Chaib Draa *et al. EURASIP Journal on Embedded Systems* (2017) 2017:4
DOI 10.1186/s13639-016-0036-8

EURASIP Journal on Embedded Systems

RESEARCH Open Access

Sensing user context and habits for run-time energy optimization

Ismat Chaib Draa[1*], Smail Niar[1], Jamel Tayeb[2], Emmanuelle Grislin[1] and Mikael Desertot[1]

Abstract

Optimizing energy consumption in modern mobile handheld devices plays a very important role as lowering energy consumption impacts battery life and system reliability. With next-generation smartphones and tablets, the number of sensors and communication tools will increase and more and more communication interfaces and protocols such as Wi-Fi, Bluetooth, GPRS, UMTS, and LTE will be incorporated. Consequently, the fraction of energy consumed by these components will be larger. Nevertheless, the use of the large amount of data from the different sensors can be beneficial to detect the changing user context, to understand habits, and to detect running application needs. All these information, when used properly, may lead to an efficient energy consumption control.

This paper proposes a tool to analyze user/application interaction to understand how the different hardware components are used at run-time and optimize them. The idea here is to use machine learning methods to identify and classify user behaviors and habit information. Using this tool, a software has been developed to control at run-time system component activities that have high impacts on the energy consumption. The tool allows also to predict future applications usages. By this way, screen brightness, CPU frequency, Wi-Fi connectivity, and playback sound level can be optimized while meeting the applications and the user requirements. Our experimental results show that the proposed solution can lower the energy consumption by up to 30 % versus the out-of-the-box power governor, while maintaining a negligible system overhead.

Keywords: Energy consumption, Run-time user and application analysis, Device's context, Applications sequences prediction

1 Introduction

Mobile and communicating devices became essential tools in our personal and professional activities. In recent years, their number and their applications have largely increased. In our modern societies, each person has several handheld devices (smartphone, tablet, portable PC, etc.). By the end of 2013, 6 % of the global population owned a tablet, 20 % owned portable PCs, and 22 % owned smartphones.[1] It is predicted that by 2017, 65 % of the US population will own a smartphone.

Next-generation mobile systems will include a large number of cores, a powerful GPU, large caches, memory capacity, and a variety of I/O tools and communication protocols. For instance, the Samsung Galaxy S6 launched in 2015 contains three times more sensors than the Samsung Galaxy S marketed in 2010. In the same time, the number of cores has also increased from 1 to 8.[2]

Consequently, on one side, next mobile system generations will contain more powerful components and on the other side, applications running on these devices will become more complex. As a result, the needs of new applications in terms of computing power, communication, and storage have significantly exceeded the capacity of the batteries. For this reason, new energy consumption management systems are needed.

Most of the existing technics for energy saving take into account neither the user individual profiles nor the changing application needs. Our proposal is to capture, store, and process such information using the computing power and the various sensors to reduce energy consumption. The key for our power saving technique is therefore to leverage users context, behaviors, and habits to predict the

*Correspondence: ismat.chaibdraa@univ-valenciennes.com
[1] University of Valenciennes, Valenciennes, France
Full list of author information is available at the end of the article

running applications and improve upon the default energy management policies of the OS. The main contributions of this work can be summarized as follows.

1. We exploit rich sensor hubs to collect and explore a large set of data to search context usage patterns in the device use. We also use metrics to gauge user needs and characterize his/her habits. The identification of the context and the user's associated actions allow us to decrease the energy dedicated to unused resources in some cases.
2. We propose a new classification and characterization method of the launched applications to find frequent sequences of application runs. On this basis, we can predict which application will probably run next. With the developed prediction and the knowledge of each application needs, we are able to adjust the provided resources and to perform optimizations such as dynamic voltage frequency scaling (DVFS) [1], data prefetching, and device management without impacting negatively the user satisfaction. Such actions decrease the energy consumption of the whole system.

The global architecture of our approach is shown in Fig. 1. This figure presents an abstraction of the several key stages of our approach. The first stage consists in collecting data about the user behavior and the device's context, such as running applications, the background processes, the device's position, the ambient luminosity, the datem and the time. These data are used in the second stage through three mechanisms:

- Off-line classification of applications in terms of resources
- Application prediction mechanism
- On-line device's context identification

In the third stage, the dynamic *optimizer actuator* uses the outputs of the second stage to perform actions such as device management and applications scheduling in order to reduce the energy consumption. We have a global framework which contains two main components:

- Context-based optimization component (COC): based on the device context and the sensory data
- User needs-based optimization component (UNOC): based on user actions and application classification.

The rest of this paper is organized as follows. In Section 2, the architecture of the framework containing the COC and UNOC is presented. In Section 3, we explain the COC. In Section 4, we present in details the UNOC with the classification and prediction mechanisms. In Section 5, we present the experimental results. Section 6 presents the related work, and finally, in Section 7, a conclusion and some perspectives are given.

2 Framework architecture for energy consumption optimization

In order to obtain the provided objectives in the previous section, we designed a framework to optimize the energy consumption in mobile systems and to improve the energy management provided by the OS. This framework consists of two components, the COC and the UNOC. They use different sensors/data and run in parallel. However, depending on the type of used device (smartphone, tablet, ultrabook, etc.) some functionalities in COC or UNOC can be more suitable. The proposed screen brightness management that depends on the device position is more appropriate to laptops or fixed ultrabook. All the rest of optimizations, screen brightness management using the ambient luminosity, the microphone level management based on the ambient noise, and the user needs/habits-based component, can be exploited in all mobile platforms with an OS and a user layer. Consequently, smartphones, laptops, and tablets can benefit of these mechanisms. The functional architecture of the proposed components are given in Fig. 2 for COC and in Fig. 3 for UNOC. These components implement the abstract architecture presented in Fig. 1.

2.1 Context-based optimization component (COC)

COC is responsible of collecting data from the embedded sensors, device position, ambient luminosity, and ambient noise, etc. These data compose the device context taken into account to apply different policies on the screen brightness and the speaker sound level as shown in Fig. 2. COC works in two phases:

1. Device context analysis: in this phase, we realize some preliminary experiments to analyze and to determine the device context. Among the available sensors, we select the most relevant sensors which provide information about the device position and the ambient noise. The results are used to develop the second phase.
2. Embedded software: in this phase, we control during run-time the system component's activities. Depending on the context, these components may have a high impact on the consumed energy. The control is done by exploiting information obtained from the embedded sensors.

The main idea is that in embedded and mobile systems, it is possible to save energy and reduce power consumption by taking the context information into account. It could be attained by monitoring sensors

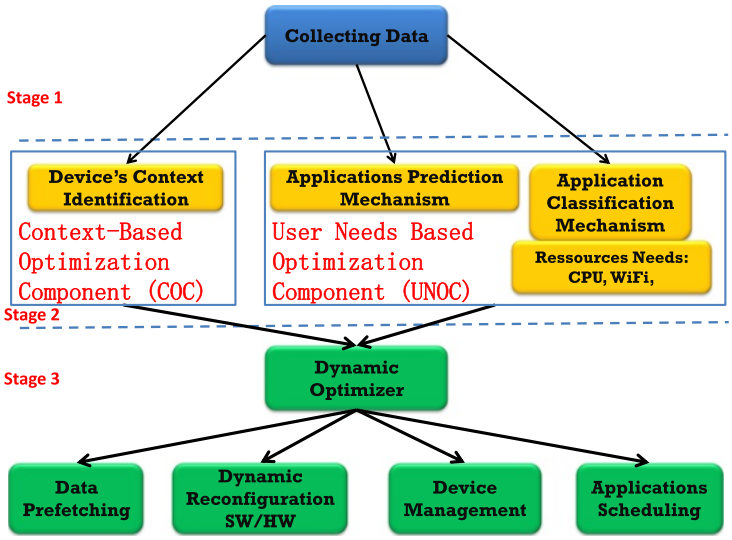

Fig. 1 Overview of the proposed approach. A generalization of the several stages of the proposed approach

that exist in mobile devices. Sensors' data are processed and correlated to possible power consumption reduction opportunities.

2.2 User needs-based optimization component (UNOC)

This component is developed to take into account the user needs and habits in the energy consumption optimization. Its structure is generic and is shown in Fig. 3. UNOC is implemented in five steps as follows:

1. *Data collecting mechanism*: user behavior and system usage information are collected.
2. Processing the collected data through the analyzer to guarantee user privacy by anonymizing the data.
3. Storing the collected data in a data base.
4. This step is implemented in two phases:

   (a) Uploading the collected data to the back-end component in order to be processed via

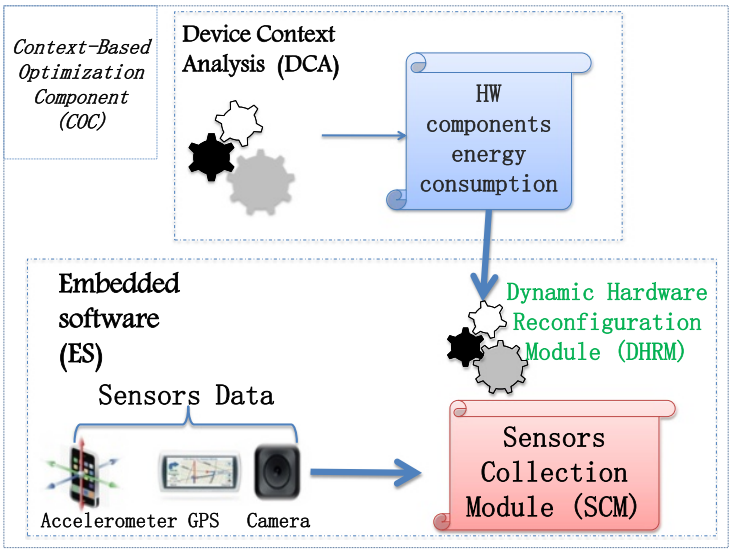

Fig. 2 Context-based optimization component architecture. Architecture of the first component of our framework based on the context

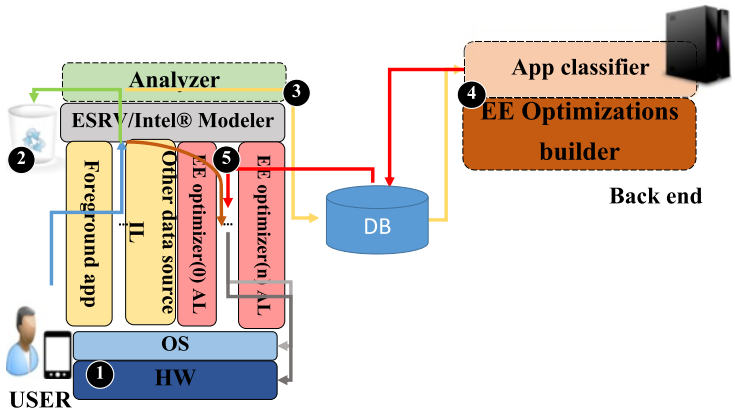

Fig. 3 User needs-based optimization component architecture. Architecture of the second component of our framework based on the user needs and habits

mechanisms like *application classification*, *applications prediction*, and user behavior profiling.

(b) Building optimization rules.

5. Pushing the obtained rules to the optimizer actuator in order to implement a specific optimization for each hardware component.

In this paper, we focus mainly on the data collecting mechanism, the application classification, the application prediction, and the optimization mechanism as follows.

1. The data collecting mechanism: it represents the first step mentioned previously. In this phase, we collect a large set of data which are as follows: running applications, date, time, elapsed time of each application, and background process.
2. This phase is composed of:
   - The off-line application classification in terms of Wi-Fi and CPU needs. In the current version of the UNOC, we focus on two components: the Wi-Fi and CPU. These two units are among the most power consuming components in mobile system but the classification can be extended to screen brightness, microphone, GPS, etc.
   - The execution average time for each application is calculated, and this phase is also off-line but the data base can be updated in a weekly basis.
   - In-line application prediction mechanism.
3. All these phases are combined and used by the optimizer actuator which manages Wi-Fi connection and CPU frequency in order to optimize the energy consumption. The next section presents the COC in details.

3 Context-based Optimization Component (COC)

A crucial aspect of energy management is having a good understanding of how, when, and where users interact with their handset and how they demand resources such as luminosity, sound level, high consumption, connectivity, etc. COC relies on the device context and user actions, which are context driven by nature. The device's context is defined by its position, the ambient light and the ambient noise. The screen brightness and the speaker sound level (respectively) are controlled by the device's position (normal or abnormal, ambient luminosity and the ambient noise (respectively)). To do so, policies are applies to sensory data to impact power consumption. The *Sensors Collection Module (SCM)* and *Dynamic Hardware Reconfiguration Module (DHRM)* were developed to achieve the COC work. The following two sections explain how the SCM and DHRM are used for brightness and sound managements.

3.1 Brightness management depending on device's position

3.1.1 Sensors Collection Module (SCM)

This module is responsible for collecting data from the embedded sensors in order to identify the most appropriate device's stand. In our mobile handset, there are several sensors such as accelerometer, ambient light sensor (ALS), simple orientation sensor, inclinometer, compass, gyrometer, and geolocation. In order to determine which collected sensors are the most relevant, some preliminary experiments have been achieved. First, we collect the sensor values in several device's position (normal standing, inclined, jostled, etc.). We compare sensors' readings for various device positions in order to pick the most relevant. The sensor values which have a large gap in different positions are ignored. The available sensors are:

- Ambient luminosity: ambient light sensor (ALS).
- Orientation: inclinometer, compass.
- Motion: gyroscope, accelerometer.
- Location: GPS
- Ambient noise: microphone

We select the following sensors:

1. For ambient luminosity and ambient noise: we use ALS and microphone because these are the only sensors that provide us these information.
2. For the orientation, we have chosen *inclinometer* because the obtained data from this sensor are more informative and compass data are changing due to magnetic strength.
3. For motion, both of accelerometer and gyroscope have three-dimensional metrics on axes x, y, and z. In the following example, we compare standard deviations:

   - Accelerometer $(x, y, z) = (0.57, 0.37, 0.52)$
   - Gyroscope $(x, y, z) = (89.42, 57.80, 54.95)$

   Normalized variation indicates sensitivity. More sensitive values are more informative. This comparison prompted us to choose gyroscope for motion.
4. We exclude location-based data collection because it is private information (PI).

The SCM is calibrated by sensory data collected while the device is in standing position. After calibration, sensors data is collected in real time. This way, if the device is tilted, its inclination data is immediately updated in the SCM. Finally, data are injected in the memory in order to be consumed by the Dynamic Hardware Reconfiguration Module (DHRM). This reconfiguration is continuous and carried out in the background. Whenever a sensor value changes, SCM takes it into account. It updates the new value and upgrades it through a shared memory, then DHRM performs optimizations on the screen brightness level. Figure 4 presents the HW and logic sensors, and Fig. 5 presents the SCM process.

3.1.2 Dynamic hardware reconfiguration module (DHRM)

This module is responsible to manage the hardware components depending on the device's position as mentioned in Section 3.1.1. Available data in the shared volatile memory is imported and taken into account by this module in order to apply business logic decision mechanisms with the values. The module handles the LCD driver of the device to manage screen brightness as shown in Fig. 6.

The DHRM compares the new captured sensor's values with the normal stand values. Then, according to this comparison, the DHRM adjusts the screen brightness to the most suitable level for the user. For example, if the gyroscope sensor value exceeds the range of allowed values, the module applies a specific optimization on the screen's brightness by decreasing it. The ambient luminosity is also taken into account to adjust luminosity. When the environment is too bright, the screen brightness is increased and vice versa. For brightness management, the power reduction opportunity is about 30 % between the max screen brightness and the min screen brightness.

The DHRM relies on the data captured by the SCM and selects four stand device's state. Here is an example for each state:

- Hard-to-watch: device shake is too important to watch it correctly.
- Mild-motion: from small movement to mild ones like when playing game.
- Normal-stand: device left in the same position for a moment.
- Abnormal-tilted: set device in $\pm 90^\circ$ on x-/y-axis with no motion.

Each state is recognized through sensors metrics. DHRM sets the corresponding screen brightness, according to the identified state, as shown in Table 1.

When the device is in normal stand, we take into account the ambient luminosity for brightness adjustment as shown in Fig. 7.

3.2 Sound management based on ambient noise

Another use case similar to Section 3.1 was achieved in order to manage the device's speaker level depending on the ambient noise. As in the case of brightness management, we have two main modules. The first one is the *Ambient Noise Collector* (ANC) and is measuring the ambient noise and shares its to the second module *Sound Control Module* (SCM) which will adapt the speaker level accordingly.

For example, in this scenario when the ambient noise is high , the SCM increases the speaker's sound level. On the other hand, if the ambient noise is at average or low, the module decreases the speaker's sound level.

Contrary to the first component, we do not store the values of the ambient noise, and we act dynamically on the speaker's volume. The sound level is modified gradually to avoid any impact on the user satisfaction, on the base of the change blindness [2].

4 User needs-based optimization component (UNOC)

4.1 Classification mechanism

In this paper, the classification is achieved off-line and used during the optimization phase as mentioned above.

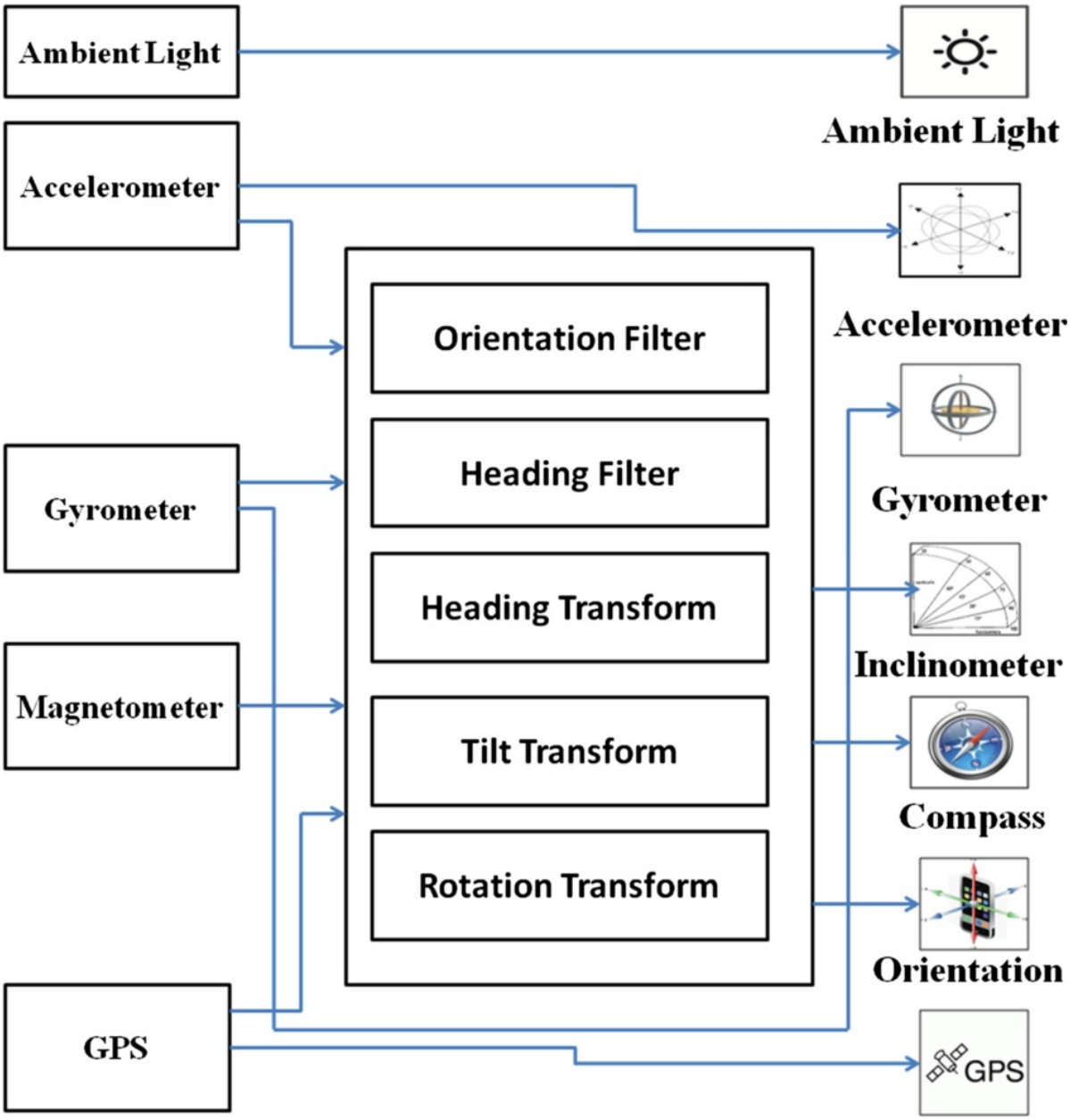

Fig. 4 Available sensors on our mobile device. This figure shows the available HW and logic sensors

Applications are classified according to their Wi-Fi and CPU usage. For the Wi-Fi, the classification is binary (Wi-Fi on/off). For the CPU, the classification is based on upper frequency thresholds. In both cases and before adjusting any resources, the optimizer actuator consults the list of background processes to avoid any conflict.

4.1.1 Classification in terms of Wi-Fi

The aim of this classification is to contribute to the management of the wireless interface according to the needs of the running applications. To achieve the off-line classification, we realized some preliminary experiments. At first, the internet rate is estimated by the sum of the upload and download rate, when no application and background process are running. A low rate threshold was fixed at 10 KB because of the connectivity management in Microsoft Windows operating system that achieves some connection rate tests, even when no application needs connection. Secondly, we run the applications we want to classify individually and acquire bandwidth use. When the

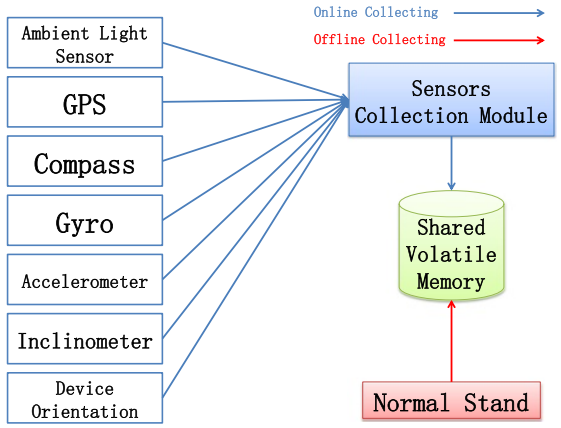

Fig. 5 Sensors information collecting. This figure illustrates the sensors values collecting by SCM

sum of the upload and download rates during the execution of the application is under the 10-KB threshold, we assume that no wireless connection is required (and vice versa). Table 2 shows the classification results for three examples of applications.

On this base, the on-line optimization is carried out: the Wi-Fi need is evaluated according to the running applications classes. Obviously, the main point is to assess the complete requirement of the mobile device current state in order to avoid the user dissatisfaction when the wireless connection is disabled.

4.1.2 Classification in terms of CPU need

Windows 8.1 manages the CPU frequency automatically. However, in some cases, the computing resources provided by the OS exceed what is required by the running applications and the user. To improve this management, we propose to classify the applications in terms of CPU frequency.

In the current implementation, we have arbitrarily defined three thresholds (800 MHz, 1.25 GHz, and 1.75 GHz) that define four classes:

1. Class c_1: applications requiring a *low* CPU frequency (<800 MHz). Text processing applications such as *Word*, *Excel*, or simple games such as *Imperial Sudoku* belong to this class.
2. Class c_2: applications requiring a *medium* CPU frequency (between 800 MHz and 1.25 GHz). Web browsers such as *Firefox* or *Google Chrome* are in this class.
3. Class c_3: applications requiring *high* computing resources (between 1.25 and 1.75 GHz). Advances games such as *2048* belong to this class.
4. Class c_4: applications requiring *very high* computing resources (over 1.75 GHz). Image processing and synthesis such as Image ray-tracing, simulation applications, and mathematical applications belong to this class. During our experiments, we found no applications belonging to this class.

To classify an application, the CPU utilization and frequency are measured during its execution. More precisely,

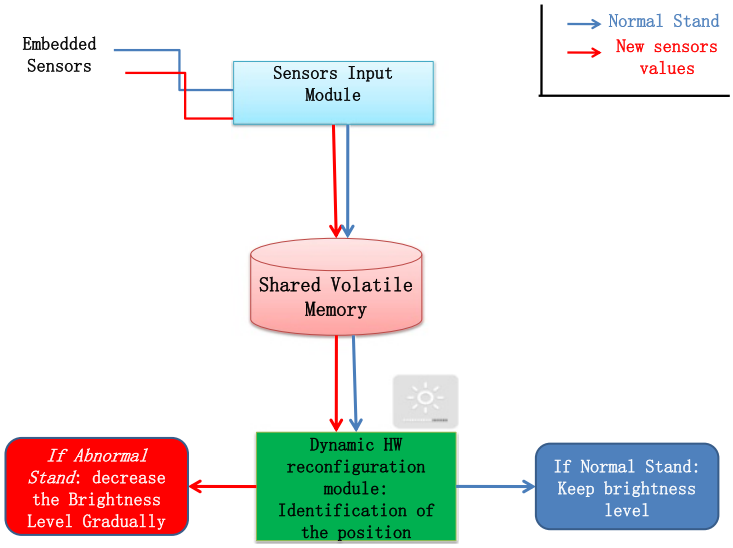

Fig. 6 Brightness management By DHRM. This figure illustrates the actions performed by DHRM (adjusting brightness)