Journal of Systems Architecture 97 (2019) 320–334

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

# ENOrMOUS: ENergy Optimization for MObile plateform using User needS

Ismat Chaib Draa[a,*], Smail Niar[a], Emmanuelle Grislin-Le Strugeon[a], Morteza Biglari-Abhari[b], Jamel Tayeb[c]

[a] CNRS, University Valenciennes, UMR 8201 - LAMIH, Valenciennes F-59313, France
[b] University of Auckland, New Zealand
[c] Intel Corporation, Portland, USA

ARTICLE INFO

*Keywords:*
Mobile systems
Mobile power consumption
Neural networks
Run-time analysis
Data mining algorithms

ABSTRACT

Optimizing energy consumption in modern mobile handled devices plays a crucial role as lowering the power consumption impacts battery life and system reliability. Recent mobile platforms have an increasing number of sensors and processing components. Added to the popularity of power-hungry applications, battery life in mobile devices is an important issue. However, the utilization pattern of large amount of data from the various sensors can be beneficial to detect the changing device context, the user needs and the running application requirements in terms of resources. When these information are used properly, an efficient control of power knobs can be implemented to reduce the energy consumption. This paper presents a framework for *ENergy Optimization for MObile platform using User needS* (ENOsMOUS). This framework is able to identify user contexts and to understand user habits, preferences and needs to improve the operating system power scheme. Machine Learning (ML) algorithms have been used to obtain an efficient trade-off between power consumption reduction opportunities and user satisfaction requirements. ENOrMOUS is a generic solution that manages the power knobs. When applied to the CPU frequency, the sound level, the screen brightness and the Wi-Fi, ENOrMOUS can lower the power consumption by up to 35% compared the out-of-the-box operating system power manager schemes with a negligible overhead.

## 1. Introduction

Mobile and communicating devices are drastically changing our professional and personal activities. The number of smartphones users in western Europe has been predicted to increase from 240.3 millions in 2016 to 279.6 millions in 2019.[1] The capabilities and hardware complexity of these handled devices are in constant improvement. They include a large number of cores, a powerful GPU, large caches and a significant number of embedded sensors. For instance, the Samsung Galaxy S7 launched in 2016 contains 10 sensors, representing 6 additional sensors in comparison to the Samsung Galaxy S marketed in 2010. The number of cores has also increased from 1 to 8 cores. In addition, applications running on current and future mobile devices are very power-hungry and more complex. As a result, the user and applications needs in terms of computing, communication and storage can deplete the device's battery in a few hours and impact user satisfaction. Nowadays, battery life has become one of the biggest obstacles for mobile device advancements. For these reasons new power consumption management systems are needed to extend battery life without impacting the processing power and user's experience.

Most of the existing energy saving techniques take into account neither the user individual profiles nor the changing application needs and user requirements. In this paper, our work is based on the user context information and habits. In the literature, many definitions of the context are given [1]. In [2], the context is defined by 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. Adapted from Dey [2], we define the user context as a set of information that characterize the situation in which the mobile device is used. The information may be divided in 3 subsets depending on their subject: the user, the time and space environment, and the device. This includes the information that may impact the configuration of a hardware resource such as ambient luminosity for the screen brightness, the applications needs in terms of CPU and so on. ENOrMOUS exploits the large set of embedded sensors to collect, store and process the user context information and application requirements at run-time. The collected data are then exploited to generate power policies. The purpose here, is to improve the default power management policies of the operating system and to increase the battery life. In the worst case scenario, ENOrMOUS offers a similar energy management to the operating system. In this paper, we assume that a mobile system can be exploited by many users and

* Corresponding author.
*E-mail address:* ismat.chaibdraa@univ-valenciennes.fr (I.C. Draa).

[1] www.statista.com/statistics/494554/smartphone-users-in-western-europe

https://doi.org/10.1016/j.sysarc.2018.10.004
Received 19 February 2018; Received in revised form 11 October 2018; Accepted 12 October 2018
Available online 13 October 2018

to differentiate between them, we identify each user by his/her specific session. Our contribution can be summarized as follows:

1. We exploit the rich sensor hubs and the operating system's application programming interfaces (APIs) to collect a large set of data relative to the user, the launched applications, the consumed resources and so one to represent the user context. This data collection phase is performed by a software module consisted of several probes. Each software probe corresponds to a specific kind of information such as environmental information, system information, etc. These probes are implemented as background process. All the collected information are stored in unified XML way to be processed in the next phase.
2. In the second phase, we propose a new user context classification based on the collected information of the previous phase. The collected information will be processed through machine learning algorithms to extract usage pattern and regularities, identify the user context and predict the user's associated actions. The employed machine learning techniques allow us to use the opportunities to decrease the energy consumed by unused resources in some cases. The classification tends to find a trade-off between the user satisfaction, the running application requirements and power consumption reduction opportunities.
3. In this last phase, the designed classification techniques are used to implement new and efficient power policies to control the power knobs and to reduce the whole system power consumption. Techniques such as Dynamic Frequency Scaling (DFS), Wi-Fi management, screen brightness and sound level adjustments, can thus be efficiently implemented.

The aim is to offer the users, with different profiles and needs, a customized management approach depending on their habits, requirements, lifestyle and job. In ENOrMOUS, the power savings are achieved at run-time and are transparent to the user. The goal here, is to minimize the user involvement.

The reminder of this paper is organized as follows. The ENOrMOUS functional architecture is presented in Section 2. In Section 3, we present the classifiers architecture. In Section 4, we present the experimental results obtained with our approach and a comparison with the OS power management is given. Section 5 presents the related work and finally we conclude and give some prospects of the project in Section 6.

## 2. ENOrMOUS functional architecture

Several software modules have been developed to implement the three phases described previously. All these modules are shown in Fig. 1. Details of these software modules are as follows:

- *Data Collection Module (DCM)*: this module encapsulates 3 probes.
  - User Probe (UP): this probe is linked to the launched applications and the user's preferences in terms of resources used during a certain period of time. The *UP* is executed at run-time and is transparent to the user. The probe indicates if the application is running in the background or foreground and its average running time in a given time period, as shown in Section 2. We assume that users have different behaviors in weekdays and weekends, and also in different periods of a given day. This probe retrieves also the user mobility and permits us to know when the user is walking, running or in a stationary state. We use a two-week period to collect the required information through UP. The user can change this information collection period.
  - Environment Probe (EP): this probe gathers information about the user's environment. It indicates: ambient luminosity, ambient noise, device stand, date and time and user position. These information are captured using available embedded sensors such as *Ambient Noise Sensor (ANS), Ambient Light Sensor (ALS), Accelerometeter, GPS* and so on.
  - System Probe (SP): this probe indicates the resources consumed by the system. The collected information are: average utilization of the CPU in percentage, the allocated CPU frequency by the OS, sound level, screen luminosity, Wi-Fi status, battery level and GPS state. This probe can also give information about the user's habits and needs in terms of system configuration preferences. For example, the CPU average utilization varies from a user to another depending on their needs.

The user context data for using each application is stored separately and later all contexts information are used to obtain the resource usage patterns as shown in Fig. 2.

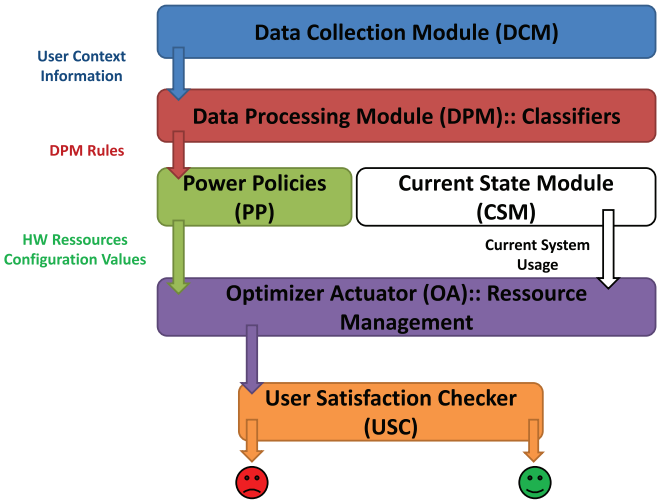


**Fig. 1.** ENOrMOUS abstract architecture.

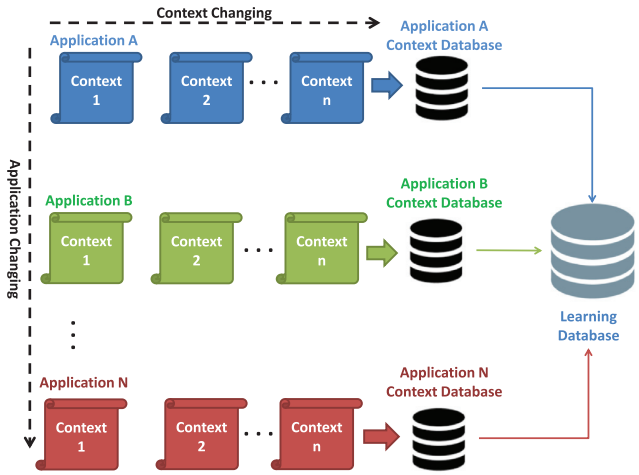


**Fig. 2.** Context changes depending on the foreground application.

DCM is the first executed module in ENOrMOUS and corresponds to the data collection phase. These data are captured and stored during two consecutive weeks in order to create a knowledge base that will be processed by the next module. DCM runs as background process and is executed when the mobile device is switched on. All these information are captured by interfacing the aforementioned probes with the embedded sensors and APIs offered by the operating system. The collected information are then stored in a data base in unified XML to be processed by the Data Processing module (DPM). Fig. 3 shows the DCM structure. As shown in the experimental section, the overheads due to DCM utilization is low. The **DCM** is launched only once, except when the user is not satisfied by the generated power policies. In this case, the data collection is redone to enrich the knowledge base as it will be detailed in the next sections.

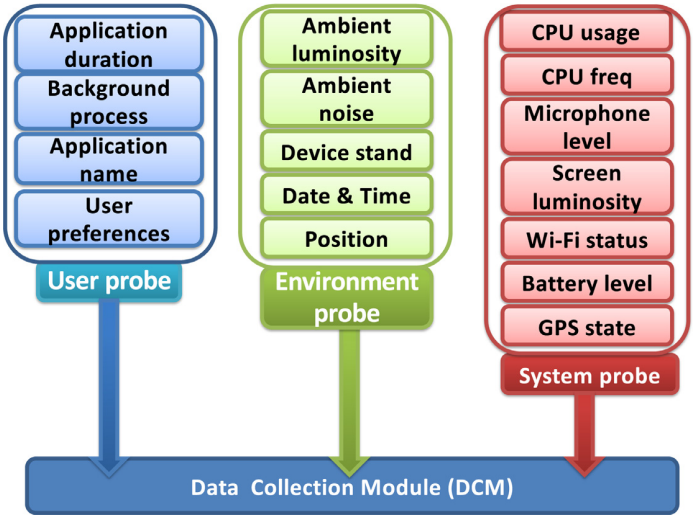


**Fig. 3.** DCM collecting user context.

- *Data Processing Module (DPM)*: this module is at the heart of our framework. It uses a large number of ad-hoc classifiers based on neural networks and data mining algorithms to process the stored DCM information and generate Power Policies (PP).
- *Power Policies (PP)*: the DPM's outputs represent a set of rules for each user context. These rules are applied to several power knobs to control them. For example, for managing screen brightness, a rule is the range of the most suitable screen brightness levels for the specified context. If the user is in a dark environment and the foreground application does not require high brightness level, the rule could be [0%–25%]. It means that the maximum value of screen brightness configuration in this context is 25%.
- *Current State Module (CSM)*: this module retrieves the current usage of the device. It permits to not impact negatively the user experience before applying the power policies by configuring some used hardware components. The retrieved information by the CSM are the background application and the consumed resources like the Wi-Fi connectivity, the screen brightness intensity, the sound volume, the mobility sensors and so on. Taking into account the current device usage will also increase the accuracy of the generated power policies.
- *Optimizer Actuator (OA)*: it performs resource management and subsystems configuration. The aim of ENOrMOUS is to propose a generic solution that will manage all the resources. In this paper, we focus on CPU frequency scaling, sound and brightness management and Wi-Fi interface configuration. These resources represent the most power-hungry hardware components. The Optimizer Actuator takes into account the current power policy and the device's current state before applying the new power policy
- *User Satisfaction Checker (USC)*: after adjusting the resources and configuring the hardware, the user's satisfaction is checked by analyzing his/her behavior as shown in Fig. 4. The user satisfaction is checked as follows:
  - If the user disables the resources management, ENOrMOUS is notified to modify the power policy that is responsible for the user's dissatisfaction. For example, when the CPU frequency is scaled, the user satisfaction checker verifies the acceptance of the proposed frequency by the user. If a new frequency is set by the user, this new value is stored and will be taken into account for the next run in the same context and for the same foreground application.
  - In order to check the user satisfaction in terms of CPU configurations, for each specified context, hooks have been implemented as *ActionListners*. Specified contexts are defined by the executed foreground applications. For each foreground application, the

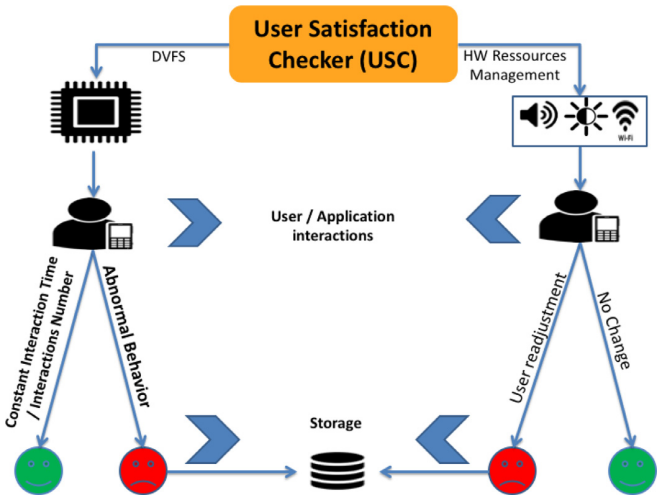


**Fig. 4.** User satisfaction checker.

evolution of the user's context is shown in Fig. 4. Thus, we have the different CPU frequencies allocated by ENOrMOUS. The number of contexts varies from one user to another depending on their behavior. We have on average 30 different contexts for the same application, knowing that we have on average 40 applications per user, the number of these contexts reaches 1200. The contexts are stored in an XML file that does not exceed 4 MB per user. The different stored contexts are enriched and redundancies are removed in order to reduce memory storage overheads. The storage of these different contexts is temporary, once the hardware configuration satisfies the user, All the context data is deleted. This operation reduces the impact on the memory consumption.

The ActionListeners identify interaction's speed and manner between the user and his mobile system. When the user launchesÃ§o an application during a given time, the number of times the user touches the screen and refreshes the application is collected and stored. The implementation of ActionListerners is simple and intuitive in the current version of ENOrMOUS. These ActionListerners have been implemented as Input libraries (ILs). The IL starts when a new foreground application is executed. If the user context has already been checked for user satisfaction,

the ActionListner is not started and vice versa. Then, an average is calculated and is compared to the current number of interactions. If a difference is detected between these two values, the CPU frequency is increased by the OA and the same process is repeated until no gap between these values is detected. A difference threshold of 20% is used for the number of touches to detect user dissatisfaction . A decrease or an increase of 20% is synonymous with a too low frequency for the user. These 20 % was determined after an explicit questionnaire with users who mentioned that their dissatisfaction can be expressed by incessant tactile support or by an expectation. These 20 % can vary from one user to another, for this reason, as a perspective we will determine this value by using classification algorithms depending on the user interaction manner. This new frequency will be allocated to for the next similar context, thus correcting classifier outputs.

In this paper, we take into account the user's reactions. Indeed, the user's reaction to an optimization is quite indicative of her/his satisfaction. This process allows us to improve the classifiers accuracy and gauge user needs deeply.

## 3. User context classification for power reduction

The user context is continuously changing at run-time. The changes concern several parameters such as the foreground and background applications running, the available Wi-Fi connectivity, the mobility of the user, the brightness or noise of the environment, the CPU workload and so on. The first consequence of these variations is that the user behavior and applications needs in terms of resources are strongly affected. Indeed, depending on the context, the user and applications may require more or less resources than those allocated by the operating system. For example, when user is in a dark or dim environment and depending on his/her preferences in terms of luminosity, we can reduce the screen brightness. This reduction is not taken into account by the OS and will permit us to have a gain in terms of power consumption. Secondly, different users may react differently to the same context changes. In a given context, two distinct users launching the same application may need different levels of resources configuration. This behavioral difference stems from many factors such as job, lifestyle, activity, eyesight and so on. In order to take advantage of these characteristics and differences, a device/user context classification has been designed. As mentioned before, the classifiers are implemented in the Data Processing Module (DPM). The classification purpose is to determine which is the device current context of use to be able to generate adaptive power policies for each hardware component. The predictive model issued from the ENOrMOUS classifiers must lead to provide to the user only the hardware resources that are needed for the his/her current context. This resource calibration has to satisfy two major requirements:

- to provide all the possible power reduction opportunities. In the worst case, the proposed power scheme is similar to the OS power consumption;
- to produce power policies which will not impact negatively the user's satisfaction.

The output of the classification is thus used to select the appropriate power policy for each of the controlled resources according to the current context, as detailed in the following.

### 3.1. ENOrMOUS resources classifiers

The aim of the classification is to create a model able to predict the right target values for four resources: CPU, sound volume, screen brightness and Wi-Fi, according to the user context. As mentioned before, the context is defined by information related to the user, the environment and the device:

$$context = (User, Environment, Device)$$

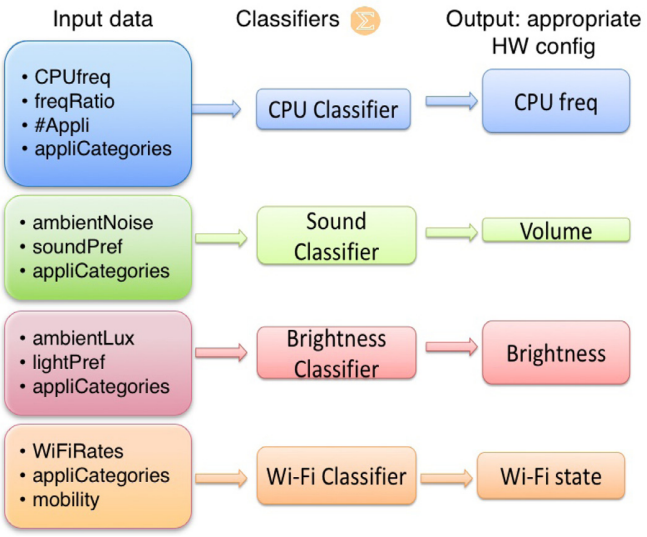


**Fig. 5.** Resources *ad-hoc* classifiers.

with :

- *User* data include two values about the user's preferences:

  $$User = (soundPref, lightPref)$$

  From the user probes (see Section 2), the sound level and the brightness level are regularly recorded during one full day. Based on this information, their average level is computed and then assumed to be preferred by the user;
- *Environment* data include ambient noise and lux levels as sensed by the device:

  $$Environment = (ambientNoise, ambientLux)$$

- *Device* data include the CPU current frequency, the ratio of the used CPU frequency to the maximum CPU frequency, the number of running applications, the application categories, the download and upload Wi-Fi rates, and the mobile use:

  $$Device = (CPUfreq, freqRatio, \#Appli, appliCategories, WiFiRates, mobility)$$

These information constitute the input of the classification. We consider the four resources as independent targets for the classification. For this reason, the context identification is achieved by 4 classifiers, one ad-hoc classifier for each resource as shown in Fig. 5. For each classifier, we have selected the input context data that are relevant given the output target resource.

#### 3.1.1. CPU classification

The CPU classifier uses data that cover application needs and user preferences in terms of computation:

- *CPUfreq*: this input is trivial and represents the CPU frequency allowed by the operating system. In our experiments, we consider four different values, 800 MHz, 1250 MHz, 1750 MHz and 2200 MHz, which are the CPU frequencies allowed by the mobile devices we used for the experiments.
- *freqRatio*: this parameter indicates the ratio of the allowed OS CPU frequency relatively to the maximal frequency. It helps us to get an idea about the user preferences, when this ratio is high, it means that the user needs high computational resources and vice versa.
- $\#Appli$: this input includes the number of running applications in background and foreground. It assists in finding the correlation between the number of running applications and the appropriate CPU frequency.

- *appliCategories*: it represents the category of the foreground application. Indeed, we made a specific classification of the applications into three distinct categories according to their requirements, as will be explained in the next Section 3.1.5. The application category is used here to determine its requirements in terms of computation.

The output of the CPU classifier is an appropriate CPU frequency among four classes. These classes are defined by the four frequency values we chose:

- *L*: context requiring a low CPU frequency (under 800 MHz).
- *M*: context requiring a medium CPU frequency (between 800 MHz and 1.25 GHz).
- *H*: context requiring high computing resources (between 1.25 and 1.75 GHz).
- *VH*: context requiring very high computing resources (over 1.75 GHz).

#### 3.1.2. Sound classification

Sound context is built according to three selected inputs:

- *ambientNoise*: this value indicates the noise level of the environment and is captured by the available embedded sensor on the platform. When the ambient noise is too high, the sound level must be increased and vice versa.
- *soundPref*: the sound level that is assumed to be preferred by the user (see Section 3.1). This data can be representative of the user hearing.
- *appliCategories*: the category of the foreground application is used here to determine its requirements in terms of sound level. For example the sound required by *Spotify* differs from the sound level needed by *Word*.

In the absence of a research background related to this question, we made the arbitrary choice of a four level scale for the sound volume, what constitutes a first attempt that should be refined. The output of this classifier is an appropriate sound level among four sound level classes:

- **VL** [0%–25%] : when the user needs a very low sound level.
- **L** [25%–50%]: when the need for the sound level is low.
- **M** [50%–75%]: when the need for the sound level is medium.
- **H** [75%–100%]: when the need for the sound level is high.

#### 3.1.3. Brightness classifier

The brightness classifier uses data that cover the ambient luminosity, the application needs and the user's preferences: Luminosity context is built according to three selected parameters from the probes.

- *ambientLux* : this input is common to the brightness management methods. For example in the Samsung Galaxy S7, the brightness adjustment in only based on the ambient luminosity [3]. This value indicates the luminosity of the environment and is captured by the available embedded sensor of the platform. This parameter value is very significant when the ambient luminosity is too high or when we need a high screen brightness level, and vice versa. The parameter values are available in the environment probe.
- *lightPref* : the screen brightness level that is assumed to be preferred by the user (see Section 3.1). These data may give information about the user's eyesight.
- *appliCategories*: the category of the foreground application is used here to determine its requirements in terms of screen brightness. For example, *Facebook* brightness needs differs from *Spotify* needs.

As for the sound, we used an arbitrary four level scale for the brightness. The output of this classifier is an appropriate brightness level among these four classes:

- **VL** [0%–25%] : when the user needs a very low screen brightness.
- **L** [25%–50%]: when the need for the screen brightness is low.
- **M** [50%–75%]: when the need for the screen brightness is medium
- **H** [75%–100%]: when the need for the screen brightness is high.

**Table 1**
Resource requirement categories for a selection of applications.

| Applications | CPU | Wi-Fi | Watching | Listening |
|---|---|---|---|---|
| Youtube | M | 1 | H | H |
| Word | L | 0 | H | L |
| Skype | L | 1 | M | H |
| Facebook | H | 1 | H | L |
| Messenger | M | 1 | H | L |
| Spotify | L | 1/0 | L | H |
| Adobe Reader | L | 0 | H | L |
| 2048 | M | 0 | H | L |
| Chrome | M | 1 | H | M |
| VLC | M | 0 | H | H |
| OneNote | L | 0 | H | L |
| Sudoku | L | 0 | H | L |

#### 3.1.4. Wi-Fi classification

For this classifier, we select 4 input data which are:

- *WiFiRates*: the rates of downloaded and uploaded data in kb/s.
- *appliCategories*: the category of the foreground application is used here to determine its requirements in terms of Wi-Fi.
- *mobility*: mobile use of the device, provided by an API that permits to know when the user is stationary, walking or running. It is calculated by using the accelerometeer data combined with the compass. It will help us define whether the user is close to a Wi-Fi hotspot or not.

The output of this class is an appropriate Wi-Fi state among these three states:

- **1**: when the user needs to be connected.
- **1/0**: the Wi-Fi interface is on but disconnected.
- **0**: the Wi-Fi interface is disabled.

All these input data values are captured at the same time and stored into a database. The time-period of a collection is adjustable depending on the foreground application change.

#### 3.1.5. Application categories

Each of the four classifiers includes in its input dataset the application category regarding the targeted resource. This parameter has been introduced to improve the classifiers accuracy. The application categories have been subjectively evaluated based on their functionality and our own usage, according to the CPU, Wi-Fi, watching and listening needs. This is illustrated in Table 1.

- For CPU, we use three categories, L for low need, M for medium need and H for high need in terms of computation.
- For Wi-Fi, we also use three categories, 1 for applications that require internet connectivity and 0 for applications that do not. We have also 1/0 for the applications that can run in offline mode like *Spotify* or *Google Maps*.
- For brightness and sound needs, we have respectively evaluated screen watching and listening parameters with three values L for low need, M for medium and H for high needs.

The application categories are used by the four classifiers previously described. These classifiers are at first trained to build a predictive model using supervised machine learning, that means we use a set of instances for which we have the desired outputs (targets). The resulting model is then used to determine the proper power adjustments according to the input data that identify new situations.

In the following, we present how we create the predictive models.

### 3.2. Creation of the predictive models

The aim is to create four models dedicated to predict the power resource requirements according to the context. These four models correspond to the four classifiers presented previously. This is a classification

problem since we know the target classes of the models. Among the numerous classification methods [4], we chose Artificial Neural Networks (ANN) because of our type of data and the uncertain aspect of user behavior [5]. Indeed, neural networks can be used to detect patterns and trends that are too complex to be noticed using statistics techniques. For example, in our case, the uncertain aspect of user behavior is linked mainly to his/her interaction manner following his context. These data may strongly impact the energy consumption of the mobile system and consequently the results of our proposed solution. In addition, we have various data that cover several axes to describe the user context as shown in Section 3.

A neural network is made of input nodes, output nodes and hidden nodes that link the input to the output nodes. The hidden nodes are organized in one or more layers, what forms the architecture of the network.

- The inputs units receive information to be processed. In our case, it is the collected user context.
- The outputs units receive the results of the processing. In our case, they are four context classes for each CPU, sound and brightness. For The Wi-Fi, we have three outputs units corresponding to each state.
- The hidden units connect the input units to the output units. Their activity is determined by the activities of the input units and the weights on the connections between the input and the output units.

Many studies discussed the use of several ANN architectures and training algorithms for classification. In ENOrMOUS, since it is based on a set of known-correct outputs for each context, the backpropagation (BP) models are appropriate. The BP algorithm adapts the network according to the input-output associations that are expected.

The method used to create this training dataset is described in the following section.

#### 3.2.1. Training dataset

The set of instances for which we have the desired outputs constitute the training dataset, which is based on the user's actions. Examples of appropriate brightness and sound level are inserted in the training dataset through the application of the change blindness mechanism [6]. Indeed, we exploit the change blindness concept by decreasing screen brightness and sound level gradually until it is no longer acceptable by the the user. When the screen brightness and the sound volume levels are gradually lowered and the user does not change these values manually, we conclude that the user is satisfied with these new screen brightness and sound volume values. Then, we store these appropriate brightness/sound values as targets for the specified context. However, when our automatic adjustment is not tolerated and is modified manually by the user, the target value for the screen brightness and the sound volume are the last values that have not been modified manually.

A similar method is used to detect relevant CPU frequency: we decrease the frequency until a manual readjustment by the user is detected. Then, we store the last frequency that satisfied the user. In order to define the Wi-Fi targets, for each input sample we try to determine a corresponding connectivity configuration by experimentation. The user feedback is taken into account to correct the classification.

This process represents the only phase where the user changes resources configuration manually. This is necessary for the construction of the learning dataset. This is done just once before the start of the classification. All remaining phases are automatic and do not require any user action. The differences in the users' needs and preferences explain why the classifiers are trained specifically for each user.

Fig. 6 shows the target classifiers selecting process.

#### 3.2.2. Network architecture

The back-propagation learning algorithm (BPLA) is one of the most studied and used algorithms for neural networks learning. It is a widely used method for ANN learning in many applications [7]. We used the sigmoid function to calculate the output of the neurons. The shape or architecture of the network is another parameter that must be decided upon to design an ANN. The architecture depends on the way the neuron layers are connected to each other, the number of hidden layers and the number of neurons per layer. Unfortunately, there is no generic method to determine the optimum values for these parameters.

About the connections between the layers, two architectures were tested: a simple Feed-forward BP architecture and a Cascade BP architecture [5]. Both of them are among the most popular multilayer ANN. Their network architectures differ by the number of connections between the neurons: in the feed-forward BP architecture, each hidden layer (n) node is connected to the previous layer (n-1) nodes, whereas in the cascade BP architecture, each hidden layer (n) node is connected to every previous layer (input layer–0, 1,..., n-1). We simulate and compare the results obtained by each type of architecture in order to choose the right model and the suitable configuration for each classifier and each user. The most accurate architecture will be chosen to be embedded on the device.

We perform tests with ANNs containing two intermediate layers and with variations from 1 to 10 neurons in each hidden layer. Weights and biases were randomly initialized. We select 200 inputs samples for each resource and each user to test and choose which of the feed-forward or the cascade BP architecture were the most appropriate. Our networks were trained during 100 training cycles (or epochs). These 200 samples were divided 80% for training, and 20 % for validation. The evaluation of our ANNs has been done by experimentations in order to select different architectures (hidden layers size) and comparing obtained results.

We noticed that:

- In our case, changing the intermediate layers and neurons number did not affect the accuracy result. On the contrary, increasing the ANN size was counter-productive and causes latency with power overheads.
- We also notice that changing the number of input samples affect the accuracy of the ANN. By selecting less than 200 samples, the accuracy was decreased.
- Increasing the number of input samples can lead to a problem of over-learning. In the literature, there is no method to determine the ideal number of samples. The most recommended method is experimentation and comparison of different results. For ENorMOUS the number of selected samples is sufficient to demonstrate the feasibility of our solution.

Table 2 presents the neural networks architecture comparison for our four classifiers. This table represents results for one user in order to show how the ANN architecture choice is done. The table presents the mean square error, the number of iterations (Epochs) performed for the validation performance to reach a minimum and the regression results.

- The mean squared error (MSE) for each ANN with the number of epochs is shown. The MSE indicates the accuracy of our algorithm, which should be at an acceptable level.
- Regression results show the relationship between the outputs of the network and the targets during the two phases: Training and validation. If the training were perfect, the network outputs and the targets would be exactly equal, but the relationship is rarely perfect in practice. The regression plot will confirm our choice concerning the right architecture and algorithm.

Training and validation represent the linear equation which shows the relationship between the targets and the obtained outputs for our two neural networks and for one user. The MSE and the regression results analysis provide the following information for the specific user:

- Cascade FBP provides better results than the simple Feed FBP for CPU and sound classifiers.
- Simple Feed FBP provides better results compared to the Cascade FBP for Brightness and Wi-Fi classifiers.
- The accuracy of our different neural networks by referring to the mean square error indicates that the number of samples is sufficient.

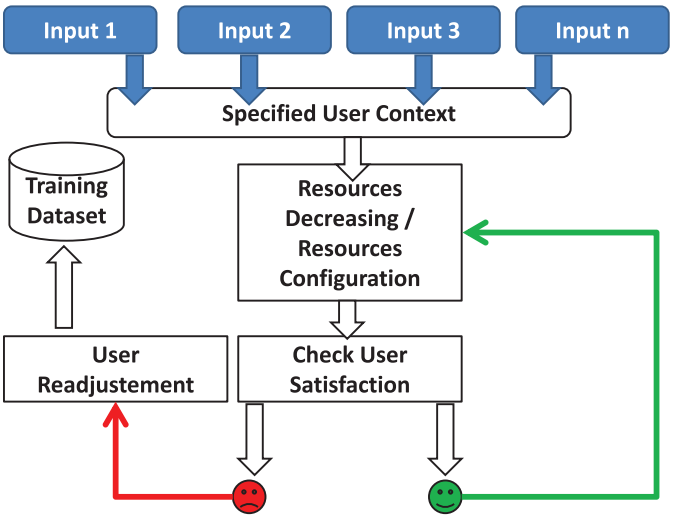


**Fig. 6.** Target classifiers selecting process.

**Table 2**
Neural network accuracy for one user.

| Resource | ANN | MSE | Epochs | Training | Validation |
|---|---|---|---|---|---|
| CPU | Feed FBP | 0.21 | 4 | 0.49*Tar + 0.073 | 0.51*Tar + 0.17 |
| | Cascade FBP | 0.062 | 12 | 0.71*Tar + 0.077 | 0.69*Tar + 0.012 |
| Wi-Fi | Feed FBP | 0.02 | 14 | 0.84*Tar + 0.058 | 0.84*Tar + 0.044 |
| | Cascade FBP | 0.11 | 10 | 0.61*Tar + 0.11 | 0.56*Tar + 0.15 |
| Sound | Feed FBP | 0.049 | 16 | 0.92*Tar + 0.002 | 0.87*Tar + 0.039 |
| | Cascade FBP | 0.044 | 3 | 0.69*Tar + 0.085 | 0.75*Tar + 0.07 |
| Brightness | Feed FBP | 4 e−08 | 19 | 0.89*Tar + 0.006 | 0.89*Tar + 0.096 |
| | Cascade FBP | 0.073 | 3 | 0.7*Tar + 0.11 | 0.7*Tar + 0.13 |

Increasing this number may eventually improve accuracy, but this requires further data collection.

The ANN architecture selection and accuracy can also be different for each user because of the interaction manner, resources requirements, running application and so on. This difference indicates the need for separate experiments to determine the most suitable architecture for each user.

## 4. Experimental results

This section presents the experimental results of our proposed solution. The purpose of these experiences is to validate our framework's architecture, evaluate the obtained energy consumption reduction and the cost of our solution. The presented evaluation is considered as preliminary results in an academic context. The section is divided in four parts. We first present the tools used in the implementation phase and in the experiments. In the second part, we present the obtained experimental results in terms of power consumption. The third part presents additional results regarding the power consumption results obtained with ENOrMOUS. We finally present the cost of our solution regarding the resource usage.

### 4.1. Tools and experimental environments

This section presents the experimental environment, the tools and their usage.

#### 4.1.1. Measure and control tool

The Intel Energy Checker SDKit (IecSDK) [8] has been used to implement our solution. The SDK has been designed to measure and optimize applications energy efficiency. Two components of the SDK are leveraged in this work: the main driver (ESRV - Energy Server) and the Modeler. The Modeler provides the services required to implement data collection process and energy saving heuristics. Several data collection extension modules, a.k.a. Inputs Libraries (ILs), as well as Actuators Libraries (ALs) have been developed. The Modeler is composed of three components: the Front-End (FE), the Input Bus (IB), and the Back-End (BE).

- The *Front-End(FE)* collects the data through the *Inputs Libraries (ILs)*: CPU Utilization, display brightness, battery level, front end applications, etc. If necessary, new data can be collected by developing new ILs.
- Once collected by the ILs, data are made visible on the *Input Bus*. Any module connected to the bus has direct access to the metrics. The IB is the main interface between the FE and the BE.
- The *Back-end*(BE) provides core services e.g. a logger or a power-to-inputs automatic correlation, a watchdog as well as an interrupts and communications manager. The BE can be expanded via *Actuators Libraries (ALs)*. ALs are designed to perform specific actions such as dynamic OS configuration and dynamic platform configuration. Usually ALs are used to implement various optimization heuristics that are driven in real-time by the inputs provided by the FE.

Fig. 7 depicts the architecture we used to implement our approach. ENOrMOUS has been implemented as:

- For the data collection module (DCM), we have three ILs corresponding to our three probes.
- For the data processing module (DPM), we have four ALs corresponding to each resources classifier.
- The current state module (CSM) has been implemented as an IL.
- Finally, we have four actuator libraries, one actuator library for each resource.

The developed Inputs Libraries and Actuator libraries are generic and cross platform. They can run under Linux/Android and iOS. They can a

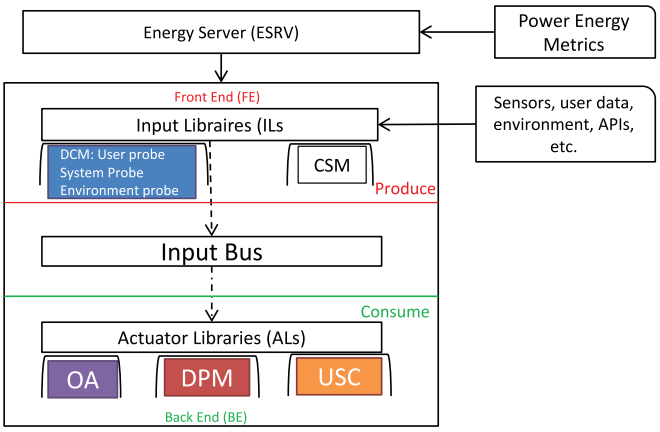


**Fig. 7.** ENOrMOUS architecture based on IecSDK.

**Table 3**
Intel 2in1 Ultrabook features.

| Analyzed plateform | 2in1 intel Ultrabook |
|---|---|
| Average battery life (Hours) | 8 H |
| Maximal power consumption (W) | 23 W |
| Minimal power consumption (W) | 11.5 W |
| Average power consumption (W) | 16 W |
| Number of cores | 2 |
| Number of threads | 4 |
| Processor base frequency | 2.00 GHz |
| Max turbo frequency | 3.20 GHz |
| Cache | 4 MB SmartCache |
| RAM | 4 GB |

priori be embedded on any mobile devices thanks to their low cost in terms of power consumption, memory and CPU usage.

#### 4.1.2. Development of the neural networks

For the neural networks setup for preliminary experiments as shown in Section 3.2.1, Matlab neural Networks toolbox functions are used to simulate and compare our networks, adjusting neurons weights to have the most appropriate configuration. After the simulation on Matlab, we have implemented the neural network functions as Actuator Library using a C Wrapper.

#### 4.1.3. Mobile device and OS

The experiments have been carried out on an Ultrabook running Windows 8.1 with a 2.50 GHz Intel dual-core i7-u3667U processor and 4GB of RAM.The Ultrabook is an Intel reference design 2 in 1 which can be used as tablet or laptop. Power measurement has been done using the Yokogawa WT210 power analyzer. Table 3 below presents our mobile device hardware features.

The ENOrMOUS principle is usable in almost all mobile systems but will require some adjustments and modifications depending on the platform. The main reason why the Ultrabook has been chosen to demonstrate the feasibility of our approach is because of the simplicity to connect it to our measurement device, the Yokogawa WT210 [9]. Other works like [10] propose a time energy model (TEM), which is a regression model to estimate the application energy consumption on real mobile devices. For more accuracy, in our work, we are using the Intel Energy Checker SDK and the Yokogawa WT 210 to measure the energy consumption of the whole process. This mobile device also contains a port for GSM Cards as well as a touch screen. In addition to these hardware features, with the Windows store, we have access to many metro style applications [11] such as *Facebook, Viber, Shazam, Instagram* and so on. These applications will be used as an entire application and will not be used thur a web browser. In addition, these applications are widely used both on smartphones and tablets. These characteristics makes our solution useful for other mobiles devices.

#### 4.1.4. Users population in the experiments

For our experiments mentioned in Section 4.2, we selected six real users with the several aforementioned probes and we collected the required information over two weeks. These six users correspond to six Master students with different profiles and habits. We also simulate six synthetic users to give more details about our power management solution. In total we have a population of 12 users.

### 4.2. ENOrMOUS power management results

In this section, we measure the gain obtained in power consumption with the use of our solution. To demonstrate our solution efficiency, we made tests in different scenarios with different real user contexts. Recall that the classification output is an appropriate frequency for the CPU, a range of values for the sound and the brightness and one state among three for the Wi-Fi.

The optimizer actuator decreases screen brightness and sound level gradually by *delta* units every *lambda* seconds to the smallest value that satisfies the user. *Delta* and *lambda* have been fixed experimentally to 3% and 4 seconds respectively to not impact the user satisfaction and to converge as quickly as possible to the optimal value.

For the CPU frequency, the Optimizer Actuator decreases the frequency through the Windows API. Finally for the Wi-Fi, the interface is directly configured to the appropriate state (on, disconnected or off).

Table 4 presents a users contexts snapshot. This table gives an instantaneous user context information for our six users, we have four sets of information, we recall that each set corresponds to one of the four neural networks. We also give the foreground application name in addition to its needs in terms of the four resources. These information represent a sample for one specified context, when the foreground application will change, the context will be different and obviously the results presented in Table 5 will also change. For each resource, Table 5 gives the corresponding classification results.

We notice from the two tables above that:

- The CPU frequency allowed by ENOrMOUS is always lower or equal to the CPU frequency set by the OS.
- The Wi-Fi classification is highly correlated with the foreground application connectivity need and the user mobility. For example user 2 is running *Facebook*, however the classification's result is Wi-Fi disconnected which is due to the user mobility.
- We can have a similar classification results for a different user contexts for example, user 4 and user 5 have the same classification results for sound [25%–50%] but their context information are different. The same information is noticed for the brightness classification regarding user 2 and user 5.

ENOrMOUS may determine a different configuration to the one we have in Table 4. If the user mobility changes for example, the Wi-Fi configuration will be different than what is presented in Table 5. Recall that the outputs of the classifiers indicate the power policies. The Optimizer Actuator retrieves these outputs and consult the Current State Module. Then, it applies the corresponding power policy on the appropriate power knob. Figs. 8–10 represent respectively the results from the OS vs. the ENOrMOUS configurations for CPU, sound and brightness.

Table 6 represents the ENOrMOUS Wi-Fi management in comparison with the default OS Wi-Fi management for the specified user context 4. These results correspond to the classification results of the specified user context in the tables above.

Fig. 11 shows the whole system power consumption gains obtained with ENOrMOUS for the specified user context 4. The measurement tests have been done 10 times, then the average has been calculated.

By analyzing the results, we notice: