The combination of a policy (a way to take decisions), a target (the quantity to be optimized and its desired value, as described in Section III-A) and one or more actuators (tuning points or knobs) is called a Service. Using this notation, the SC will be aware of all the available services. In this case, a single central entity is envisaged and acts as a decision engine. However, also a self-adaptive application can be mapped into the proposed approach. In fact, the application can monitor itself and choose a suitable actuator. In this case monitor and actuators are integrated in the application itself and do not necessarily need to be exposed to the overall system. Applications could also choose to expose their actuators to allow system-wide modifications. In fact, applications developers are encouraged to use a system interface to define some tuning knobs, embedded inside the applications.

Each phase will be now described into details.

1) The Observation Phase: Observation is a crucial phase in the loop, and the behavior of the overall system relies on the sensors and monitors ability to gather information about the online execution. The main monitor is represented by the application performance monitor, in charge of collecting data about the performance of instrumented applications running on the device. To monitor performance, the proposed solution makes use of the Application Heartbeats API [8] to monitor the speed of an application and its progresses toward its goals. Each application is responsible for deciding and expressing its goals in such a way that they can be used by the heartbeat monitor (a minimum and maximum “heart rate” threshold) and is instrumented with the framework. This means that the application takes care of issuing an heartbeat each time it executed a substantial portion of its the code. A monitor is instantiated for each application that has to be monitored and it is integrated inside the application itself. The application and an external entity (for example SC) could retrieve the application data. This framework is designed for Unix systems, thus it has been integrated with this work inside the Android framework, to be used both by Java applications and native C applications.

2) The Decision Phase: The Services Coordinator stands at the center of the decision loop and exploits the awareness given by the available observation mechanisms to elaborate a
plan for future behavior. It exploits the information coming from the monitor in order to determine the available services and the monitored applications and constantly verifies the availability of new or old services and the presence of new or old monitored processes and it gathers the information coming from the monitored processes. SC than analyzes the performance-related data in order to understand whether to enact a correction policy; and to decide which services to enable or disable. In order to take such a decision, it must be aware of all the running applications that want to be monitored and of the state of each implemented service. For each service, it knows if the service is available or not and which is the target that it is able to optimize. Service availability is not obvious because since more services can use the same actuator, a specific service may not be available due to the fact that another service is using the same actuator. It is our idea to keep the SC as simple as possible. The interface between the Services Coordinator and the services should be extremely flexible and a model of a service should not be imposed, nor SC should need to be updated or restarted when new services are plugged in, to the extent of self-configuration. Apart from the decision process performed by the SC, all services will embed a decision making mechanism that is independent from the Services Coordinator. When a conscious service is enabled its own loop-based decision mechanisms is activated. Services are active components, able to perform actions on the system based on their policies. Therefore, there are two decision levels, a global and a local one. The global level is to activate and deactivate local decision making mechanisms. These local mechanisms optimize for objectives using tuning knobs and contain a decision policy.

3) The Action Phase: in our frameworks the actuators are integrated inside services. There are many kinds of services that might be available and that might affect the system in different ways; some might affect performance, accuracy, or both; some might impact on the whole system while others might target one or more applications. Examples of actuators can include application knobs, i.e. components able to modify and tune a specific parameter of the application taken into consideration, frequency and voltage scaling, core allocation. They can be categorized into application-specific or system-wide actuators, depending on whether they are able to modify only local application parameters or system-wide behaviors.

IV. APPLICATIVE SCENARIOS

In this Section, we are proposing different scenarios to demonstrate the need on context-awareness in nowadays mobile devices. The device used in this phase is the LG P500 smartphone, single core and equipped with Android 2.3 Gingerbread. It was integrated with the complete adaptive system based on the methodology and the structure illustrated in Section III, where the ODA loop is implementing the self-aware and self-adaptation part of the system. The videos demonstrating the proposed scenarios can be found at the morphone.OS\(^1\) on the CHANGE group website.

1\[http://www.changegrp.org/morphone\]

A. Tradeoffs between different goals

One of the major problems in adaptive systems is to find a way to reach a tradeoff between two or more concurrent goals. In a mobile device, the two main objectives are the ones related to performance and power consumption. The following test is intended to explore the behavior of actions in order to reduce the global system power consumption on the single applications performance. To prove how the heartbeats framework can be useful in monitoring the performance of an instrumented application, an Mp3-Decoder application was implemented as an Android application. It was indeed developed both as a native application and using Java and the Android SDK, in order to test the behavior of the Heartbeats framework in both these layers. Specifically, MP3decoder is able to decode and play a stream of data encoded in the MP3 format using different decoding algorithms. Currently, the miniMP3 and the MPG123 algorithms are supported (the former is characterized being slower than the latter, but leading to a better audio quality). This specific implementation makes possible the usage of two different actuators, the first one able to resize the buffer of data decoded in each iteration at runtime, the second one able to change the decoding algorithm used at each iteration to decode a buffer of data. The application was instrumented with the Heartbeat framework in such a way to emit an heartbeat every time a buffer of data is decoded, no matter what the decoding algorithm is. A CPU frequency scaler actuator has been activated to reduce the system power consumption, by blindly decreasing iteratively the frequency until the minimum value is reached. Both the CPU frequency scaler and the MP3-decoder application are controlled by the system through the Services Coordinator. Figure 3 shows an application execution with the miniMP3 decoding algorithm selected and an initial buffer size of 160KB. The application’s performance is expressed as an heart rate and the execution shows that it is influenced by the CPU frequency reduction. This test proves not only that the heartbeats framework is useful to monitor an application performance, but also that it is possible to use this information to apply adaptive techniques to reach an adequate goals tradeoff. In this specify case, the tradeoff consist of decreasing the CPU frequency as much as possible without impacting on the application performance QoS. It is possible in this case to activate the buffer resized actuator to reduce the buffer size until the value of 152KB to
restore a good QoS and still be able to keep the CPU frequency to a low value, without letting the application heart rate staying under the minimum threshold.

B. Power Management

The main pitfalls in the previous described approach are that every single application has to be instrumented and application specific actuator have to be implemented. Such specific cases can rarely be useful standalone but have to be applied in conduction with a global system power management. The power management system we are developing rely on a client-server architecture and it is based on a strict interaction between the user and the system. Data about the device usage are collected and analyzed, in order to generate an accurate and device-specific power consumption model and to provide the user suggestions on how to optimize the battery life. The adaptive process is performed both online on the device and offline on a separate server. The online process, shown in Figure 4, is performed mainly by a background service, which is responsible of monitoring the system conditions. Most of the information are retrieved using public Java Android APIs. Those low-level information not accessible from the Java layer can rarely be useful standalone but have to be applied in conduction with a global system power management. The server side is composed of a server architecture and it is based on a strict interaction between the user and the system. Data about the device usage are collected and analyzed, in order to generate an accurate and device-specific power consumption model and to provide the user suggestions on how to optimize the battery life. The adaptive process is performed both online on the device and offline on a separate server. In the case of studies will be presented in order to clarify this aspect.

Fig. 4: Client-side power management architecture.

Fig. 5: Server-side power management architecture.

C. Ease your life

Making a mobile device aware of its context makes the user experience better: the main idea is that the device should not be pervasive into the user life but it has to adapt his behavior in order to make it easier. Into the following section two different case of studies will be presented in order to clarify this aspect.

1) Automatic alarm setting: Nowadays it is common to use mobile devices to set appointments, reminders and notes. As a consequence, the device can be aware of the user agenda and it can anticipate common actions based on that information, to ease the user’s life from repetitive commands. One action that is commonly performed using a mobile device is the morning alarm setting. We have designed a simple service for this purpose. This smart alarm service is able to understand what is the best time for the user to wake-up. The alarm time is computed using few information:

- the time TA of the first appointment of the day next to the current one;
- the location LA of this appointment;
- the time TB of the last appointment of the day next to the current one;
- the location LB of this appointment.
the time-to-get-ready \( TR \), intended as the time necessary for the user to get ready starting from the alarm time.

Given the \( LA \) information, it is possible to compute the trip time \( TT \) needed to reach the first appointment location, considering as a starting point the current location (usually the home position) and a car trip. These parameters are user-defined and the \( TT \) is computed using one of the available real-time traffic information websites. The \( TR \) information can be obtained in different ways. In our implementation, this value is firstly initialized by the user, and then it is afterwards learnt given the past recorded times between the set alarm time and the time when the user is actually ready to go out. This last information can be retrieved, for instance, using the GPS. After having retrieved those information, we can compute the best time to wake-up with the formula (1)

\[
T_{\text{wakeUp}} = TA - TT - TR
\]  

When it is recognized that the user wants to set the morning alarm for the following day, a notification with the suggested alarm time appears. We have decided the unobtrusive way to let the user decide whether to set the alarm or not, by clicking on the notification popup, rather than automatically setting the alarm without the user approval. To summarize, the device must be aware of the user’s agenda and the current position. It has to recognize the user intention to set the alarm and it should be able to provide a correct suggestion about the waking time. It should finally be able to adjust the time-to-get-ready parameters on the past user behaviors.

2) Getting driving directions: A scenario like the previous one involves getting driving directions whenever the user gets into his car. The device must be able to recognize that the location where it is found to operate, it is the user’s car. This can be performed, for instance, exploiting the car’s bluetooth connection. The phone can then check in the user’s calendar the next appointment’s location and automatically compute the driving direction to the selected location. In this situation, the mobile device has to be aware not only of the current position, intended as GPS position, but also of the location significance. In other words, he has to understand that the device with the specific bluetooth ID it has just recognized is the user’s car, and based on this awareness he has to react to the situation, in this case computing the requesting driving directions.

D. Improve device integration

Most people today have more than one mobile device: smartphones, tablets, laptops. And many of them also have some fixed location. What we see is that there is no efficient integration between all these devices: into the following sections two different case studies of an improved integration are shown.

1) The Swish protocol: Let’s take the example of a user who is writing a document on his mobile device while he is going to his work place, document that will be part of a presentation for the next morning meeting. Writing a text is an operation that can easily be performed on a mobile phone, while composing a presentation on a such small device can be more complicated. He decides then to continue his work on his office laptop. Normally, he has to save the previously written text in a document and use a shared folder or an email to make the document available on his laptop. This operation can be time consuming. The swish protocol has been developed as an idea to easily transfer contents between different devices, with different characteristics. We will now explain the terminology involved.

- The active device is the predominant device, that is the device that is being used by the user at the current time.
- It is the source device, in the swish effect.
- The paired device is the nearest device with respect to the active device. It represent the destination of the swish effect. It is considered paired if a transfer has already been performed, not paired otherwise.
- All the devices that are associated with the same user are called local devices. A swish effect between two devices belonging to this category does not require an explicit approval from the user.
- Two or more devices not associated with the same user are called foreign devices. A swish effect between two foreign devices require a prior authentication phase.

The context switching between the active and the paired device is performed with a specific gesture on the active device. After that, the current view is transferred to the paired device. Figure 6 illustrates this concept. In this scenario, the device is able to discover the devices in its neighbourhood. It is able to know if they are associated with the same user and perform a context switch between different kind of devices.

2) No-stress mode: Usually, all the devices belonging to the same user are synchronized with the same email and calendar accounts. In this way, every time an e-mail is received or an event reminder is shown, all the devices notify the same event at the same time. Normally a user is active on a single device at a time and multiple notifications are useless and annoying. Consequently, a synchronization mechanism to identify the predominant device currently in use can be implemented. When this device is identified, all the notification will be delivered only on the predominant device. The mechanism works in such a way that the predominant device has to declare to all the other devices that it is the one. After that declaration, all other devices must enter the no-stress mode.

In this operational mode, notifications are hidden from the user. This information can also be used to reduce the device energy consumption, for instance decreasing the check inbox frequency.
The use of other techniques, e.g. retrieving the user position, the use of GPS is high power consuming components. For instance, considering the case of text, since many information are provided through high energy consumed while retrieving information from the context, the introduction of adaptivity could really be beneficial to understand the user behavior. The device, through its self-awareness capabilities is able to know that an application capable of tracking the running session is in execution, while through Environment-awareness it could understand that the retrieved speed increases. In this sense, the information “the user is jogging” is inferred from both the self and the context components, even if the user does not explicitly say that to the device. If only one between self-awareness or Environment-awareness was used, a misleading conclusion can be achieved. Only the conjunction of more context components can help the correct detection of the user current behavior.

V. Conclusion and Future Work

This paper presented a context-aware mobile device obtained implementing and integrating adaptive features into the current Android OS. The approach was validated in different scenarios, the results of which are encouraging and showing that the introduction of adaptivity could really be beneficial in the mobile context, making the system able to optimize different targets in a coordinated manner in order to achieve a global optimization of the overall system, depending on the varying internal and external conditions. This work will be further extended. Due to the growing importance of the power saving aspect in the mobile world, it is important to find new ways to save energy. One idea could be to reduce the energy consumed while retrieving information from the context, since many information are provided through high energy consuming components. For instance, considering the case of retrieving the user position, the use of GPS is high power consuming. The use of other techniques, e.g. GSM footprints [18] can help in reducing the impact of the data collecting phase on the overall power consumption. On the other hand, it is possible to exploit the context information to apply OS related policies in order to reduce power consumption. In this direction, the power management work presented in this paper will be extended to obtain a precise and device-specific power model consumption.

REFERENCES


