gLCB: An Energy Aware Context Broker

Luca Ardito[a], Marco Torchiano[a], Marco Marengo[b], Paolo Falcarin[c]

[a] *Politecnico di Torino, Department of Control and Computer Engineering, Corso Duca degli Abruzzi 24, 10129 Torino, IT*
[b] *Telecom Italia, Strategy & Innovation, Via Reiss Romoli 274, 10148 Torino IT*
[c] *University of East London, School of Architecture, Computing, and Engineering, Docklands, E16 2RD London UK*

Abstract

Context. Worldwide mobile device sales will reach 821 Million units in 2012 and will rise to 1.2 Billion in 2013 [1]. Inevitably the paradigm for access information and internet services is increasingly migrating to mobile. Context-aware services can help to improve the user experience because they can adapt themselves to the users' context but, despite the improvements in terms of hardware, the the main obstacle towards a widespread adoption of such services is the limited battery life. A context-aware service requires the installation of a Context-Broker Application, which generates a continuous flow of data from the smartphone and a constant usage of its equipped sensors: as a consequence the considerable increase of energy consumption becomes a problem.
Aim. The aim of this work is to propose *gLCB* an Energy Efficient Context-Aware middleware for Android OS, which is able to collect Context Information and to send it to a remote platform in an energy-efficient way. The *gLCB* application proposes a new energy-aware data collection based on user profiles.
Methods. We define policies based on battery consumption profiles, which are selected depending on modifications of the context information. Moreover, we have implemented an automatic consumption testing mechanism and a statistical treatment of results to provide a reliable validation of *gLCB* in terms of energy efficiency.

Email addresses: luca.ardito@polito.it (Luca Ardito), marco.torchiano@polito.it (Marco Torchiano), marco.marengo@telecomitalia.it (Marco Marengo), falcarin@uel.ac.uk (Paolo Falcarin)

Post-Print draft published in Sustainable Computing, Elsevier 2013 Vol. 3 pp. 18-26 ISSN 2210-5379

Results. Experimental results show that our middleware got the best trade-off between number of server uploads and battery lifetime with the policies computed automatically by the device. This means that the way in which software is written can impact the energy consumption of a mobile device and service adaptation can be based on the actual value of the battery charge.

Keywords: Context-Awareness, Energy Awareness, Energy Aware Software, Android, gLCB, Local Context Broker

---

1. Introduction

The increasing proliferation of mobile devices and the intention of defining universal standards related to the mobile market, motivate many companies to implement context-aware standards, with the purpose of stimulating a rapid and wide adoption of a variety of useful applications.

A context-aware system has to be able to combine contextual information, which is related to the bounded environment. This info, called context data, is any information that can be used to characterize a specific entity situation [2].In this way it is possible to describe the actual situation, by determining some automatic behavioural variations or by notifying the user about some specific event. This kind of system has to be constantly in execution to gather raw data and to execute different types of operations based on context reasoning.

Context-Aware services collect contextual information automatically. With a wide range of possible user situations, it is important that services have a way to adapt appropriately to best support low battery scenarios. A system is context-aware whether it uses the context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

Many approaches are not completely dynamic, flexible or effective when we need to automatically match battery consumption requirements in differing contexts. A Context and Energy Aware system has to be flexible and able to react to environment variations.

Many features of modern devices like high processor speed, more efficient displays, more powerful data storage and WiFi/GPRS/UMTS network adapters, and specific hardware to enable advanced 3D graphics, considerably influence the device energy costs. Current approaches are not able to

implement energy-aware self-adaptation because they do not consider aspects related to context management policies.

Context can be used to find a lower energy consumption performance as well as implementing proper adaptation policies.

An energy aware context broker has to deal with:

- Sensing: to detect as much contextual information as possible,
- Transmission: to send gathered information to a context platform (for further processing),
- Adaptation: to manage the energy cost caused by *sensing* and *transmission* phases.

The context broker should be able to collect (and to send) context data only when really needed to avoid useless duplicated data management. It is also fundamental to include the battery charge state in the optimization strategy the context broker is adopting: the context broker shall be able to manage the information granularity depending on the battery level.

It is also indispensable to respect the following fundamental principles [3] in order to reduce the energy consumption to the lowest possible level:

- less work requires less energy,
- event programming avoids polling,
- multi-core environment programming,
- periodic timer should be avoided and
- the system should be scalable.

A context-aware application has to be able to minimize its operations in situations where the device is running out of energy [4]. In order to cut the device runtime operations, it is important:

- to implement an efficient algorithm or to modify an existing one to reduce operations have to be carried out to a minimum,
- to improve the compilation process by introducing optimizations based on target processor and

- to prefer a compiled and optimized code instead of an interpreted one.

Nowadays, operating systems for mobile devices provide different APIs to check the battery state (e.g. battery charge percentage, battery technology and temperature). Software components can be notified whenever batteries state changes.

Energy Awareness is concerned with several aspects: the quantity of energy that is used, what this energy is used for, where it comes from, the resulting effects (e.g. environment impact, resources consumption). In addition it poses the problem of how to reduce the energy consumption and its collateral effects.

Our previous work [5] [6] demonstrated a relationship between software and energy consumption. Even if software does not consume energy directly, however it has a direct influence on the energy consumption of the hardware underneath. As a matter of fact applications and operating systems indicate how the information is processed and, consequently, drive the hardware behaviour. We think that writing energy efficient code in a mobile environment can be more appreciated by users because there is a direct effect on their mobile batteries lifetime.

The Energy-aware middleware we are going to introduce in this paper uses certain resources (i.e. GPS, Bluetooth) related to different operations that have a high impact on energy consumption.

gLCB implements features, which focus on reducing the energy consumption by avoiding the execution of redundant operations.

The rest of this paper is organized as follows. Section 2 describes the main characteristics of our Context-Awareness Platform. Section 3 describes characteristics of the architecture and the components of *gLCB*, section 4 describes our validation criteria, section 5 introduces results, section 6 includes related works and finally, Section 7 includes conclusions and future work.

2. Context Awareness Platform

gLCB is responsible of retrieving context information through device hardware sensors, and it is also responsible of delivering such data to a Context Awareness Platform (CAP) [7].

The CAP allows the collection of context data from users' devices. Context data can eventually be processed by other components of the platform and become high-level context information (e.g. GPS coordinates can be translated into a civil address). This process is called reasoning [8].

Figure 1 describes the functional architecture of our CAP. From the figure we can deduce how context is transformed from raw data into high-level information and how high-level information is exposed to external applications.

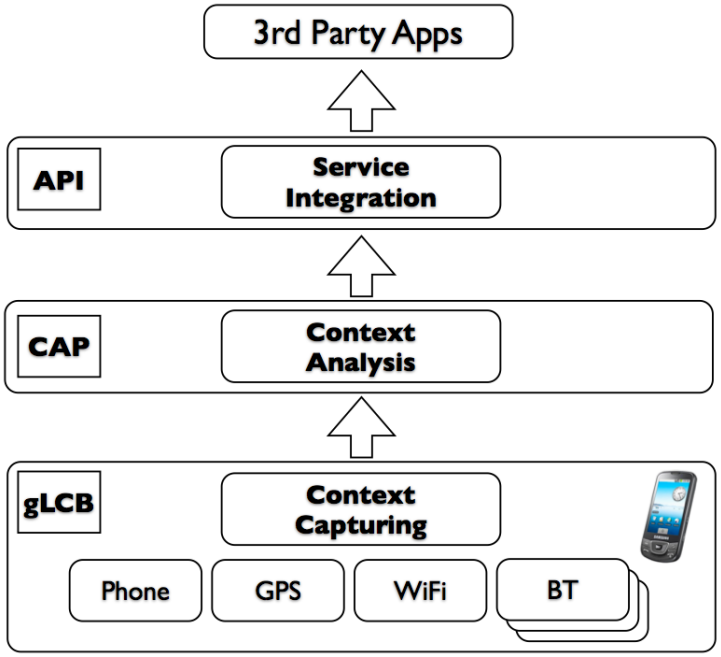

Figure 1: Context Awareness Platform: from raw data to high-level information

2.1. Context Capturing Layer

In this layer, raw context data are retrieved from the available device sensors (e.g. GPS, phone, Bluetooth, WiFi, etc) and aggregated.

The aggregated data are then asynchronously transferred to the CAP. This layer of abstraction collects context data in a fast and economic way, in order to integrate heterogeneous information sources and consequently supporting various protocols and different kind of data formats.

This is the layer where *gLCB* acts.

2.2. Context Analysis Layer

gLCB captures the low level data representations, which may not be meaningful to applications, and sends them to the Context Broker.

The CAP will modify this information in high-level representations which are easier to interpret and to use (e.g., an address is more significant that GPS coordinates). In context aware architectures, the reasoning component elaborates raw data and generates high-level information. The reasoning process may require significant computational effort so it is usually a server-side operation.

Besides reasoning, the CAP may also try to learn user behaviours: this technique is known as learning. Learning usually involves the analysis of a context history database and may be executed offline or online. Offline learning is a batch process, which runs periodically (daily, weekly, etc.) and is applied to the whole context history.

Online learning is executed at every context change and receives continuous feedback.

2.3. Service Integration Layer

This layer exposes context information towards the service platforms via API. The interfaces exposed by the CAP also allow third party applications to provide context data. As pictured in figure 2 the actors involved in the CAP are:

- Context source: a component, which feeds context data into the CAP. A source typically provides raw data (e.g. a mobile phone);
- Context consumer: a component which uses context information (e.g. an application);
- Context provider: a context source which is also capable of producing higher level information by acting as a consumer of raw data and a source of high level information;
- Context broker: the component which stores context data.

2.4. Context Capturing Layer

More than one context consumer and context source can access the platform to request and send context data. Specifically the context data are sent from applications executed in mobile devices. In this way, it is possible to perform data mining and clustering operations.

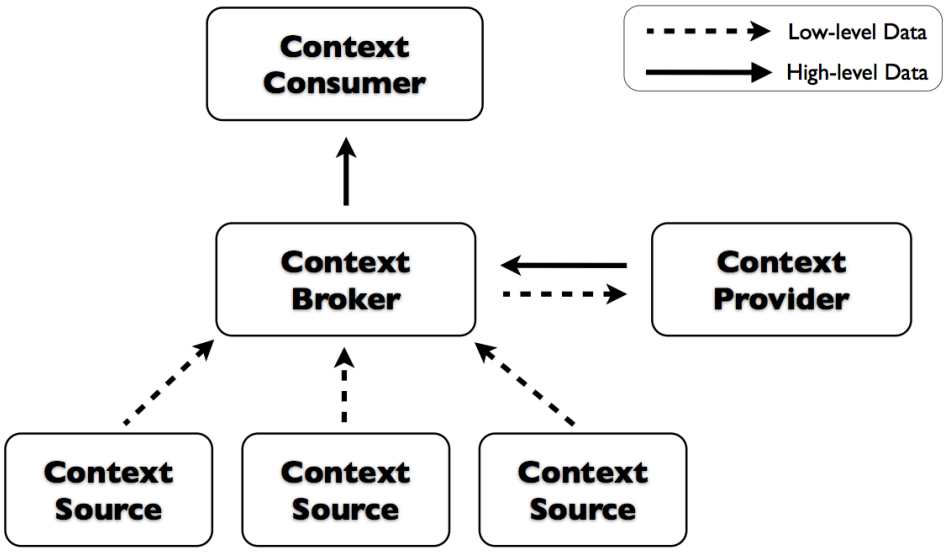

Figure 2: Context Awareness Platform: actors

Context Consumers can retrieve context through synchronous requests to the Context Broker or asynchronously by registering to context changes. This second scenario optimizes the allocation of resources and the network traffic: applications do not need timers to poll the CAP, the CAP triggers applications only whenever a meaningful context change takes place.

3. gLCB

The middleware software we developed for Android OS, is based on two levels: the first one is associated to the local context broker, and the second one to the sensors layer. The lower level establishes a communication channel to the higher level by sharing an interface. Mobile context-aware applications can rely on the local context broker to retrieve context data. To do so, the application is requested to bind to the LCB [9] service; when the binding is established, the application can perform context queries which will result in the retrieval of context data from the current device (if the requested information is available locally) or from the CAP. The local context broker manages its sensors via a Sensor Manager (Figure 2).

The Sensor Manager is responsible of:

- discovering new sensors,

- the management of sensors life cycle and,
- the management of their settings.

This component also exposes an interface, which allows the local context broker to perform one-shot requests to sensors or to subscribe to context data variations. Every sensor is installed as a service and runs in the background on the device.

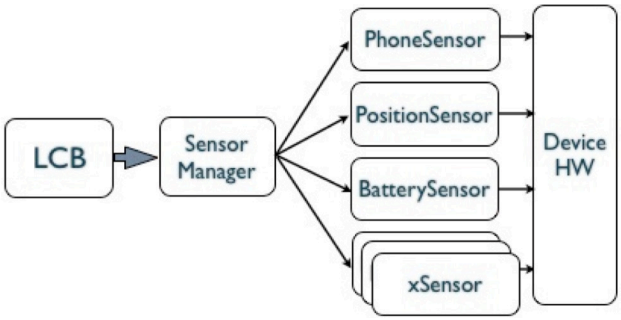

Figure 3: LCB sensors management

3.1. Critical aspects

We needed to guarantee a starting sensors mechanism in order to provide the independence of the components involved during the application execution. Independence is crucial as the number of developed sensors may vary and the test phase cannot depend on a newer release of *gLCB*. In this way the Android operating system is able to recognize each sensor as an independent application. The other critical aspect concerns the way in which data are obtained from sensors. A possible solution should be the sequential scanning of every sensor and the consequent data publishing in the server side. This approach introduces some critical problems mainly related to:

- High device battery consumption;
- Static data search and data publishing based on specific movement;
- Redundant context data transmission;
- A single monolithic application including every sensor.

3.2. Limited Search and Publishing

Context data search and publishing are the most relevant stages related to device battery consumption. Sensors that manage data provided by the operating system (e.g. IMEI number, ringtone volume etc.) do not reveal critical problems related to information management because such information are fixed. On the other hand, the scanning of new wireless networks, the nearby Bluetooth devices or the geographic position calculation, represents expensive operations in terms of energy consumption. These operations have to be limited whenever unnecessary.

3.3. Asynchronous Sensors Association

We created a sensor starting mechanism that was not dependent on the main application starting mechanism. In this way, new sensors are independent services associated with different starting components. Each sensor is characterized by a component called BroadcastReceiver. This component is an independent class that intercepts a specific system message called Intent. *gLCB* broadcasts an Intent whenever it is launched. Every sensor is able to intercept such intent and to react with a "Sensor Available" intent which notifies *gLCB* about its availability. In this way, it is possible to associate an undefined number of sensors with the main application. The only restriction is that they must be installed in advance on the device. Table 1 lists every available sensor.

3.4. Asynchronous Context Data Publishing

Another critical issue about power consumption is the redundant publication of unchanged context data. A periodical sequential scanning of all sensors will cause a waste of energy as every sensor is activated at each scan.

A possible solution to this problem is to implement a more efficient context scanning algorithm, e.g. an event-based algorithm which activates the sensors only in response to specific events. Table 2 lists every available sensor and the events that trigger the data update on the server side. All the sensors share two events: *Sensor Start* and *Out-of-date Context Data*. The first event means that the sensor collects and sends context data whenever the sensor is started. The second event means that after a certain period the sensor must refresh the context data because it is deleted by the CAP. This time period can be managed to increase or decrease the number of context updates regardless of data variation. To save energy we introduced a new parameter called *mindelta*. With this parameter the user can set, for each

Table 1: Sensors Description

Sensor	**Description**
WiFi	List of WiFi networks
DeviceActivity	Information about current applications running
Location	Geographical user position
DeviceStatus	Terminal publishing status
Phone	GSM or UMTS cell on which is connected
Bluetooth	Bluetooth neighbours
Data	Connectivity device info
Call	Call status

sensor, the minimum time period which must elapse between two updates. Mindelta must be lower than the data expire time to ensure the context data availability. Basically by managing *expire time* and *mindelta* parameters, the user can find a good trade-off between data processing and energy efficiency. In this way, the data stored on the server side is stable, updated, and available for Context Consumers.

3.5. Different Update Policies

We created seven user profiles: *VERY LOW* (Table 3), *LOW* (Table 4), *NORMAL* (Table 5), *HIGH* (Table 6), *AUTO*, and *CUSTOM*. Each one can change the gLCB behaviour in terms of context data searching and publishing by managing:

- the number of active sensors;
- the expire time parameter;
- the mindelta parameter.

The *LocationSensor* is more complex than the others and manages three further parameters, which trigger a context data update:

- enables or disables the GPS module;

Table 2: Sensors Events Description

Sensor	**Events triggering a data update**
WiFi	Sensor Start
	Out-of-date Context Data
	New WiFi Networks
DeviceActivity	Sensor Start
	Out-of-date Context Data
Location	Sensor Start
	Out-of-date Context Data
	Movement Greater Than Threshold
DeviceStatus	Sensor Start
	Out-of-date Context Data
	User Profile Change
Phone	Sensor Start
	Out-of-date context data
	New Cell Connection
Bluetooth	Sensor Start
	Out-of-date Context Data
	New Bluetooth Handsets
Data	Sensor Start
	Out-of-date Context Data
	Connectivity change (3G or WiFi)
Call	Sensor Start
	Out-of-date Context Data
	10 Calls (Incoming or Outgoing)

- computes the distance in meters between the actual value and the previous;
- computes the accuracy ratio between the actual value and the previous.

The AUTO profile, selects the best profile from *VERY LOW*, *LOW, NORMAL*, *HIGH*, profiles basing upon the actual battery level (see Table 7). By selecting the *CUSTOM* profile, the user can decide the updating policy for every sensor.

Sensor	State	Texp(s)	Mindelta(s)
Phone	ON	3600	180
Location	OFF	-	-
WiFi	OFF	-	-
Bluetooth	OFF	-	-
DeviceInfo	ON	3600	3600
DeviceStatus	ON	3600	1200
DeviceSettings	ON	3600	3600
DeviceActivity	OFF	-	-
DataSensor	ON	3600	1200

Table 3: VERY LOW user profile configuration

Sensor	State	Texp(s)	Mindelta(s)
Phone	ON	3000	180
Location	ON	3000	120
	D=5000m		
	A=10		
	GPS OFF		
WiFi	OFF	-	-
Bluetooth	OFF	-	-
DeviceInfo	ON	3600	3000
DeviceStatus	ON	3000	180
DeviceSettings	ON	3000	360
DeviceActivity	OFF	-	-
Data	ON	3600	180

Table 4: LOW user profile configuration

3.6. Application Execution

When started, *gLCB* loads the configurations used during the last execution. At the first start *gLCB* triggers the *HIGH* profile. For every sensor, it is possible to identify the update state, name, the time of the latest update attempt, and the outcome result. There are three possible outcomes:

- Update OK,
- KO.net (no publishing because of net error) and

Sensor	State	Texp(s)	Mindelta(s)
Phone	ON	1200	120
Location	ON	1200	120
	D=5000m		
	A=2		
	GPS ON		
WiFi	ON	1200	120
Bluetooth	ON	1200	300
DeviceInfo	ON	3600	300
DeviceStatus	ON	1200	120
DeviceSettins	ON	1200	120
DeviceActivity	ON	1200	120
DataSensor	ON	3600	120

Table 5: NORMAL user profile configuration

Sensor	State	Texp(s)	Mindelta(s)
Phone	ON	900	30
Location	ON	900	0
	D=250m		
	A=1		
	GPS ON		
WiFi	ON	900	120
Bluetooth	ON	900	120
DeviceInfo	ON	3600	120
DeviceStatus	ON	900	60
DeviceSettings	ON	900	60
DeviceActivity	ON	900	60
DataSensor	ON	3600	30

Table 6: HIGH user profile configuration

- KO.cb (no publishing because of context broker error).

After that, *gLCB* displays the actual selected profile and if it is selected manually or automatically, shows the number of successful publications, the date and time in which the application has been started.

Table 7: AUTO profile behaviour

Interval	**Profile selected**
Re-charger connected	HIGH
100% - 61%	HIGH
60% - 41%	NORMAL
40% - 16%	LOW
15% - 5%	VERY LOW
<5%	LCB Switched off

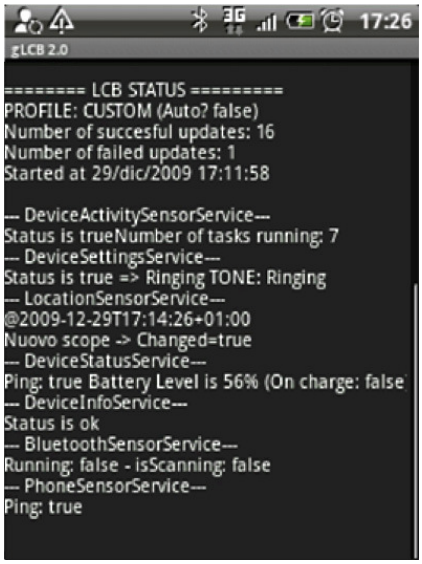

Figure 4: A screenshot of gLCB log activity

In figure 4, we can identify the user has 7 tasks in execution, the ringtone is enabled, new geographic position information was revealed, the battery level is at 56%, *DeviceInfo* and *PhoneSensor* sensors are enabled, and *BluetoothSensor* sensor is not enabled.

When the user switches off *gLCB*, a special context publication will be carried out to the server, which recognizes *gLCB* is switching off itself and cancels every context data related to that device. This avoids other Context

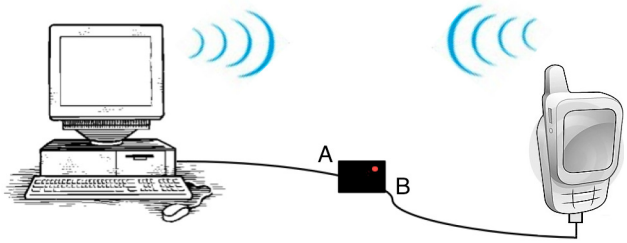

Figure 5: BatterySwitch

Consumers to read not valid data.

4. Validation Criteria

We chose an empirical approach to demonstrate the algorithm optimization effectiveness and we employed two different techniques to measure our middleware energy consumption. The first technique – "user side" – measures the time required by each profile to run out the phone battery (Time Measurements), while the second –"lab side" – aims at measuring the instant power consumption of each different user profile (Instant Power Measurements).

For both the approaches we scheduled the execution of a set of automated tests which did not require the user participation in order to get a better measurement precision.

The reference device is a Smartphone Samsung Galaxy (i7500) connected to the net through Telecom Italia Mobile (TIM) 3G network.

4.1. Time Measurements

The procedure we adopted to perform the battery duration measurement consists of the following steps:

- charge the battery until maximum battery level;
- select a specific *gLCB* user profile and let it start collecting data;
- record the time instants when the battery charge level changes;
- stop when the battery charge level reaches a predefined minimum value (5%);

Such a procedure lends itself to an easy automatic repetition of several measurement cycles. We carried out measures for each profile and each configuration automatically for thirty times in order to have statistical evidence.

In addition to the total discharge time, the above procedure, although in a quantitative way, can show how the battery behaviour changes. From the user perspective it is a very useful finding.

In order to conduct the above measurement procedure we built a dynamic battery charger, which allows starting or interrupting the battery charging through a specific command sent by a controlling PC.

We called this particular battery charger *BatterySwitch* and it has two usb ports as shown in figure 5:

- One port is connected to a pc (USB PORT A),
- The other port is connected to the mobile phone to manage the charge of the battery (USB PORT B).

BatterySwitch has a small firmware that can manage the supply of the USB port B, which will be connected to the mobile phone.

BatterySwitch is detected as a HID-interface once plugged into a pc and it is possible to interact with it through a simple application. For this reason it is needed a program that will:

- Be executed on the pc,
- Analyze the USB stack,
- Detect the device,
- Send bit '0' to start charging the battery,
- Send bit '1' to stop charging the battery.

This simple program has a thread listening to the port number 20248. The smartphone opens a socket to that port and it sends character 'a' or character 's'. The program will send respectively bit '0' or bit '1' to *BatterySwitch* to start or stop charging the device battery.

It is possible to set up a group of configurations that, at the end of each measurement, the device publishes the collected data to the server, and starts the next measure.

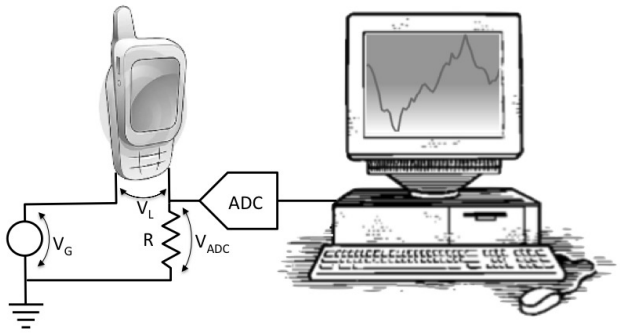

Figure 6: Circuit developed to get instant power consumption values

4.2. Instant Power Measurements

To measure the instant power consumption, we followed the procedure below:

- select a specific *gLCB* user profile and let it start collecting data;
- record the power consumption;
- stop after a predefined time (1 hour) has elapsed .

The device we used to measure power consumption is presented in figure 6. A power supply behaves like an ideal voltage generator with a constant 5 V tension. An analog to digital converter (ADC) connected to the PC reads the voltage drop across a resistor $R = 1\Omega$. The current flowing in the circuit can be computed by measuring the voltage drop on the resistor ($I = V_{\mathrm{ADC}}/R$). The instant power consumption value can be computed as:

$$P = V_L \cdot I = (V_G - V_{\mathrm{ADC}})\frac{V_{\mathrm{ADC}}}{R} = \frac{V_G V_{\mathrm{ADC}} - V_{\mathrm{ADC}}^2}{R}$$

The ADC sampling frequency is 49 MHz and each user profile is measured for 1 hour. Each measurement is repeated 30 times to get statistical evidence.

To ease the collection of all these sets of measures execution, we developed a simple scheduler on the mobile side, which runs *gLCB* with different user profiles.

5. Results

5.1. Time measurements

The complete battery discharge curves for each profile are presented in Figure 8. We can easily appreciate the non-linearity of the behaviour, which represents the main reason that led us to consider the total discharge time. In the figure, the rightmost abscissa reached by each profile represents the time required to achieve a 5% charge level: this is the time we consider as the total discharge time.

Table 8 reports, for each user profile, the total discharge time. The different profiles essentially differ for the frequency of context data updates transmitted by the gLCB to the CAP server, the second column presents the average number of updates per hour.

The average duration of our Samsung Galaxy battery in stand-by (without applications running) is 15 hours 33 minutes and 22 seconds. Since this value is expected to decrease according to the selected user profile, table 8 also reports the discharge variation in percentage with respect to the stand-by configuration (last column).

We can observe that as the number of publications increase, the mobile phone battery discharge time decreases linearly. The battery discharge time (t) is linked to the update frequency (f) by a linear relationship:

$$t = 13h : 28m : 14s - 39m : 33s \cdot f$$

The above equation captures the observed behaviour precisely ($R^2 = 97\%$). The offset of the equation does not correspond to the stand-by time since the execution of gLCB, even without updates, consume power. Also the profile labelled "AUTO" does not match the linear relationship because it dynamically adapts the update frequency to the battery charge level. Actually with an average update frequency of 7.54 we could have expected a discharge time of 8 hours and 30 minutes, which is shorter than the one achieved using the AUTO heuristic. We can also notice that the *LOW* profile needs to be reconfigured because its values are too close to the *VERYLOW* profile.

The energy consumption measures we carried out do not consider neither battery quality and temperature, nor which resources are used by *gLCB*. This is just an approximate view, but despite that we can see our expectations have been verified: different user profiles have different energy consumption patterns.

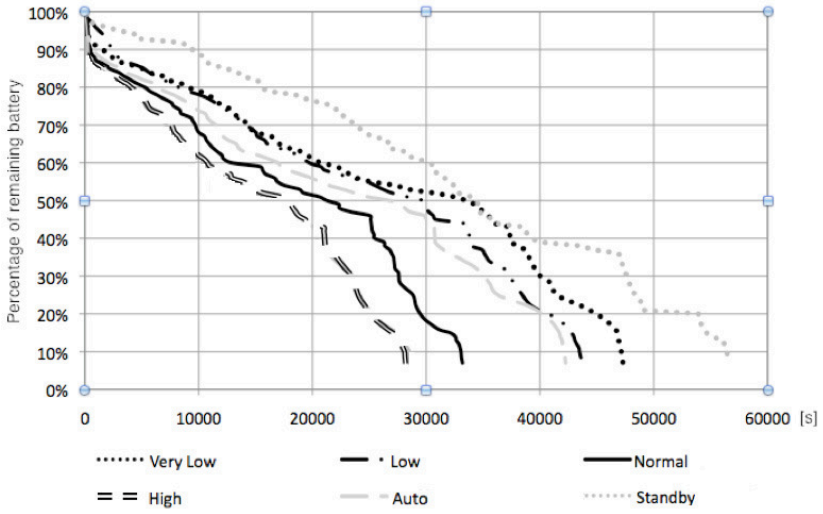

Figure 7: Discharge battery curves per user profile

5.2. Instant Power Measurement

The measurement with a fixed power supply allowed us to collect instant power figures during the operations of gLCB. Table 9 reports, for each profile, the mean instant power measured during the test time. In addition the last column shows the percentage of increment w.r.t. the stand-by profile.

The *AUTO* profile could not be measured because it adapts on the basis of the battery charge level, but in this configuration the power is provided by an external power supply and not by a battery.

Figure 9 contains the box plots of instant power measured, it shows the actual distribution of power consumption in each experimental run. To test whether the difference among the different profiles is statistically significant also in presence of the measured variance, we performed a set of statistical tests. We selected non-parametric tests due to the non-normal distribution of the data, in particular we applied the Kruskal-Wallis test to detect and overall difference among the profiles and the Mann-Whitney test for pair-wise comparisons.

According to the Kruskal-Wallis test there is evidence of a significant difference in terms of power consumption among the profiles (p-value < 0.001). More in detail, on a pair-wise comparison basis, we observed a significant

Table 8: Battery Duration and Updates per User Profile

Profile	Updates per hour	Discharge Time
STANDBY	-	15h 33min 22sec
VERYLOW	1,25	13h 8 min 8 sec
LOW	1,32	12h 7 min 35 sec
NORMAL	6,35	9h 13 min 6 sec
HIGH	8,48	7h 55 min 55 sec
AUTO	7,54	11h 41min 33sec

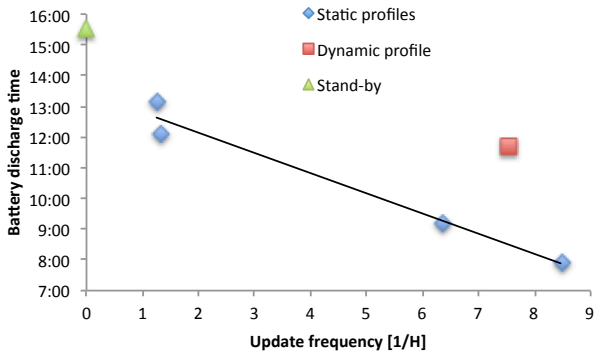

Figure 8: Update frequency vs. discharge time

difference between the following pairs: Normal and Low ($p < 0.001$), Low and Verylow ($p = 0.003$), and Verlow and Standby ($p < 0.001$). The magnitude of the difference, measured in terms of standardized effect size can be considered large. The only non statistically significant difference is between High and Normal.

6. Related Work

The energy issue is becoming very important in particular for mobile handsets [10]. In the literature can be found some approaches, which aim at reducing mobile applications power consumption based on context information. The context information retrieval is not the main feature of these

Table 9: Profiles average instant power consumption and variations based on User Profile

Profile	**Power**
STANDBY	473 [mW]
VERYLOW	522 [mW]
LOW	544 [mW]
NORMAL	594 [mW]
HIGH	617 [mW]

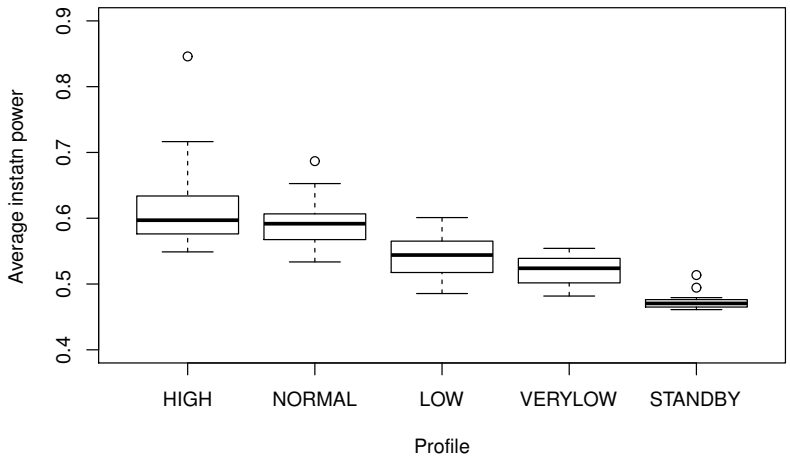

Figure 9: Box Plot of profiles average instant power consumption

applications but it is a side effect needed to implement the policies to save energy.

Flinn et al [11] show that there is a relationship between OS and some applications, which can be used to achieve some targets about battery life of laptops. They implemented the energy-aware adaptation as a trade off between dynamic balancing of energy conservation and quality of apps.Their approach is goal directed: the user can specify the battery life that he/she would like to achieve and the system adapts itself (display, processor speed, timeouts, etc.).

Ravi et al. [12] proposed a context-aware battery management: based on the historical data of locations in which the user recharged the battery. Their system predicts and notifies the user about the next possibility to recharge

the battery of the mobile phone based on his/her location.

Rahmati et al. [13] showed that an ideal selection of policies can more than double battery's life. These improvements may vary with the pattern of data transfer and the availability of WiFi. The key to achieve an improvement of battery life is obtained by accurately estimating the quality of the connection without turning on the WiFi network interface. Their approach is based on the complementarity of energy WiFi and cellular network profiles. Authors show that the cellular network requires less power to be always connected, but has a high cost per megabyte. The result of this project is a series of policies which can optimize data transfer based on the user location. This is possible by combining information from data history.

Lin et al. [14] based their experiments on the observation for which the accuracy of GPS position varies with the with the user's location. Their method automatically determines the accuracy needed mainly for mobile search-based apps.

Farrell et al. [15] assume that the GPS position update should take place within a specific region. They use a prediction mechanism that is based on the device velocity to determine the areas in which to exclude the GPS location because not necessary. Their work includes an interesting study on energy consumption patterns of triangulation WiFi, Bluetooth and GPS.

Kang et al. [16] present SeeMon: a scalable and efficient context monitor which works in environments with limited resources such as PDAs and cell phones. Their approach is based on the idea of removing unnecessary computations during the environment monitoring. SeeMon looks like a middle-tier framework of context-aware applications and a network of sensors; it exposes a set of APIs, which can be used concurrently by other applications. SeeMon also provides an efficient context-based system of queries with a strong semantic meaning, which allows an overhead (usually created by not necessary context) reduction. It is also introduced by authors the Essential Sensor Set concept that represents the minimum number of sensors need in a system to satisfy one query. Proper management of ESS allows for efficient use of device energy resources.

Boszormenyi et al. are focused on content adaptation [17]; a typical example of content adaptation is changing the service presentation depending on the context data. The data properties can be modified to adapt the service basing upon terminal capabilities, network capabilities and even user preferences.

Prior research related to the limited battery lifetime problem is mainly

focused on optimizing energy consumption at different levels e.g. hardware [18] and application layer [11], including compiler-based energy optimization [19]. Battery lifetime research has focused on analytical methods related to physical battery characteristics [20], [21].

To sum up the approaches mentioned before, differ from our approach mainly in aspects related to the energy consumption adaptation based on inferred situations [22]. *gLCB* collects a set of real-time information, which will be published on a context platform. The context platform provides context information for other applications via a set of APIs. So we needed to find solutions to identify the change of context as efficiently as possible. We propose some principles to determine how mobile devices should behave conforming to context information that can be used to infer energy consumption policies. gLCB recommends adaptation when context information analysis suggests to apply changes.

7. Conclusions

The main goal of context-aware systems is to provide relevant information, and/or services, based on current user context. In this paper we analysed the energy consumption behaviour of *gLCB*: a context-aware middleware, which runs in background in Android OS based mobile phones, and sends context information to a remote platform.

We described the architecture of *gLCB*, which is designed to adapt its behaviour on the basis of the remaining battery information.

We analysed some principles based on Energy-Aware software to determine how mobile devices should behave according to scarce or plentiful energy, and how context information can be used to infer energy consumption policies. These aspects are important to improve the efficiency of context-aware systems in terms of energy consumption.

The energy consumption analysis involved two different empirical experiments: in the first one we measured the average time employed to run out the mobile device battery in each user profile, while in the second one we measured the average instant power consumed by each user profile.

Since the behaviour of a mobile terminal in motion is not predictable because the network signal received is not stable, we consider average values and we repeated each experiment 30 times. In this way we normalized the data and we got statistical evidence.

Considering the results we obtained, we provided information related to:

- the battery average discharge-time,
- the average power consumption,
- the number of context updates per hour

of each user profile.

These results have been associated with the battery duration in standby conditions, as reference point to compare how efficient is the proposed approach. Time measurements show that AUTO profile provides the best performance between energy consumed and data uploaded.

As a first result of the time measurements we are planning to reconfigure the LOW Profile because there are big differences, in terms of average battery duration, average power consumption, and context updates per hour, from VERYLOW and NORMAL profiles. Then we can see that the impact of software in power consumption is real and a self-adaptive behaviour can help increasing battery life of mobile device. Even if the battery discharge measure needs a very complex system, we use these results only in a qualitative way. We gathered the instant power consumption value to get a more detailed view about the device energy consumption.

This deeper analysis, followed by a statistical treatment of the data collected, highlights a significant difference in terms of power consumption among the profiles. Other information such as CPU usage percentage, battery temperature, technology and voltage, could be used to evaluate how long and how many resources the application uses. It could be possible to improve even more the efficiency of *gLCB* by considering single resources usage.

Thus software is responsible (although indirectly) of variations in power consumption of a mobile device. These results suggest us to plan other experiments, which aim at generalizing a model which predicts software power consumption.

8. References

References

[1] R. Van der Meulen, Gartner says 821 million smart devices will be purchased worldwide in 2012; sales to rise to 1.2 billion in 2013, 2012.

[2] G. Abowd, A. Dey, P. Brown, N. Davies, M. Smith, P. Steggles, Towards a better understanding of context and context-awareness, in: H.-W. Gellersen (Ed.), Handheld and Ubiquitous Computing, volume 1707 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 1999, pp. 304–307.

[3] G. Kaefer, 2009. Green SW Engineering: Ideas for Including Energy Efficiency into your Software Projects, post Conference ICSE 2009.

[4] J. Lorch, A. Smith, Software strategies for portable computer energy management, Personal Communications, IEEE 5 (1998) 60 –73.

[5] A. Vetro', L. Ardito, M. Morisio, G. Procaccianti, Monitoring it power consumption in a research center: Seven facts, IARIA, 2011, pp. 64–69.

[6] G. Procaccianti, A. Vetro, L. Ardito, M. Morisio, Profiling power consumption on desktop computer systems, in: D. Kranzlmller, A. Toja (Eds.), Information and Communication on Technology for the Fight against Global Warming, volume 6868 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2011, pp. 110–123.

[7] P. Falcarin, M. Valla, J. Yu, C. Licciardi, C. Frà, L. Lamorte, Context data management: an architectural framework for context-aware services, Service Oriented Computing and Applications (2012) 1–18.

[8] B. Beamon, M. Kumar, Hycore: Towards a generalized hierarchical hybrid context reasoning engine, in: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on, pp. 30 –36.

[9] L. Lamorte, C. A. Licciardi, M. Marengo, A. Salmeri, P. Mohr, G. Raffa, L. Roffia, M. Pettinari, S. T. Cinotti, A platform for enabling context aware telecommunication services, Third Workshop on Context Awareness for Proactive Systems (2007).

[10] J. F. Mejia Bernal, L. Ardito, M. Morisio, P. Falcarin, Towards an efficient context-aware system: Problems and suggestions to reduce energy consumption in mobile devices, IEEE Computer Society Press, 2010, pp. 510–514.

[11] J. Flinn, M. Satyanarayanan, Energy-aware adaptation for mobile applications, SIGOPS Oper. Syst. Rev. 34 (2000) 13–14.

[12] N. Ravi, J. Scott, L. Han, L. Iftode, Context-aware battery management for mobile phones, in: Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on, pp. 224 –233.

[13] A. Rahmati, L. Zhong, Context-for-wireless: context-sensitive energy-efficient wireless data transfer, in: Proceedings of the 5th international conference on Mobile systems, applications and services, MobiSys '07, ACM, New York, NY, USA, 2007, pp. 165–178.

[14] K. Lin, A. Kansal, D. Lymberopoulos, F. Zhao, Energy-accuracy trade-off for continuous mobile device location, in: Proceedings of the 8th international conference on Mobile systems, applications, and services, MobiSys '10, ACM, New York, NY, USA, 2010, pp. 285–298.

[15] T. Farrell, R. Cheng, K. Rothermel, Energy-efficient monitoring of mobile objects with uncertainty-aware tolerances, in: Proceedings of the 11th International Database Engineering and Applications Symposium, IEEE Computer Society, Washington, DC, USA, 2007, pp. 129–140.

[16] S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, J. Song, Seemon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments, in: Proceeding of the 6th international conference on Mobile systems, applications, and services, MobiSys '08, ACM, New York, NY, USA, 2008, pp. 267–280.

[17] L. Boszormenyi, H. Hellwagner, H. Kosch, M. Libsie, S. Podlipnig, Metadata driven adaptation in the admits project, Signal Processing: Image Communication 18 (2003) 749 – 766. ¡ce:title¿Special Issue on Multimedia Adaptation¡/ce:title¿.

[18] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, H.-I. Yang, The case for cyber foraging, in: Proceedings of the 10th workshop on ACM SIGOPS European workshop, EW 10, ACM, New York, NY, USA, 2002, pp. 87–92.

[19] T. Heath, E. Pinheiro, J. Hom, U. Kremer, R. Bianchini, Code transformations for energy-efficient device management, Computers, IEEE Transactions on 53 (2004) 974 – 987.

[20] D. Panigrahi T, D. Panigrahi, C. Chiasserini, S. Dey, R. Rao, A. Raghunathan, K. Lahiri, Battery life estimation of mobile embedded systems, in: VLSI Design, 2001. Fourteenth International Conference on, pp. 57 –63.

[21] P. Rong, M. Pedram, An analytical model for predicting the remaining battery capacity of lithium-ion batteries, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 14 (2006) 441 –451.

[22] L.-W. Goix, M. Valla, L. Cerami, P. Falcarin, Situation inference for mobile users: A rule based approach, in: Mobile Data Management, 2007 International Conference on, pp. 299 –303.