Thèse de doctorat

Pour l'obtention du grade de Docteur de l'Université Polytechnique

des HAUTS-DE-FRANCE

Discipline :

INFORMATIQUE

Présentée et soutenue par Ismat, CHAIB DRAA.

Le 26/06/2018, à Valenciennes

Ecole doctorale :

Sciences Pour l'Ingénieur (SPI)

Equipe de recherche, Laboratoire :

Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines (LAMIH)

Optimisation de la consommation d'énergie des systèmes mobiles par l'analyse des besoins de l'utilisateur

JURY

Rapporteurs

- Etiemble, Daniel. Professeur émérite, Université Paris-Sud.
- Boukhobza, Jalil. Maitre de conférences, Université de Bretagne Occidentale (UBO).

Examinateurs

- Delot, Thierry. Professeur. Université Polytechnique des Hauts-De-France
- Grislin, Emmanuelle. Maître de conférence. Université Polytechnique des Hauts-De-France.
- Belleudy, Cécile. Maître de conférences, Université de Nice Sophia Antipolis.

Directeur de thèse

- Niar, Smail. Professeur. Université Polytechnique des Hauts-De-France

Loin de moi l'idée de faire un quelconque prosélytisme mais je remercie en premier lieu Dieu, s'en suivent mes parents et mon père de substitution qui m'a tout appris pendant 4 ans, le professeur Smail Niar, mes petits frères Yacine et Imed qui font office de fierté au quotidien, ainsi que tous mes amis, mes collègues d'ici ou d'ailleurs.
Je tiens également à remercier chaleureusement Emmanuelle Grislin et Jamel Tayeb qui m'ont beaucoup aidé et soutenu durant cette thèse.
Je remercie également toutes les personnes qui m'ont permis de réaliser cette thèse, en particulier mes enseignants et mes directeurs de formation.
Je dédie cette thèse à mes grands parents les défunts et les vivants et à tous les gens de Mazouna, d'Oran et d'Algérie. Je vous aime

Tu ne connaîtras jamais la fin de l'histoire en faisant demi-tour à deux minutes de la victoire. Merci à toi Monsieur OXMO PUCCINO pour tes textes qui m'ont tant apporté.

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 hardware resources. When these information are used properly, an efficient control of power knobs can be implemented to reduce the energy consumption.

This thesis has been achieved in collaboration with Intel Portland. The work described presents two software components and one framework for energy consumption reduction in mobile systems.

The first component is the *Sensor-Based Optimization Component (SBOC)* in which 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.

The second composnent is *Application Needs Based Optimization Componenat (ANBOC)*. 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 by performing optimizations such as dynamic frequency scaling (DFS), data prefetching, and device management without impacting negatively the user satisfaction. Such actions decrease the energy consumption of the whole mobile system.

Finally, the third contribution is ENOrMOUS framework for *ENergy Optimization for MObile platform using User needS* (ENOrMOUS). 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.

Keywords: Mobile Systems - Energy Consumption - User Habits And Needs - Machine Learning - Data Mining - Run-Time Analysis

Résumé

De nos jours, l'omniprésence des systèmes mobiles ne fait qu'accroitre et ces derniers deviennent indispensables pour nombreux d'entre nous. Les constructeurs de ces plateformes mobiles inondent le marché avec des produits de plus en plus performants et contenant un grand nombre d'applications énergivores. Le revers de la médaille de cette popularité est la consommation d'énergie. En effet, les caractéristiques des systèmes mobiles actuels ne font qu'accentuer le besoin d'une refonte des techniques d'optimisations de la consommation d'énergie.

Dans une époque où les consciences s'élèvent pour un monde plus « green », de nombreuses solutions sont proposées pour répondre à la problématique de la consommation d'énergie des systèmes mobiles. Cependant, dans les solutions existantes, le comportement de l'utilisateur et ses besoins sont rarement considérés. Cette omission est paradoxale car c'est le comportement de l'utilisateur final qui détermine la consommation d'énergie du système.

D'autre part, l'utilisation des données qui émanent des différents capteurs embarqués et du système d'exploitation peut être bénéfique pour mettre en place des politiques efficientes de gestion de puissance. Exploité à bon escient, l'important flux d'informations disponible est susceptible de servir à la caractérisation du comportement de l'utilisateur, ses habitudes et ses besoins en matière de configuration hardware. En assimilant ces informations et en les traitant, nous pouvons proposer des optimisations d'énergie sans altérer la satisfaction de l'utilisateur.

Dans cette thèse nous proposons le modèle CPA pour Collect – Process – Adjust. Ce modèle permet de collecter des données émanant de différentes sources, les traiter et en générer des politiques d'optimisations d'énergie. Les travaux de cette thèse ont été réalisés en coopération avec Intel. L'objectif de cette collaboration est la conception et la réalisation de solutions permettant d'améliorer la gestion d'énergie proposée par le système d'exploitation.

Pour concrétiser cette tâche, nous proposons dans un premier temps le composant **Sensor Based Optimization Component (SBOC)**. Ce composant exploite les données exposées par les capteurs embarqués afin de mettre en place des heuristiques permettant une configuration dynamique des composants matériels. Menant ainsi à une réduction de la consommation d'énergie dans certains cas de figure.

Nous proposons ensuite le composant **Application Needs Based Optimization Component (ANBOC)**. Ce composant logiciel classifie les applications selon leurs besoins en ressources et prédit les futures actions de l'utilisateur. La classification combinée à la prédiction permet de configurer les composants matériels de la plateforme dans une optique d'économie d'énergie. Cela est obtenu en implémentant efficacement des actions telles que l'ajustement dynamique de la fréquence du processeur et la gestion des interfaces de communication.

La troisième contribution de cette thèse est la réalisation du framework **ENOrMOUS pour ENergy Optimization for MObile platform using User needS**. Nous mettons en place à travers ce framework une solution englobante pour parer aux limitations des deux composants précédents. ENorMOUS corrèle la satisfaction de l'utilisateur à la consommation d'énergie de son système en réa-

lisant une classification du contexte de l'utilisateur en fonction des ressources matérielles. Cette classification permet de trouver un compromis entre la satisfaction de l'utilisateur et les opportunités d'économie d'énergie.

La dernière contribution de cette thèse est l'ajout d'une fonctionnalité inexistante dans les systèmes mobiles actuels, qui permet de rallonger la durée de vie de la batterie en fonction des requêtes de l'utilisateur. Cette dernière contribution est basée sur **ANBOC** et **ENOrMOUS**.

Les différentes contributions de cette thèse génèrent un gain d'énergie avoisinant les 38 %, en comparaison avec la gestion proposée par le système d'exploitation. Ce gain d'énergie est atteignable, tout en respectant la satisfaction de l'utilisateur.

Mots-clés: Systèmes mobiles - Consommation d'énergie - Besoins des utilisateurs - Intelligence artificielle

Publications

Revues internationales

- Chaib Draa Ismat, Niar Smail, Tayeb Jamel, et al. Sensing user context and habits for run-time energy optimization. EURASIP Journal on Embedded Systems, 2017, vol. 2017, no 1, p. 4.
- Chaib Draa Ismat, Niar Smail, Grislin Emmanuelle et al. ENOrMOUS: ENergy Optimization for MObile plateform using User needS. Journal of Software Architecture, 2018 (Submitted)

Conférences internationales avec comité de lecture

- Chaib Draa Ismat, Niar Smail, Tayeb Jamel, et al. User information analysis for energy consumption optimization in mobile systems. In : Proceedings of the Workshop on High Performance Energy Efficient Embedded Systems (HIP3ES) colocated with HIPEAC. 2015.
- Chaib Draa Ismat, Tayeb Jamel, Niar Smail, et al. Application sequence prediction for energy consumption reduction in mobile systems. In : Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on. IEEE, 2015. p. 23-30.
- Chaib Draa Ismat, Nouiri Maroi, Niar Smail, et al. Device Context Classification for Mobile Power Consumption Reduction. In : Digital System Design (DSD), 2016 Euromicro Conference on. IEEE, 2016. p. 682-685.
- Chaib Draa Ismat, Grislin Emmanuelle, et Niar Smail. An Energy-Aware Learning Agent for Power Management in Mobile Devices. In : International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. Springer, Cham, 2017. p. 242-245.

Table des matières

I Contexte général et état de l'art **1**

1 Introduction générale et problématique **3**
1.1 Contexte général 4
1.2 Problématique 6
1.3 L'impact des activités de l'utilisateur sur la consommation d'énergie 10
1.4 L'étude des activités de l'utilisateur 12
1.4.1 La sélection des composants hardwares 13
1.4.2 La collecte des activités de l'utilisateur 14
1.4.3 Analyse des activités de l'utilisateur 14
1.4.4 Exploitation des résultats de l'analyse 14
1.5 Les contributions de cette thèse 15
1.6 Plan du mémoire 17
1.6.1 Première partie: contexte et état de l'art 17
1.6.2 Deuxième partie: contributions 17
1.6.3 Troisième partie: conclusion et perspectives 18

2 État de l'art **19**
2.1 Les systèmes d'exploitation soucieux de la consommation d'énergie 20
2.1.1 Les systèmes d'exploitation soucieux de l'énergie 20
2.1.2 Les profileurs d'énergie et les gestionnaires de ressources 23
2.2 Optimisation de la consommation d'énergie des interfaces et protocoles de communication 26
2.3 Optimisation de la consommation d'énergie des capteurs embarqués des plateformes mobiles 28
2.4 Travaux centrés sur l'offloading computing 32
2.5 L'importance des activités de l'utilisateur dans la consommation d'énergie 33
2.5.1 Interaction de l'utilisateur avec les applications et les ressources 34
2.5.2 Interaction avec la batterie et les cycles de recharge . . . 38
2.5.3 L'exploitation des activités de l'utilisateur pour la modélisation d'énergie 40
2.6 Conclusion 42

3 Intel Energy Checker SDK et Environnement d'Expérimentations **45**
3.1 Intel Energy Checker Software Development Kit 46

3.2 Les composants principaux de l'Intel Energy Checker Software Development Kit (IECSDK) 47
3.2.1 L'Energy Server (Energy Server (ESRV)) 48
3.2.2 L'Intel Modeler 49
3.3 Plateforme de test et de mesure de puissance 53
3.4 Conclusion 54

II Contributions 57

4 Sensor Based Optimization Component 59
4.1 Contexte et motivation 60
4.2 Architecture modulaire de SBOC 62
4.2.1 Sensor Collection Module Sensor Collection Module (SCM) 64
4.2.2 Étude des données des capteurs et Sensor Processing Module (SPM) 66
4.2.3 Ressource Manager Module (RMM) 74
4.3 Évaluation et coût de SBOC 76
4.3.1 Scénarios, configuration et jeu de test 76
4.3.2 Coût de SBOC 82
4.4 Conclusion 84

5 Application Needs Based Optimization Component 85
5.1 Contexte et Motivation 86
5.2 Collecte de données 88
5.2.1 Architecture du DCM 88
5.3 Classification offline des applications 93
5.3.1 Classification binaire des applications en fonction des besoins en connectivité 93
5.3.2 Classification des applications selon les besoins en fréquence CPU 94
5.4 Prédiction des futures applications de l'utilisateur 100
5.4.1 Prédiction des applications 101
5.5 Fonctionnement de l'Optimizer Actuator (OA) 106
5.5.1 Configuration des ressources hardwares 108
5.5.2 Satisfaction de l'utilisateur 111
5.6 Résultats expérimentaux 112
5.6.1 Résultats de la prédiction et de la classification 112
5.6.2 L'évaluation d'ANBOC 114
5.6.3 Coût d'ANBOC 116
5.7 Conclusion 117

6 ENergy Optimization for MObile plateforms using User needS 119
6.1 Contexte et motivation 120
6.2 Architecture fonctionnelle d'Enormous 124
6.3 Classifiation des ressources via l'apprentissage supervisé 128
6.3.1 Classification des ressources 130
6.3.2 Création des modèles prédictifs 134
6.4 Architecture de nos réseaux de neurones 135

6.4.1 Définition de l'architecture de nos Multi Layer Perceptron (MLP) . . . 135
6.4.2 Notation utilisée pour la représentation de nos MLP . . . 137
6.5 Algorithme d'apprentissage propagation du gradient . . . 138
6.5.1 Feed Forward Back-Propagation(Feed-Forward Back Propagation (FBP)) . . . 143
6.5.2 Cascade Back-Propagation (Cascade Back Propagation (CBP)) . . . 143
6.5.3 Paramétrage de nos MLP . . . 143
6.6 Résultats expérimentaux . . . 146
6.6.1 Outils et environnement expérimental . . . 146
6.6.2 Résultats de la réduction de consommation de puissance avec Energy Optimization for Mobile plateform using User needS (Enormous) . . . 147
6.6.3 Coût d'Enormous . . . 156
6.7 Conclusion . . . 157

7 User Requests Based Optimization Component 159
7.1 Mode de fonctionnement et modules logiciels . . . 160
7.2 Fonctionnement de l'URBOC Optimizer Actuator (UOA) . . . 163
7.2.1 Application des Hard Optimization Policies (HOP) sans prédiction . . . 164
7.2.2 Le rôle de la prédiction dans User Request Based Optimization Component (URBOC) . . . 165
7.3 Exemples illustratifs du fonctionnement URBOC . . . 166
7.3.1 Application des HOP sans l'utilisation de la prédiction . 167
7.3.2 Scénario pour l'utilisation de la prédiction dans URBOC 171
7.4 Conclusion . . . 173

III Conclusion et perspectives 175

8 Conclusion générale et perspectives 177
8.1 Conclusion Générale . . . 178
8.1.1 Sensor Based Optimization Component Sensor Based Optimization Component (SBOC) . . . 178
8.1.2 Application Needs Based Optimization Component Application Needs Based Optimization Component (ANBOC) 178
8.1.3 Enormous . . . 178
8.1.4 User Request Based Optimization Component (URBOC) 179
8.2 Perspectives . . . 179
8.2.1 Perspectives générales . . . 180
8.2.2 Améliorations d'ANBOC . . . 180
8.2.3 Améliorations d'Enormous . . . 182

Table des figures ii

Liste des tableaux vi

Bibliographie x

TABLE DES MATIÈRES

Glossaire

AL Actuator Library. 52, 74, 106, 146, 147

ALS Ambient Light Sensor. vi, 68, 71–73, 125

ALs Actuator Libraries. ii, 48, 52, 54, 55, 84, 117, 146, 157

ANBOC Application Needs Based Optimization Component. iii, viii, 17, 85, 87, 88, 93, 98–101, 106–111, 114–120, 122, 124, 125, 127, 128, 131, 132, 134, 146, 147, 153, 158–162, 165, 173, 177–181

ANNs Artificial Neural Networks. 129

ANS Ambient Noise Sensor. 131

API Application Prgoramming Interface. 21, 47, 66, 90, 96, 100, 126, 147, 162, 164, 167

APIs Application Programming Interfaces. 29, 59, 63, 122, 126, 147

BLC Battery Life Checker. 162, 164, 165, 168–170

BPLA Back Propagation Learning Algorithm. viii, 138, 140, 142

CBP Cascade Back Propagation. 120, 142–144

CPA Collect Process Adjust. ii, iii, 15, 16, 47, 59, 61, 84, 85, 87, 88, 117, 119, 122, 123, 127, 157, 159, 177, 178

CSM Current State Module. 127, 146, 152, 161

DCM Data Collection Module. iii, viii, 87–93, 96, 98, 100, 102, 112, 124

DPM Dynamic Power Management. 8, 25

DVFS Dynamic Voltage Frequency Scaling. iii, 8, 9, 94, 95, 108, 109, 115, 123

EDCM Enormous Data Collection Module. iv, 124–127, 130, 133, 146, 156, 161

EDPM Enormous Data Processing Module. 126–129, 146, 147, 156, 161

Enormous Energy Optimization for Mobile plateform using User needS. iii, iv, 118–124, 126–128, 133, 134, 136, 138, 142, 146–151, 153–173, 177–179, 182

EOA Enormous Optimizer Actuator. 127, 128, 147, 162

EP Environment Probe. 125, 126

ESRV Energy Server. ii, 46–49, 54, 96, 180

FBP Feed-Forward Back Propagation. 120, 142–144

GPS Global Positionning System. 65, 66

GSP Generalized Sequenced Pattern. iii, 102–105, 112, 117, 165

HOP Hard Optimization Policies. iv, vii, 160, 162–173

IECSDK Intel Energy Checker Software Development Kit. ii, iv, 45–49, 54, 55, 59, 84, 116, 117, 126, 146, 177, 180

IL Input Library. 49, 50, 52, 62, 64, 84, 89, 146

ILs Input Libraries. ii, 47, 48, 50–52, 54, 55, 117, 146, 157

MLP Multi Layer Perceptron. iii, vi, vii, 120, 135–138, 142–144, 146

MSE Mean Square Error. 144

OA Optimizer Actuator. iii, 86, 88, 93, 99, 105–108, 110, 111, 127

OS Operating System. iii, 5, 9, 16, 17, 20–24, 32, 38, 42, 45, 59, 63, 67, 76, 78–80, 82, 84, 85, 87, 88, 90, 93–96, 98, 106, 107, 110, 115–117, 119, 126, 147, 153, 159, 163, 174, 178–180

PDH Performance Data Header. 96

PP Power Policies. 127, 162, 169

RMM Resource Manager Module. iii, 62, 63, 74, 75, 77, 79, 80, 84

SBOC Sensor Based Optimization Component. ii, iii, 17, 59–63, 68, 74, 76–80, 82, 84, 85, 88, 119, 120, 125, 131, 132, 146, 150, 159, 177, 178

SCM Sensor Collection Module. 60, 62, 64, 66, 73, 74, 76, 84

SHA Secure Hash Algorithm. 180

SKD Statistics Knowledge Database. iv, vii, 161, 162, 165, 170, 171, 173

SP System Probe. 125

SPADE Sequential PAttern Discovery using Equivalence classes. 102

SPM Sensor Processing Module. ii, iii, 60, 62, 66, 67, 73–75, 84

TTM Time To Market. 50, 52, 55

UMTS Universal Mobile Telecommunication System. 26, 27

UOA URBOC Optimizer Actuator. iv, 160, 162–166, 169, 170, 172

UP User Probe. 125

URBOC User Request Based Optimization Component. iv, 17, 159–161, 163, 165–167, 169, 170, 172–174, 177, 179

USC User Satisfaction Check. 128, 135

Première partie

Contexte général et état de l'art

Chapitre 1

Introduction générale et problématique

Dans ce chapitre, nous présentons le contexte général de cette thèse. Nous y exposons également la problématique centrale à laquelle nous proposons des solutions tout au long de ce manuscrit. Le contexte de notre travail est étroitement lié aux avancées technologiques de l'informatique récente et plus précisément de l'informatique mobile. En effet, ces dernières années, l'émergence des systèmes mobiles sous leurs différentes formes cadence tous les aspects de notre quotidien. Il existe aujourd'hui dans le marché toutes sortes de systèmes mobiles pour tous les besoins et tous les budgets. La nature de ces systèmes diffère également selon l'utilisation, nous retrouvons un large panel de ces plateformes mobiles tels que les smartphones, les tablettes, les *Ultrabooks* ou encore les montres connectées. Malgré ce large choix et ces différences apparentes dans la nature de ces systèmes, ils ont néanmoins tous un point en commun qui est la limitation de l'autonomie de la batterie et de façon plus générale, la problématique de la consommation d'énergie. En effet, la consommation d'énergie de ces systèmes mobiles est une problématique qui concerne tout le monde, allant de l'utilisateur *Lambda* jusqu'aux multinationales qui produisent ces systèmes en passant par les développeurs d'applications mobiles et le monde de la recherche scientifique.

Dans cette thèse, nous nous intéressons à l'optimisation de la consommation d'énergie de ces systèmes en prenant en compte les besoins de l'utilisateur, ses habitudes et son contexte. Cette tâche est réalisée en exploitant l'important flux de données disponible dans ces plateformes mobiles. Le travail réalisé dans cette thèse est multidisciplinaire et est à mi-chemin entre les systèmes d'exploitation mobiles, les techniques d'intelligence artificielle et l'informatique ubiquitaire. Cette thèse a été achevée dans le cadre d'une collaboration avec l'entreprise Américaine *Intel Corporation*. Le but de cette coopération est de mettre en place des solutions innovantes pour traiter de la problématique récurrente de la consommation d'énergie dans les systèmes mobiles.

Ce premier chapitre est organisé comme suit:

- Dans la section 1.1, nous situons le contexte général de notre travail.
- La section 1.2 présente les différentes problématiques auxquelles nous essayerons de répondre à travers nos contributions.
- Dans les sections 1.3 et 1.4, nous détaillons le point culminant auquel nous nous intéressons, qui est la prise en compte des besoins de l'utilisateur dans l'optimisation de la consommation d'énergie.
- La section 1.5 énumère les contributions de cette thèse, suivi du plan dans la section 1.6

Sommaire

1.1 Contexte général **4**
1.2 Problématique **6**
1.3 L'impact des activités de l'utilisateur sur la consommation d'énergie **10**
1.4 L'étude des activités de l'utilisateur **12**
1.4.1 La sélection des composants hardwares 13
1.4.2 La collecte des activités de l'utilisateur 14
1.4.3 Analyse des activités de l'utilisateur 14
1.4.4 Exploitation des résultats de l'analyse 14
1.5 Les contributions de cette thèse **15**
1.6 Plan du mémoire **17**
1.6.1 Première partie: contexte et état de l'art 17
1.6.2 Deuxième partie: contributions 17
1.6.3 Troisième partie: conclusion et perspectives 18

1.1 Contexte général

Depuis des années, l'informatique s'immisce de plus en plus dans la vie de tous les jours. Cela a commencé avec des ordinateurs de la taille de plusieurs pièces, pour arriver aujourd'hui au début du 21ème siècle, à des ordinateurs ultra-portables de la taille d'un cahier A4. Mais toutes ces avancées ont surtout donné naissance aux systèmes mobiles. Ces systèmes peuvent se décliner sous différentes formes telles que les smartphones, les tablettes, les ultrabooks, les montres connectées, etc. De nos jours, ces plateformes mobiles dépassent le stade de simples outils et deviennent littéralement une partie intégrante de nos vies. En effet, nous nous servons de ces systèmes pour nos communications, nos loisirs, nos activités personnelles, nos activités professionnelles, nos rencontres et tout ce qui peut se rapprocher de loin ou de près à notre quotidien.

Depuis plusieurs années, le nombre des systèmes mobiles est en augmentation exponentielle, ce qui en fait incontestablement la technologie de la décennie. De plus, cette augmentation est considérée comme un indicateur des tendances du marché des nouvelles technologies. En effet, les ventes de smartphones par exemple en 2016 ont avoisiné les 1.470.000.000. Il se vend près de 38 smartphones chaque seconde dans le monde [1]. À titre indicatif, en France le nombre de smartphones prédit d'ici 2019 est de 41 millions, ce qui représentera 66% de la population du pays. La figure 1.1 représente le nombre d'utilisateurs de smartphones dans le monde de 2014 jusqu'à une prédiction pour l'an 2020.

Ces chiffres démontrent l'avènement des systèmes mobiles et leur popularité indéniable. La conséquence d'une telle demande est l'offre de la part des fabricants de systèmes de plus en plus performants et diverse. Ces plateformes proposent une large panoplie de fonctionnalités et d'applications. Nous retrouvons des caractéristiques matériels complexes en amélioration constante, ils incluent un grand nombre de cœurs, de puissantes cartes graphiques, des écrans

1. https://www.planetoscope.com/electronique/728-ventes-mondiales-de-smartphones.html

avec une résolution de plus en plus importante pour une qualité d'image en constante évolution. Ces systèmes contiennent également de larges caches, gèrent de multiples réseaux et embarquent un nombre significatif de capteurs. Prenons l'exemple du *Samsung Galaxy S7* lancé en 2016 comportant 10 capteurs embarqués, ce qui représente 6 capteurs additionnels [2] en comparaison avec le *Galaxy S* lancé en 2010. Le nombre de cœurs quant à lui, a augmenté de 1 à 8 cœurs. Nous pouvons également citer la *Surface Book* de Microsoft qui est un Ultrabook 2^{en}1 pouvant être utilisé comme une tablette ou comme un pc portable, qui peut avoir jusqu'à 32 GO de RAM.

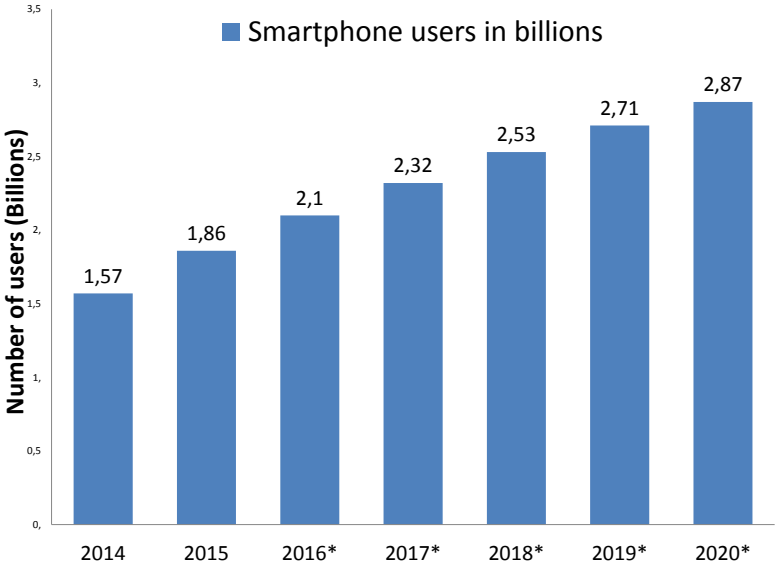

FIGURE 1.1 – L'évolution du nombre d'utilisateurs de smartphones de 2014 à 2020

Ajoutons également le fait qu'il existe aujourd'hui un marché important pour les systèmes d'exploitations mobiles qui parvient même à reléguer au second plan le marché des systèmes d'exploitation classiques. Nous connaissons tous des gens de notre entourage utilisant des systèmes sous *Android OS*, *iOS* ou encore *Windows Phone* [56]. Ce large panel permet de répondre aux différentes demandes du marché sachant que chaque Operating System (OS) tente d'apporter des fonctionnalités en plus par rapport à ses concurrents.

Au niveau des applications, la situation a énormément évolué aussi. En effet, les avancées technologiques ont permis aux développeurs d'applications mobiles d'être de plus en plus inventifs. Nous trouvons aujourd'hui des applications dédiées à presque toutes nos activités quotidiennes allant des plus classiques jusqu'à des applications utilisées pour suivre son alimentation, les pratiques sportives, les transports, l'e-commerce, etc. En addition à cela, les réseaux sociaux sont omniprésents dans notre vie, ce qui accroit encore davantage l'utilisation de ces systèmes mobiles.

Cet accroissement opère sur le plan individuel et sur le plan planétaire. Une

2. http://www.gsmarena.com/samsung-phones-9.php.

étude a démontré récemment que les Français consultent leurs smartphones en moyenne 26,6 fois par jour. Pour les jeunes (18-24 ans), ce chiffre est quasiment doublé et avoisine les 50 fois par jour. Une autre étude affirme que pour un Français sur cinq, il se passe moins de cinq minutes entre le réveil et le premier coup d'œil à sa plateforme mobile. Une proportion qui double là encore chez les 18-24 ans [3].

Cependant ces systèmes qui cadencent le rythme de nos vies aujourd'hui n'ont pas que des avantages et présentent un certain nombre de limites malgré les avancés majeures dans le domaine. En effet, il reste encore quelques limitations dont la résolution est critique. La plupart de ces limites sont liées principalement aux différentes propriétés des systèmes mobiles telles que: la mobilité, la diversité des natures des applications, la connectivité permanente et l'autonomie en énergie. Ces limites ont été détaillées dans [93] et sont principalement:

- L'insuffisance de la bande passante;
- La sécurité dans la connectivité;
- Les risques potentiels sur la santé;
- L'interface humaine avec le système;
- Les interférences de transmission du signal de connexion;
- La consommation d'énergie.

Parmi toutes ces limites citées ci-dessus, une des plus importante et qui fait l'objet de nos travaux de recherche est la consommation d'énergie.

1.2 Problématique

L'un des revers de la médaille de toutes ces caractéristiques des systèmes mobiles et des avancées technologiques est la problématique de la consommation d'énergie. Alors que les fonctionnalités des systèmes mobiles de dernière génération sont désormais assez similaires, l'enjeu de l'autonomie est devenu l'un des nerfs de la guerre entre les constructeurs. En effet, les batteries lithium-ion [64] qui équipent la plupart de nos systèmes mobiles actuels sont meilleures que les anciennes batteries au nickel-hydrure métallique de leurs récents ancêtres. Reste qu'elles n'ont pas fondamentalement progressé depuis plusieurs décennies, contrairement aux systèmes actuels qu'elles alimentent.

Les systèmes mobiles obéissent évidemment à la loi de Moore [115] qui affirme que la puissance de calcul d'un processeur double tous les dix-huit mois, à prix constant. Cette loi est vérifiable en informatique mais pour les batteries, c'est la chimie qui fait la loi et la chimie n'avance pas au même rythme que l'informatique comme l'a mentionné Jean-Marie Taracscon [4], directeur du réseau sur le stockage électrochimique de l'énergie (RS2E). Nous arrivons à la conclusion que les batteries se développent lentement alors qu'en parallèle [43], nos systèmes mobiles sont de plus en plus gourmands en énergie avec toutes les évolutions que nous leurs attribuons. Il est donc impossible dans ces conditions d'obtenir des autonomies plus importantes en se focalisant uniquement sur les batteries.

3. https://www.lci.fr/societe/journees-sans-telephone-portable-10-chiffres-qui-prouvent-que-nous-sommes-accros-a-notre-smartphone-2024614.html

4. https://www.francetvinfo.fr/internet/smartphones-pourquoi-les-batteries-vraiment-performantes-ne-sont-pas-pour-tout-de-suite-60017.html

Avant de rentrer dans le vif du sujet, posons nous la question pourquoi la consommation d'énergie dans les systèmes mobiles est un enjeu important ? Nous répondons à cette question avec les points suivants:

- Comme nous l'avons mentionné ci-dessus, les performances demandées par les systèmes mobiles augmentent à une plus grande vitesse que les avancées technologiques dans les capacités des batteries. Par conséquent, la nécessité d'augmenter les performances des systèmes mobiles entre en conflit direct avec le désir de rallonger la durée de vie de la batterie.
- La consommation énergétique a un impact direct sur la sécurité des utilisateurs. En effet, les constructeurs réduisent la taille de leurs systèmes et par conséquent la dissipation de la chaleur devient un énorme problème. Dans [100], il est démontré que la température d'un système mobile augmente considérablement lorsque ses composants hardwares sont activés. Nous avons pu voir les risques que la dissipation thermique est susceptible d'engendrer avec le *Samsung Galaxy Note 7* en 2016. En effet, le géant Sud-Coréen a dû retirer du marché tous ses produits *Note 7* pour des raisons de surchauffe de batterie.
- Pour une autonomie plus longue, la solution la plus préconisée par les constructeur est une augmentation de la taille des batteries. Cette augmentation est diamétralement opposée avec la diminution de l'épaisseur, le poids et le cout financier du téléphone.
- Une autre problématique posée par la consommation d'énergie est la limitation du temps de fonctionnement. Les constructeurs ont tout intérêt à augmenter l'autonomie de leurs produits car cette dernière est intrinsèquement liée au temps de fonctionnement du système mobile. Étant donné que les batteries sont de taille fixe, la durée de fonctionnement en un seul cycle de charge est également limitée.
- L'enjeu économique est très présent et l'autonomie des plateformes mobiles est un des facteurs les plus importants qui puisse pousser un consommateur à acheter un nouveau produit.

Toute l'importance de l'optimisation de la consommation énergétique qui découle des points susmentionnés a donné naissance à des axes de recherche académique et industrielle importants. En effet, pour parer à cette limitation énergétique, de nombreux travaux ont souligné l'importance d'optimiser la consommation de puissance dans les systèmes mobiles. Ces travaux se sont intéressés aux différents niveaux d'optimisation possibles afin de trouver la brèche qui permettrait idéalement de faire face à ce goulot d'étranglement. Les différents niveaux visés par les travaux de recherche informatique et électronique dédiés à cette problématique sont illustrés dans la figure 1.2.

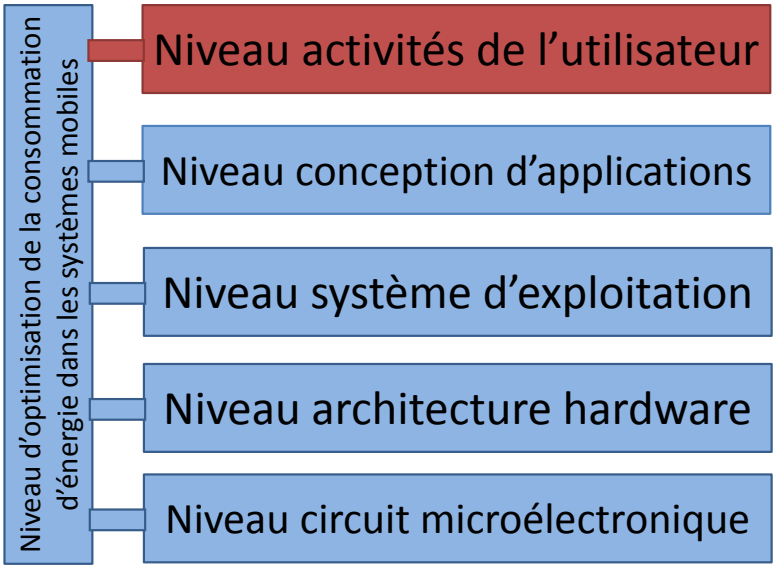

FIGURE 1.2 – Les différents niveaux d'optimisation d'énergie dans les systèmes mobiles

Nous détaillons les niveaux présentés dans la figure ci-dessus, en commençant par le niveau le plus bas.

- **Niveau circuit microélectronique** : les approches de ce niveau consistent à réaliser la conception des circuits de silicium consommant moins d'énergie. Cela est réalisé en exploitant les lois de la physique à l'aide de simulateurs [12] pour la validation des circuits conçus avant la conception. Ce niveau est très coûteux financièrement et demande une refonte totale des systèmes existants qui ne peuvent pas en bénéficier.
- **Niveau de l'architecture hardware**: les techniques du niveau architectural hardware se focalisent sur la fabrication des composants matérielles à plusieurs états de consommation d'énergie. En générale, ces composants supportent deux états qui sont l'état actif et l'état *idle* [5]. L'état actif en question peut être subdivisé en différents sous-états afin de procéder à une économie d'énergie [84]. Une des techniques les plus répandues est le Dynamic Power Management (DPM) dans toutes ses variantes [11, 14]. Les inconvénients de ces techniques sont principalement une forte latence qui cause une perte considérable de la puissance quand le composant hardware se trouve dans une état *idle*. Par conséquent, une pénalité en matière de performance en résulte toujours durant le passage d'un état à un autre. Nous pouvons également citer d'autres techniques appartenant à ce niveau comme le Dynamic Voltage Frequency Scaling (DVFS), les *C-State* et les *P-State* [6].
- **Niveau du système d'exploitation**: il s'agit là de mettre en place dans les systèmes d'exploitations mobiles des politiques permettant une bonne gestion de la consommation d'énergie [112]. Cela est réalisé en combinant l'utilisation des différentes ressources matérielles disponibles dans

5. état inactif

le système et les différents états que peut avoir un composant (eg *idle*, actif, ...). Nous pouvons retrouver aujourd'hui dans la majorité des systèmes d'exploitations un profil d'économie d'énergie qui tend à proposer à l'utilisateur de rallonger la durée de vie de la batterie de quelques minutes voir quelques heures d'utilisation supplémentaires. Une autre technique très utilisée au niveau de l'OS pour la gestion de la consommation d'énergie du processeur est l'exploitation des gouverneurs CPU [18]. Le gouverneur est un module du noyau du système d'exploitation qui est responsable de la gestion de la fréquence du processeur en fonction de la demande des applications. Lors du lancement d'un jeu par exemple, le gouverneur va augmenter la fréquence du processeur, puis va procéder à la diminution de cette fréquence lors de la mise en veille. Le plus connu des gouverneurs étant le *OnDemand Governor* utilisé sous *Linux* et *Android OS*. Ces gouverneurs s'appuient sur l'utilisation du DVFS mentionné dans le niveau précédent. Nous pouvons également retrouver dans les OS une gestion adaptative de la luminosité de l'écran basée sur la lumière ambiante. Les optimisations de ce niveau ont tout de même permis d'améliorer la consommation énergétique, il reste néanmoins un bon nombre d'opportunités de réduction de puissance qui ne sont pas prises en compte au niveau OS. Un des objectifs de cette thèse est d'améliorer les politiques d'optimisations énergétiques proposées par les systèmes d'exploitation en s'inspirant du dernier niveau présenté dans cette liste.

- **Niveau conception d'applications**: ce niveau s'intéresse à réaliser des applications consommant de l'énergie de façon optimale. Il exige la connaissance des effets de l'application sur la consommation totale d'énergie. Pour cela, le développeur doit connaitre le modèle de consommation de puissance de la plateforme ciblée. Certains travaux de recherche ont été engagés dans ce sens pour développer des outils mesurant l'efficacité énergétique de l'application. Cette mesure intervient pendant les différentes étapes de la conception [59]. Néanmoins, le problème avec ce niveau est qu'il n'est pas dans la culture des développeurs de prendre en considération l'efficacité énergétique de leurs applications. Cette approche n'est pas encore ancrée dans les habitudes des développeurs. Ajoutons à cela le fait qu'il va falloir optimiser les applications selon les différentes plateformes existantes, causant ainsi des problèmes de portabilité. De plus, il n'existe pas beaucoup d'outils qui permettent la mesure de l'efficacité énergétique au niveau de la conception de l'application.
- **Niveau activités de l'utilisateur**: ce niveau d'optimisations s'intéresse essentiellement à l'utilisateur, ses activités, son environnement et son interaction avec son système mobile. L'objectif est de cerner les besoins de l'utilisateur et sa manière d'interagir avec son système et ses applications. Cela permet de proposer une gestion plus efficace que celle offerte par le système d'exploitation. Le travail réalisé dans cette thèse s'inspire de ce niveau en y apportant un bon nombre d'améliorations et de modifications, toujours dans une optique d'optimisation de la consommation énergétique des systèmes mobiles.

Les différents niveaux mentionnés ont permis d'optimiser la consommation énergétique des systèmes mobiles de manière significative. Cependant, l'évolution croissante de nos plateformes mobiles rend insuffisantes les différentes

optimisations réalisées. En effet, chaque niveau comporte certaines limites qui poussent encore la communauté de recherche à explorer de nouvelles pistes pour parer à cette problématique. Comme mentionné précédemment, le point de départ de notre approche s'inspire du dernier niveau d'optimisations basées sur l'utilisateur. Nous avons jugé nécessaire d'accorder plus d'importance à l'utilisateur et ses besoins pour divers raisons que nous explicitons dans la section suivante et tout au long de nos contributions.

1.3 L'impact des activités de l'utilisateur sur la consommation d'énergie

L'utilisateur impacte directement la consommation énergétique de son système mobile [8, 20, 51, 77]. Dans la plupart des systèmes existants, c'est l'utilisateur qui détermine directement ou indirectement les applications à exécuter ainsi que les différents composants matériels à solliciter. Cela se fait selon ses besoins comme par exemple: le bluetooth pour écouter de la musique sans fil, le GPS pour la navigation, la Wi-Fi pour une connectivité internet, etc.

Afin de déterminer cet impact, des travaux de recherches ont été menés et tous se sont accordés sur le fait que la consommation d'énergie varie en fonction des activités de l'utilisateur. Par conséquent, d'un utilisateur à un autre, nous aurons des activités différentes ce qui implique logiquement une consommation d'énergie qui varie. Les travaux [8, 20, 51, 77] ont également souligné l'importance d'étudier les activités réelles de l'utilisateur pour détecter les applications les plus énergivores et celles qui sont les plus utilisées par ce dernier. La figure 1.3 montre un exemple de l'impact des activités de l'utilisateur sur la consommation totale d'énergie illustrée dans le travail d'Alex Shye et al [120].

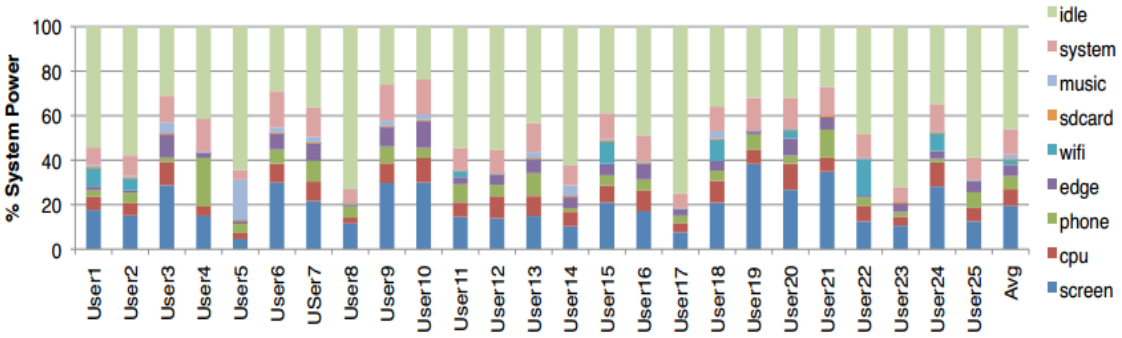

FIGURE 1.3 – La consommation de'énergie de différents utilisateurs dans l'état actif et *idle* [120].

En analysant la figure ci-dessus, nous remarquons évidemment que la consommation de puissance de chaque composant matériel varie d'un utilisateur à un autre. Il résulte de cette différence un gap dans la consommation d'énergie totale entre les différents utilisateurs. Cette différence en matière de consommation est due principalement à la diversité dans les activités des utilisateurs. Cette diversité est détectée par exemple avec les utilisateurs 5 et 10. L'état *idle* de

l'utilisateur 5 consomme plus de 60% de l'énergie totale, ce qui signifie que le système de cet utilisateur passe la plupart de son temps dans un état de veille. Tandis que pour l'utilisateur 10, l'état *idle* ne consomme que 25% de l'énergie totale. Cela nous indique que cet utilisateur passe plus de temps à utiliser son système mobile que l'utilisateur 5.

La consommation d'énergie est également différente dans l'état actif, c'est à dire que les utilisateurs ne sollicitent pas les composants matériels de la même manière. La figure 1.4 illustre cet aspect. Elle représente la consommation d'énergie de différents composants matériels dans l'état actif. Un exemple de cette différence est entre l'utilisateur 19 et l'utilisateur 5.

Avec l'utilisateur 19, l'écran est le composant qui consomme le plus de puissance, environ 40% de la consommation totale et 58% lorsque le système est dans un état actif, suivi de la consommation du système d'exploitation qui représente un ratio de 23% de la consommation totale.

Cependant l'activité la plus énergivore avec l'utilisateur 5 est la lecture de musique avec une consommation de 55% de l'énergie globale, suivie de la consommation l'écran et celle du système d'exploitation.

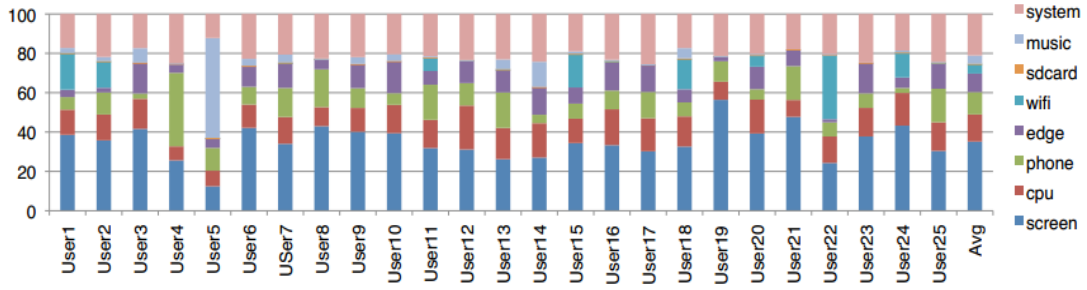

FIGURE 1.4 – La consommation d'énergie des différents utilisateurs dans l'état actif [120]

Ces résultats confirment que les activités de l'utilisateur sont différentes selon les applications exécutées, les fonctionnalités sollicitées et les performances exigées. Donc, l'activité des composants matériels est bel et bien différente d'un utilisateur à un autre. Par conséquent la consommation d'énergie résultante de ces activités varie également.

De plus, la consommation des composants hardwares est différente d'une activité à une autre [34], selon la charge de travail du composant. La figure 1.5 démontre ce principe. Nous y retrouvons la consommation d'énergie d'un smartphone *Nexus One* dans quatre cas de figures distincts qui sont: l'état *idle*, une connexion 3G active, une connexion Wi-Fi active et le capteur GPS actif.

En analysant les quatre parties de la figure, nous observons qu'il y a une différence dans la consommation totale du système corrélée avec le composant utilisé. Lorsque le système est en mode *idle*, la puissance est de 723.3 mW. Elle est de 2034.8 mW lorsque la connexion 3G est activée, de 1881.7 mW dans le cas où la Wi-Fi l'est et de 2091.8 mW lorsque que le GPS est utilisé. Cette observation nous confirme que la consommation varie d'un composant hardware à un autre [28, 79, 105]. De plus, la consommation de chaque composant diffère en fonction de l'état global du système, si ce dernier est en mode actif ou en mode *idle*.

Un exemple de cette observation est la consommation de l'interface Wi-Fi. Lorsque la connectivité est sollicitée, l'interface Wi-Fi consomme environ de 1243.7 mW ce qui représente 66% de la puissance totale. Tandis que cette même interface Wi-Fi consomme 85.2 mW dans les autres cas de figure. Donc l'interface Wi-Fi consomme 14.6 fois plus de puissance électrique lorsqu'elle est utilisée. De même pour le GPS et la 3G.

De ces observations, nous pouvons également noter que la consommation de puissance dans l'état *idle* de chaque composant hardware est pratiquement stable [119, 120]. Cette information conforte notre hypothèse sur la forte corrélation entre les activités de l'utilisateur et la consommation d'énergie de son système.

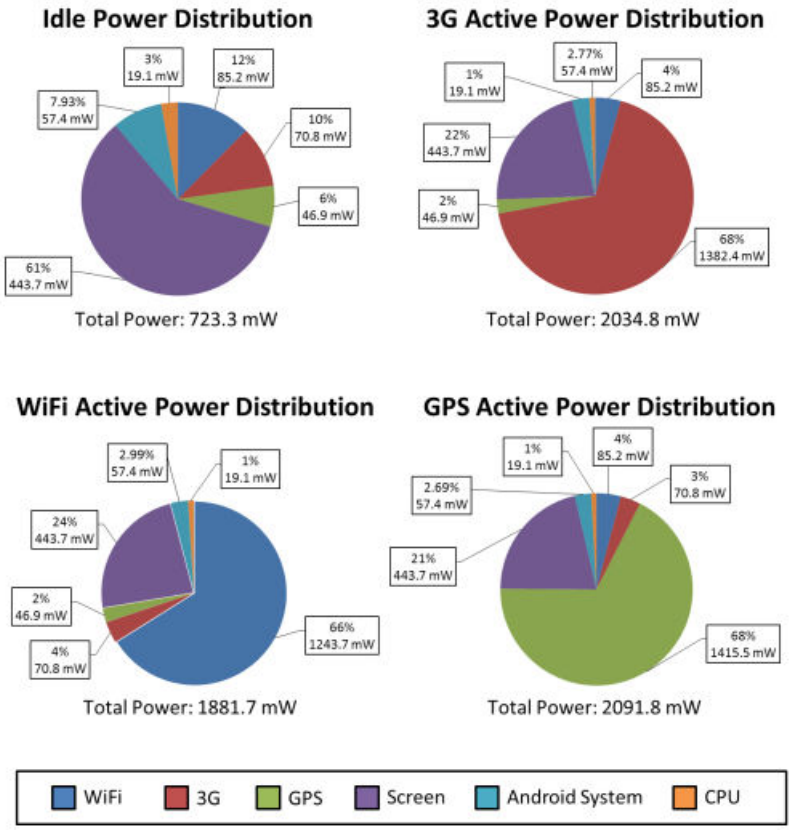

FIGURE 1.5 – La consommation énergétique de différents composants hardware dans Google Nexus One smartphone

1.4 L'étude des activités de l'utilisateur

Après avoir observé l'impact des activités de l'utilisateur sur la consommation d'énergie des systèmes mobiles, un certain nombre de travaux de recherche [20, 23, 119, 120] ont souligné l'importance de l'utilisation de ces activités pour

guider l'optimisation de la consommation d'énergie. Ces travaux reposent principalement sur le fait que la quantité d'énergie consommée est fortement liée à l'état actif du système et de ses composants hardwares.

Généralement, les auteurs étudient les activités de l'utilisateur pour les analyser et les caractériser. Cette étude est réalisée dans un environnement réel représentant le fonctionnement du système mobile sans influencer l'expérience de l'utilisateur, les performances du système ou encore la qualité du service (QoS).

D'après les différents travaux que nous avons pu étudier, nous avons remarqué que l'étude des activités de l'utilisateur se fait en plusieurs étapes. Le nombre d'étapes varie en fonction de la façon avec laquelle les activités de l'utilisateur vont être exploitées. La figure 1.6 schématise ces différentes étapes dont le nombre n'est pas figé et peut varier. Nous avons principalement:

- La sélection des composants hardwares ciblés dans l'étude;
- La collecte des activités de l'utilisateur final;
- L'analyse des activités collectées;
- Exploitation des résultats de l'analyse de deux manières possibles:
  1. Modélisation de la consommation d'énergie;
  2. Proposition de techniques d'optimisations de la consommation d'énergie;

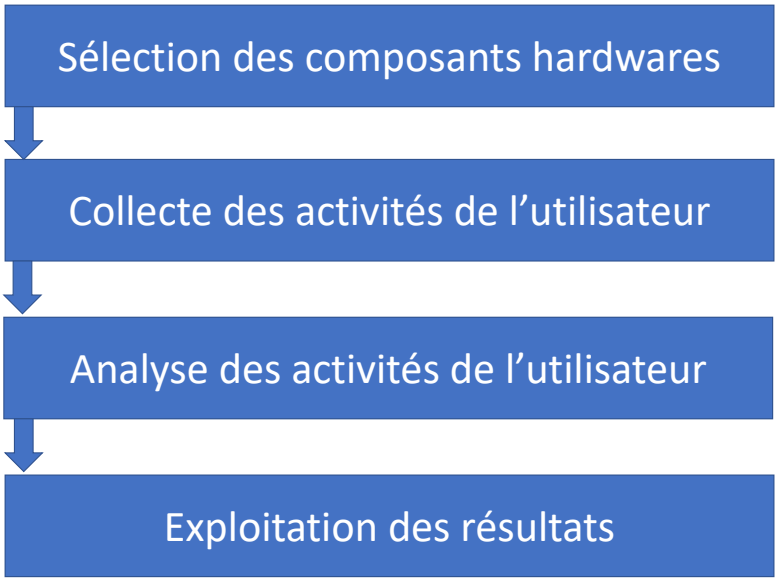

FIGURE 1.6 – Les étapes de l'étude des activités de l'utilisateur

1.4.1 La sélection des composants hardwares

Dans cette étape, une sélection d'un ensemble de composants hardwares est réalisée dans le système ciblé. Généralement, une étude détaillée de l'architecture du système est effectuée en amont. Pour chaque composant choisi, un

ensemble de paramètres est sélectionné comme illustré dans la tableau 1.1. Le tableau ci-dessous montre un exemple des paramètres considérés comme triviaux dans l'étude du CPU et l'écran. Nous pouvons par exemple estimer la consommation d'énergie du CPU en se focalisant sur son taux d'utilisation et sa fréquence.

Tableau 1.1 – Exemples de paramètres utiles pour l'étude du CPU et de l'écran

Composants	Paramètres
CPU	CPU Utilization (%)
	CPU Frequency (MHz)
Screen	Brightness level (%)
	On / Off State

1.4.2 La collecte des activités de l'utilisateur

Après la sélection des composants hardwares à étudier, une opération de collecte des activités de l'utilisateur liées à ce composant est effectuée. Pour réaliser cette tâche, les travaux de recherche utilisent généralement une journalisation de ces activités. Les données collectées sont ensuite stockées dans un serveur distant via une architecture client-serveur. Cette phase de collecte doit obéir à deux conditions essentielles:

1. Le respect de la confidentialité des données de l'utilisateur et la non-altération de sa satisfaction;
2. La phase de collecte doit avoir un impact minimal sur la consommation de ressources hardwares;

1.4.3 Analyse des activités de l'utilisateur

Cette phase est généralement expérimentale et permet de réorganiser les activités collectées pour chaque composant hardware. L'objectif est d'identifier les activités et les composants qui consomment le plus d'énergie. On peut y trouver également des études sur une potentielle relation entre la consommation des différents composants hardwares et les activités engendrant cette consommation. Les résultats de cette phase sont généralement des patterns d'activités de l'utilisateur. Ces patterns désignent un modèle d'utilisation de la plateforme observé de façon récurrente chez les utilisateurs.

1.4.4 Exploitation des résultats de l'analyse

L'identification des patterns d'activités est suivie par leur exploitation en fonction des activités collectées. Cette exploitation est subdivisée en deux classes:

- La modélisation de la consommation d'énergie: les travaux de recherche de cette classe visent à exploiter les patterns des activités pour proposer

des modèles de consommation d'énergie des composants hardwares ciblés. Ici, un modèle est un ensemble d'équations permettant d'approximer la quantité d'énergie consommée par un composant hardware en fonction des activités de l'utilisateur.
- La proposition de politiques d'optimisations de la consommation d'énergie: les contributions de cette thèse appartiennent à cette classe [35, 36, 37, 38, 39].

Dans la section suivante, nous présentons brièvement les contributions de cette thèse et le modèle de base mis en place.

1.5 Les contributions de cette thèse

Comme mentionné précédemment, le revers de la médaille de la popularité des systèmes mobiles est leur consommation d'énergie. Cette dernière reste une problématique récurrente soulignée par les travaux de recherches industrielles et académiques. Ces travaux se sont intéressés aux différents niveaux présentés dans la partie précédente en se focalisant sur l'impact des composants hardwares ou softwares existants sur la consommation globale du système. L'utilisateur final ainsi que ses besoins et ses activités ont rarement été pris en considération dans les différentes approches proposées comme il est souligné dans [20]. Cependant, la consommation énergétique d'un système mobile est fortement corrélée à la manière dont la plateforme est utilisée et exploitée. C'est au final, cet utilisateur qui détermine les fonctionnalités sollicitées et les applications lancées.

Dans le cadre de cette thèse, nous nous focalisons principalement sur l'utilisateur, ses besoins et ses habitudes. La question posée est comment pouvoir mettre en place une solution basée sur l'utilisateur ? C'est là où nous introduisons le deuxième axe traité dans nos travaux qui est la donnée dans son sens le plus large. En effet, ces dernières années, la donnée est considéré comme le nouveau pétrole avec l'avènement des réseaux sociaux, du *cloud computing*, le *big data* et les techniques d'intelligence artificielle. Comme mentionné précédemment, une des caractéristiques des systèmes mobiles actuels est le fait qu'ils comportent une large panoplie de capteurs embarqués porteurs d'informations sur l'utilisateur et son environnement. En addition à des interfaces de programmation qui permettent de collecter des informations sur l'utilisation du système, les applications utilisées, l'état des ressources hardwares, etc.

Dans notre travail, nous essayons de tirer profit de ces masses de données qui circulent dans les systèmes mobiles pour caractériser les besoins de l'utilisateur, ses habitudes et son interaction avec ses applications. Nous utilisons ensuite ces informations pour réduire la consommation de puissance de la plateforme mobiles en se concentrant sur un ensemble de composants hardwares. Pour tirer profit des données massives présentes dans les systèmes mobiles à des fins de réductions de consommation d'énergie, nous avons mis en place le modèle Collect Process Adjust (CPA).

Ce modèle est composé de trois phases communicantes qui sont:
- Collect: la première phase de notre modèle consiste à collecter différentes données disponibles dans le système mobile. Ces données sont accessibles via les capteurs embarqués et les interfaces de programmation ex-

posées par l'OS. Nous avons des données qui sont variées et hétérogènes et qui peuvent être par exemple, les applications lancées, l'utilisation des différentes ressources matérielles ou encore des informations sur l'environnement de l'utilisateur telles que le bruit ambiant ou la lumière ambiante de la pièce.

- Process: cette seconde phase prend en amont les données collectées précédemment pour réaliser un traitement dessus et en sortir avec des informations hétérogènes sur l'utilisateur, ses applications, ses habitudes et ses besoins. Cette phase est au cœur de notre modèle CPA et est considérée comme la phase la plus complexe et novatrice de nos travaux. C'est dans cette phase que nous exploitons une seconde discipline abordée dans cette thèse qui est le traitement des données et plus précisément les techniques d'intelligence artificielle et de fouille de données.
- Adjust: cette phase représente l'ajustement des ressources hardwares sur la base des résultats de la phase de traitement précédente. Par ajustement, nous voulons dire les contrôler et ne fournir à l'utilisateur et aux applications uniquement les configurations hardwares nécessaires.

Le schéma CPA est schématisée dans la figure 1.7 ci-dessous.

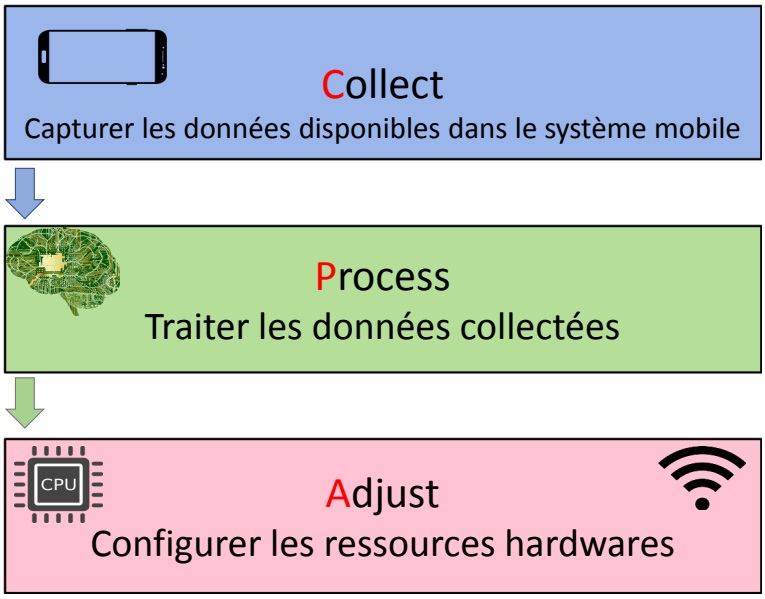

FIGURE 1.7 – Modèle CPA pour l'optimisation de la consommation énergétique des systèmes mobiles

Pour résumer, toutes nos contributions suivent le modèle CPA présenté ci-dessus. Le modèle mis en place nous permet de traiter la problématique de la consommation d'énergie en se focalisant sur les habitudes et les besoins de l'utilisateur. Nous nous positionnons dans nos travaux par rapport à la gestion proposée par le système d'exploitation. L'objectif étant d'améliorer la politique standard de gestion d'énergie disponible dans l' OS et exploiter les opportunités de réduction qui ne sont pas prises en considération par ce dernier.

1.6 Plan du mémoire

Ce manuscrit de thèse est composé de trois parties et sept chapitres. La première partie regroupe le contexte de notre étude. Un état de l'art est donné en se focalisant sur les solutions académiques et industrielles qui ont tenté de traiter la problématique de la consommation d'énergie dans les systèmes mobiles. La seconde partie détaille chacune de nos contributions. La troisième partie conclut ce document.

1.6.1 Première partie: contexte et état de l'art

- Le chapitre actuel 1 qui présente le contexte et la problématique.
- Le chapitre 2 présente un état de l'art concernant l'ensemble de nos contributions. Nous présentons également un large panorama des solutions software existantes.
- Le chapitre 3 présente l'environnement expérimental pour la gestion d'énergie, sur lequel reposent toutes nos contributions.

1.6.2 Deuxième partie: contributions

- Le chapitre 4 présente notre première contribution sur la gestion d'énergie qui est le composant SBOC pour **Sensor Based Optimization Component**. Ce composant utilise uniquement les données des capteurs embarqués pour la configuration des sous-composants hardwares. Avec SBOC, les données sont traitées de manière dynamique sans phase d'apprentissage.
- Le chapitre 5 présente notre second composant logiciel ANBOC pour **Application Needs Based Optimization Component**. Dans cette contribution, nous nous focalisons sur les applications lancées par l'utilisateur pour améliorer la gestion d'énergie proposée par le système d'exploitation. Avec ANBOC, nous réalisons une classification *offline* des applications. Nous mettons en place également un mécanisme de prédiction des futures applications lancées par l'utilisateur. La classification combinée à la prédiction permettent d'anticiper les besoins des applications en ressources. Menant ainsi à une économie d'énergie.
- Le chapitre 6 présente Enormous pour **ENergy Optimization for MObile plateform using User needS**. Ce composant se base sur le contexte de l'utilisateur en prenant en compte toutes les informations susceptibles de guider une politique d'optimisation de puissance. Nous y procédons à une classification dynamique de chaque contexte en fonction de ses besoins en configuration hardware et sa satisfaction. Le contexte de l'utilisateur englobe les informations sur l'utilisateur, les applications, l'environnement et le système. Enormous tend à être plus généraliste et complet que les solutions précédentes.
- Le chapitre 7 présente URBOC pour **User Request Based Optimization Component**. Ce composant logiciel est basé sur Enormous et sur le mécanisme de prédiction d'ANBOC. Dans cette contribution, nous avons mis en place un service inexistant dans les OS mobiles actuels. Ce service donne la possibilité à l'utilisateur d'envoyer des requêtes temporelles dans le but de rallonger la durée de vie de sa batterie jusqu'à une

heure souhaitée.
Le tableau 1.2 résume nos différentes contributions.

Tableau 1.2 – Synthèse des contributions de cette thèse

Contrib.	HW ciblé	Description
SBOC	*écran et hauts-parleurs*	- Approche dynamique sans apprentissage basée sur les données des capteurs de l'environnement
ANBOC	*CPU et Wi-Fi*	- Approche dynamique axée sur les applications; - Classification supervisée des applications; - Data mining pour la prédiction des applications;
Enormous	*CPU, écran, hauts parleurs et Wi-Fi*	- Approche englobante axée sur le contexte de l'user; - Apprentissage supervisé pour la classification du contexte de l'user;
URBOC	*CPU, écran, hauts parleurs et Wi-Fi*	- Extension d'Enormous basée sur les requêtes de l'user et la prédiction des applications; -Utilisation de la classification et du data mining

1.6.3 Troisième partie: conclusion et perspectives

- Le chapitre 8 conclut le manuscrit en résumant les différentes contributions. Nous présentons également les différentes perspectives et le travail actuel entamé avant la rédaction de cette thèse.

Chapitre 2

État de l'art

Dans le chapitre précédent, nous avons mis l'accent sur l'importance et l'émergence des systèmes mobiles, qui prennent une place considérable dans notre vie. Ces plateformes sont devenues indispensables pour beaucoup d'entre nous et leur nécessité au quotidien ne fait qu'accroitre. L'évolution de ces systèmes offre de nouvelles fonctionnalités et des catalogues d'applications diverses pour le plus grand plaisir des utilisateurs. Cependant, cette évolution engendre une importante consommation d'énergie, réduisant ainsi considérablement la durée de vie de la batterie. Ce qui fait de cette problématique un sujet de plus en plus récurrent aussi bien pour les grandes firmes de la *Silicon Valley*, que pour les chercheurs que nous sommes.

En vulgarisant, le principe devient assez simpliste, moins nous rechargeons notre système mobile, plus nous en sommes satisfaits. Cependant, étant donné que ces systèmes sont beaucoup sollicités, leur consommation d'énergie augmente considérablement. Nous nous retrouvons alors dans une relation conflictuelle entre durée d'utilisation et autonomie du système.

Pour contourner ce problème et prolonger l'autonomie des systèmes mobiles, plusieurs travaux de recherche ont mis l'accent sur l'optimisation de la consommation d'énergie en se focalisant sur tous les niveaux mentionnés dans le chapitre 1. Dans ce chapitre, nous nous concentrons principalement sur les différentes approches qui agissent au niveau logiciel. Nous discutons plusieurs approches allant du système d'exploitation jusqu'à l'utilisateur final, en passant par les interfaces de communication, les capteurs embarqués et le *cloud computing*.

Ce chapitre est organisé comme suit:

1. La section 2.1 présente les systèmes d'exploitation qui prennent en compte la consommation d'énergie, les profileurs d'énergie et les moniteurs de ressources.
2. La section 2.2 traite les travaux qui ont été réalisés sur les interfaces et les protocoles de communication en vu de réduire la consommation d'énergie.
3. Dans la section 2.3, nous présentons les solutions qui se sont focalisées sur l'optimisation des capteurs embarqués et plus précisément, les capteurs de géolocalisation.
4. Dans la section 2.4, nous discutons les approches visant à réduire la consommation d'énergie en exploitant le *cloud computing*.
5. Dans la section 2.5, nous détaillons les travaux qui se sont intéressés aux activités de l'utilisateur pour améliorer et modéliser la consommation d'énergie.
6. La dernière section 2.6 conclut ce chapitre.

Sommaire

2.1 Les systèmes d'exploitation soucieux de la consommation d'énergie . . . **20**
2.1.1 Les systèmes d'exploitation soucieux de l'énergie . . 20
2.1.2 Les profileurs d'énergie et les gestionnaires de ressources . . . 23
2.2 Optimisation de la consommation d'énergie des interfaces et protocoles de communication . . . **26**
2.3 Optimisation de la consommation d'énergie des capteurs embarqués des plateformes mobiles . . . **28**
2.4 Travaux centrés sur l'offloading computing . . . **32**
2.5 L'importance des activités de l'utilisateur dans la consommation d'énergie . . . **33**
2.5.1 Interaction de l'utilisateur avec les applications et les ressources . . . 34
2.5.2 Interaction avec la batterie et les cycles de recharge . 38
2.5.3 L'exploitation des activités de l'utilisateur pour la modélisation d'énergie . . . 40
2.6 Conclusion . . . **42**

2.1 Les systèmes d'exploitation soucieux de la consommation d'énergie

Nombreux sont les travaux qui se sont intéressés à la gestion de l'énergie dans les systèmes mobiles en déléguant cette tâche aux OS. Dans cette section, nous présentons les travaux qui ont été menés pour la mise en place de systèmes d'exploitation mobiles sensibles à l'économie d'énergie, les mécanismes d'ordonnancement et les profileurs d'énergie non intrusifs. Pour finir, nous discutons les avantages de la collaboration entre les applications et les OS pour aboutir à une meilleure gestion d'énergie.

2.1.1 Les systèmes d'exploitation soucieux de l'énergie

Le concept de système d'exploitation soucieux de l'énergie a été proposé à la fin des années 90 avec des OS conçus pour les ordinateurs portables comme *Odyssey* [92] et *ECOSystem* [42]. En 2000, M. Ellis a souligné que l'énergie devrait être considérée comme une ressource de première classe en plus de la perspective traditionnelle de maximisation du rendement fondée sur l'OS [132]. Bien que ce sujet ait été presque abandonné au milieu des années 2000, il a récemment attiré l'attention des chercheurs en raison des limitations énergétiques des systèmes mobiles actuels. En effet, nous retrouvons aujourd'hui des applications énergivores qui peuvent réduire la durée de vie des batteries à quelques heures de fonctionnement. Motivant ainsi la mise en place de systèmes d'exploitation mobiles soucieux de l'énergie tels que *Cinder* [112] et *ErdOS* [133]. Deux propositions s'opposent quant à savoir comment et par qui les politiques de gestion d'énergie dans les systèmes mobiles doivent être assurées.

D'une part, certains auteurs suggèrent que les applications doivent s'adapter dynamiquement aux limitations énergétiques comme dans *Chameleon* [81]. Toutefois, cette approche manque d'une entité centrale chargée de surveiller toutes les consommations de ressources engendrées par d'autres applications.

D'autre part, nous avons une autre école qui suggère que la gestion des ressources et de l'énergie devrait être réalisée entièrement au niveau de l'OS. Cependant, cette solution peut présenter des problèmes de flexibilité.

Odyssey et *ECOSystem* présentent une solution intermédiaire. Ils suivent une approche hybride dans laquelle les applications et le système d'exploitation collaborent pour réduire la consommation de puissance de la plateforme. Idéalement, l'OS doit connaître les besoins des applications et les ressources disponibles jusqu'à la prochaine recharge de la batterie afin de réduire la consommation d'énergie tout en maximisant l'expérience de l'utilisateur. Cependant, de nouveaux modèles de programmation, des planificateurs, des outils de mesure de l'énergie, des profileurs de performances et des Application Prgoramming Interface (API) doivent être développés afin de soutenir la gestion de l'énergie à ce niveau.

En 1995, Noble et al. ont introduit *Odyssey* [92] qui est un OS conscient de l'énergie basé sur *Linux*. Dans ce système, les applications peuvent adapter la qualité du service fourni aux utilisateurs en fonction de l'énergie et des ressources hardwares disponibles. *Odyssey* surveille les demandes de ressources et fournit une API aux applications afin de permettre un canal de communication bidirectionnelle avec l'OS. Cela permet de connaitre la disponibilité et la demande de ressources. En d'autres termes, la gestion de l'énergie est basée sur l'estimation des besoins énergétiques futurs afin de déterminer pour chaque application la quantité d'énergie consommée pendant une période de temps donnée [49]. Pour ce faire, *Odyssey* utilise le profileur d'énergie en ligne *PowerScope* [48].

ECOSystem est un autre système d'exploitation basé sur Linux qui alloue de l'énergie à des tâches concurrentes. Cet OS tient en compte l'équité entre les applications et le compromis entre performance et consommation d'énergie. Dans *ECOSystem*, l'énergie est équitablement allouée à de multiples composants matériels et applications. Avec ce modèle, les auteurs visent à atteindre un objectif d'augmentation de la durée de vie de la batterie en utilisant le taux de déchargement de la batterie comme indicateur, contrairement à *Odyssey* qui utilise un modèle de puissance.

Les caractéristiques de ces deux systèmes d'exploitation permettent de définir des politiques qui dégradent sélectivement le niveau de service des applications. Cela a pour objectif de préserver la capacité énergétique pour des tâches plus importantes. Néanmoins, cette approche suscite quelques critiques. Par exemple, Flinn et Satyanarayanan affirment qu'*ECOSystem* détériore les performances en supprimant l'ordonnancement des applications.

Nous pouvons considérer que nos contributions appartiennent à cette classe. Le point en commun réside dans la volonté d'améliorer la gestion d'énergie du système d'exploitation. Au lieu de proposer un nouvel OS soucieux de l'énergie, nos différentes contributions peuvent être intégrées aux systèmes d'exploitation existants comme des sur-couches logicielles.

Contrairement aux approches sus-mentionnées, nous ne réduisons pas la

qualité de service des applications mais nous essayons de la maintenir pour ne pas impacter la satisfaction de l'utilisateur. Dans le chapitre 5, nous réalisons la prédiction des futures applications pour leur allouer les ressources nécessaires à leur bon fonctionnement sans réduire le QoS.

Focalisons-nous maintenant sur les systèmes d'exploitation mobiles soucieux de l'énergie. *Cinder* [112] est un OS mobile conçu sur la base l'*Exokernel HiStar* qui exploite la modélisation de la puissance. Il a été testé sur des appareils HTC Dream mais nécessite la construction d'un modèle énergétique hors ligne du système ciblé comme dans *Odyssey*. *Cinder* assure le suivi des applications et des services responsables de l'utilisation des ressources. Il vise à assurer une répartition efficace de l'énergie en assurant l'isolement et la subdivision des applications.

Les auteurs affirment que cette approche permet également de détecter les logiciels malveillants et les applications défectueuses. Leur solution peut facilement limiter l'accès des applications aux ressources hardwares et à l'énergie. Comme dans *ECOSystem*, *Cinder* alloue l'énergie aux applications en utilisant deux abstractions appelées *reserve* et *taps* pour former un graphique de la consommation des ressources. Dans *Cinder*, la batterie du système est représentée dans le graphique des ressources comme étant la réserve racine. Lorsqu'une application consomme une ressource, le noyau de *Cinder* réduit les valeurs dans la réserve correspondante. Son ordonnanceur n'autorise l'exécution des threads que s'ils ont suffisamment de réserve pour fonctionner.

CondOS est un système d'exploitation contextuel qui tient compte des économies d'énergie [26]. Dans cet article, les auteurs affirment qu'au lieu de permettre aux applications de gérer de façon indépendante, mais inefficace, les capteurs disponibles, le système d'exploitation doit en être responsable. Cette prise en compte pourrait apporter différents avantages tels qu'une meilleure gestion de la mémoire, un meilleur planning et une meilleure sécurité ainsi que des économies d'énergie.

Enfin, il convient de mentionner un modèle d'allocation d'énergie intéressant et totalement décentralisé qui a été proposé pour *Nemesis OS* [91], qui est un système d'exploitation conçu pour les applications multimédia. Dans ce travail, les auteurs décrivent comment répartir l'énergie à l'aide d'un modèle de micro-économie. Le système attribue l'énergie aux applications en fonction de leur coût et de leur utilité, mais il tient également compte de l'encombrement des ressources du système. Les applications ont un crédit défini à utiliser lors de l'accès aux ressources, ce qui les motive à adapter leur consommation d'énergie.

L'inconvénient majeur d'une telle approche est que le système d'exploitation s'immisce considérablement dans l'expérience de l'utilisateur. Une telle approche risque d'engendrer une forte insatisfaction des utilisateurs, les dissuadant ainsi d'exploiter de telles solutions.

Le tableau 2.1 ci-dessous résume les OS soucieux de l'énergie.

Tableau 2.1 – Systèmes d'exploitation soucieux de l'énergie

Articles	Nom	Description
[144], [132]	***EcoSystem***	OS hybride soucieux de l'énergie, basé sur un ordonnanceur d'énergie
[92] [49]	***Odyssey***	OS hybride basé sur Linux qui adapte la QoS des applications selon demande d'énergie.
[112]	***Cinder***	Système d'exploitation hybride mobile basé sur HiStar OS. Il permet aux utilisateurs et aux applications de contrôler et de gérer les ressources hardwares.
[133]	***ErdOS***	OS mobile contextuel construit comme une extension d'Android OS. Il exploite les techniques de gestion proactive des ressources et permet un accès opportuniste et transparent aux ressources distantes.
[26]	***CondOs***	Architecture d'OS contextuelle pour une gestion efficace des capteurs.

2.1.2 Les profileurs d'énergie et les gestionnaires de ressources

Les OS traditionnels exécutent la charge de travail jusqu'à ce qu'ils atteignent le niveau de performance maximal. Ils passent ensuite en mode de faible puissance ou au niveau de performance le plus bas pour réaliser des économies d'énergie. Il existe des façons plus efficaces de gérer les ressources d'un point de vue énergétique. Cependant, cela exige une connaissance précise de toute tâche exécutée à l'intérieur du système afin de cerner la demande en ressources. Cette information a été cruciale dans la mise au point de nos différentes contributions. Ces travaux nous ont fait prendre conscience de la nécessité d'avoir une vision globale de ce qui se passe dans le système et également dans le comportement de l'utilisateur et son environnement.

2.1.2.1 Les profileurs d'énergie

Les profileurs d'énergie se sont avérés très utiles dans le domaine de l'analyse des performances. Cette section montre comment ces moniteurs sont exploités pour étudier les schémas d'utilisation de l'énergie. Nous abordons les différents types de profileurs d'énergie et les techniques utilisées. Des systèmes similaires peuvent être trouvés dans la section 2.4 afin d'estimer le coût de l'exécution d'une tâche localement ou à distance.

Odyssey OS utilise *PowerScope* [48] pour cartographier la consommation d'énergie avec la structure du programme de la même manière que les profileurs de CPU, comme le permet l'utilisation de la commande *prof* dans les machines *Unix*. Le *mapping* réalisé peut aider à identifier les applications qui sont énergivores.

PowerScope combine des techniques nécessitant un appareil de mesure de puissance externe et une seconde machine pour réduire les surcoûts d'énergie et toute interférence possible pendant l'étape de profilage. *Powerscope* utilise un échantillonnage statistique pour recueillir les traces. Cette approche n'est pas évolutive et nécessite d'être réitérée pour chaque composant hardware. Néanmoins, ce type d'outil présente deux avantages: il permet aux développeurs d'optimiser des applications afin d'atteindre les objectifs de conception et il permet également au système d'exploitation de gérer plus efficacement les ressources. *PowerScope* estime la consommation d'énergie des applications en mesurant le temps passé dans chaque état, tout en comptant le nombre de transitions d'états.

Une évolution de *PowerScope* est décrite dans [90]. Dans ces travaux, les auteurs introduisent un système prédictif pour gérer les ressources de manière proactive. L'outil de suivi des ressources exploite des techniques simples d'apprentissage. Cela permet au moniteur de ressources de traiter un plus grand nombre d'adaptations en augmentant la précision en une seule étape.

Joule Watcher [9] est un profileur d'énergie qui fonctionne par événement et qui estime également la consommation d'énergie au niveau des threads. Pour ce faire, les auteurs utilisent des compteurs embarqués dans les drivers matériels ciblés. Ces compteurs servent à enregistrer les événements impliquant une consommation d'énergie en surveillant le CPU, la batterie et la mémoire. Ces paramètres sont accessibles dans le thread et depuis le contexte du système, ce qui permet d'estimer la consommation d'énergie des threads de manière individuelle. De toute évidence, en raison du nombre limité de ressources qu'il contrôle, son intégration réelle dans un système d'exploitation moderne soucieux de l'énergie peut être compromise.

2.1.2.2 Les gestionnaires de ressources HW

En ce qui concerne la gestion des ressources HW, les travaux antérieurs ont décrit que l'efficacité énergétique doit être réalisée du côté des applications en réalisant des transformations pour augmenter les économies d'énergie possibles. Les politiques de gestion de puissance quant à elles, sont mises en œuvre dans l'OS. Heath et al [60] ont décrit comment les transformations d'applications peuvent augmenter les temps d'inactivité des systèmes mobiles en informant l'OS de la durée d'une prochaine période d'inactivité.

Self-tuning power manager (STPM) [4] est un middleware de gestion des ressources sensible à l'énergie centré sur les périphériques d' E/S. *STPM* exploite la collaboration entre les applications et l'OS pour fournir une mise en cache sensible à l'énergie. L'idée est que les applications divulguent des indices fantômes sur leurs intentions d'utiliser un périphérique d' E/S donné, alors que le système expose des informations contextuelles sur son état général aux applications. Grâce à ces connaissances, *STPM* adapte la stratégie de gestion de l'énergie aux patterns observés à partir des applications. L'auteur affirme que sa solution prend de meilleures décisions en matière de gestion d'énergie que les autres solutions parce qu'elle tente d'obtenir la meilleure combinaison de performance et d'économie d'énergie possible, tout est tenant compte de la charge de travail spécifique de l'application.

De même, *CIST* [74] montre qu'il est possible de réaliser des économies d'énergie plus importantes en prenant en compte les besoins des applications plutôt que d'analyser le comportement des ressources hardwares. Les auteurs soulignent qu'il n'est pas nécessaire d'apprendre des applications comme le fait le STPM [4]. *CIST* exploite les techniques de gestion dynamique de la puissance (DPM) [102] pour réduire la consommation d'énergie au cours de l'exécution.

Koala[122] est une approche proactive qui consiste à allouer les ressources hardwares en se basant sur la prédiction de la performance et de la consommation d'énergie de chaque application. Cela est réalisé en collectant des statistiques sur ces applications. Lors de chaque événement d'ordonnancement, le comportement de l'application est associé à la politique de gestion de puissance du système. Cela a pour but de déterminer l'état de fonctionnement le plus approprié dans le but d'une réduction énergétique. Les auteurs ont utilisé une politique arbitraire pour démontrer qu'il est possible d'échanger dynamiquement la performance et les demandes énergétiques d'une application.

Contrairement à *Koala*, nous ne réalisons pas de politiques arbitraires pour adapter dynamiquement les performances d'une application donnée. Dans notre classification *offline* des applications selon leurs besoins en ressources hardwares, nous prenons en compte l'interaction entre l'utilisateur et l'application en cours d'exécution.

Le tableau 2.2 ci-dessous résume les gestionnaires de ressources HW.

Tableau 2.2 – Profileurs de ressources

Ref	Nom	Description
[48]	***PowerScope***	Profileur de ressources hybride qui cartographie la consommation d'énergie selon la structure du programme
[9]	***JouleWatcher***	Profileur de ressources piloté par événement au niveau des threads.
[4]	***STPM***	Solution middleware hybride pour Linux qui supporte la gestion adaptative de l'énergie basée sur les modèles de demande observés dans les applications.
[74]	***CIST***	Middleware hybride orienté applicatif qui adapte son comportement aux besoins de l'application. Axé sur les interfaces sans fil.
[122]	***Koala***	Moniteur de ressources proactif

Enfin, nous citons également *Llama* [8] qui est une autre technique de gestion adaptative des ressources. Elle permet d'estimer pour un utilisateur et un système donnés la quantité d'énergie disponible dans la batterie susceptible de rester inutilisée. En se basant sur cette prédiction, il ajuste la qualité du service des applications afin d'atteindre les besoins énergétiques prévus. Cependant, les auteurs n'ont pas correctement évalué les économies d'énergie potentielles. Ils se sont montrés plus intéressés par la façon dont leur solution modifie les patterns de rechargement des participants pendant leur expérience.

Dans la dernière contribution de cette thèse développée dans le chapitre 7, nous réalisons également une prédiction du niveau de la batterie restant dans le

but de satisfaire les contraintes temporelles de l'utilisateur. Sur la base de cette prédiction, nous ajustons les ressources hardwares et softwares pour maintenir la disponibilité du système jusqu'à l'heure demandée par l'utilisateur. Contrairement à *Llama*, notre approche est dynamique et contextuelle.

2.2 Optimisation de la consommation d'énergie des interfaces et protocoles de communication

L'efficacité énergétique pour l'envoi et la réception des données est un domaine de recherche qui fait l'objet d'une attention particulière. Ces approches sont utiles, par exemple pour la conception des futurs protocoles de communication, mais elles n'ont cependant pas un grand impact sur le développement actuel des applications. Dans cette section, nous nous concentrons sur le choix des interfaces sans fil, la planification de la transmission de données, la compression des données et leurs impacts sur la consommation d'énergie.

Les systèmes mobiles sont équipés de plusieurs interfaces sans fil avec des caractéristiques techniques et des profils énergétiques variés. D'un point de vue énergétique, il est inefficace de garder plusieurs interfaces sous tension alors qu'elles ne sont pas toutes utilisées. Par exemple, la consommation d'énergie augmente de 18,3 % lorsque l'Universal Mobile Telecommunication System (UMTS) et la Wi-Fi sont allumés, par rapport à l'UMTS seul [101].

Par conséquent, la décision de choisir une interface spécifique joue un rôle important. Rahmati et Zhong [107] se concentrent sur l'économie d'énergie en choisissant l'interface sans fil appropriée basée sur plusieurs facteurs contextuels. Le défi pour une consommation efficace de l'énergie disponible est de décider à quel moment la Wi-Fi doit être activée. Par conséquent, les auteurs formulent un problème de décision statistique, qui repose sur différentes informations contextuelles telles que le temps, l'historique et les conditions du réseau cellulaire. Avec leur approche, les auteurs améliorent la durée de vie moyenne de la batterie de 39%.

Afin de réduire la consommation d'énergie des smartphones, Taleb et al [125] proposent une commutation dynamique entre 4G et Wi-Fi. Les auteurs visent à passer efficacement d'un réseau cellulaire à une connexion Wi-Fi alternative. Ils ont donc développé des modèles énergétiques concernant la taille et le débit binaire du téléchargement. À l'aide de ces informations, une application a été développée pour basculer dynamiquement entre la 4G et la Wi-Fi. Par rapport à une utilisation de la Wi-Fi uniquement et de la 4G uniquement, des économies d'énergie s'élevant à 18% et 30% respectivement, sont possibles.

Comme les connexions cellulaires nécessitent plus d'énergie que les connexions Wi-Fi, il est avantageux d'utiliser la Wi-Fi pour télécharger des grandes quantités de données. En pratique, lorsque la Wi-Fi n'est pas omniprésente, la planification du transfert de données pourrait être appropriée. Particulièrement pour le streaming vidéo, le prefetching semble être une méthode prometteuse pour économiser de l'énergie. Par conséquent, Gautam et al. [52] ont analysé les économies d'énergie potentielles avec l'application Android *App Incoming*. Cette application permet le *prefetching* de flux vidéo en fonction du comportement de l'utilisateur lors des vidéos précédentes. Avec l'approche proposée, une connexion Wi-Fi est utilisée pour télécharger des vidéos à l'avance afin

d'éviter le streaming via 3G ou 4G. Les auteurs constatent une économie de 98% pour les interfaces de communication concernant le streaming.

Outre l'interface sans fil choisie, la puissance du signal émanant des réseaux cellulaires influe également sur la consommation d'énergie. Pour économiser l'énergie dans les réseaux 3G, Schulman et al [116] proposent un algorithme de planification qui utilise une puissance de signal 3G élevée. Les auteurs analysent la consommation d'énergie en fonction de l'intensité du signal. Ils constatent une consommation six fois plus élevée par bit transféré lorsque l'intensité du signal est faible. Pour tirer parti d'une puissance de signal élevée, les auteurs ont développé un algorithme de planification pour deux types d'applications. D'une part pour les applications de synchronisation, telles que le courrier ou les actualités et d'autre part pour les applications de streaming. Pour réaliser cette tâche, l'algorithme se base sur les mesures des signaux 3G cco. Pour les applications de mailing, des intervalles de synchronisation flexibles sont utilisés pour l'actualisation de l'application. Pour les applications de streaming, l'algorithme module le flux de trafic en fonction des caractéristiques énergétiques de l'interface cellulaire. Ainsi, l'algorithme proposé prend en compte l'énergie consommée par la communication et par la file d'attente. Avec leur simulation, les auteurs montrent des économies d'énergie allant jusqu'à 10% pour la synchronisation du courrier électronique et jusqu'à 60 % pour le streaming.

Pour réduire le trafic de fond inutile qui entraîne une consommation d'énergie supplémentaire notable, Burgstahler et al [21] analysent l'impact énergétique des notifications de clients *Push and Pull*. Selon le temps de latence acceptable sur les notifications des clients, une approche basée sur les notifications *Push* peut aider à économiser de l'énergie par rapport à une approche basée sur les notifications *Pull*. Dans [22], les auteurs montrent une potentielle économie d'énergie pouvant atteindre 7 % grâce à une approche de transition. Selon le contexte des utilisateurs, les auteurs suggèrent de passer d'un paradigme de notification à l'autre pour économiser de l'énergie et en même temps, réduire les délais de notification.

Pour réduire le temps d'activation d'une interface sans fil, la réduction de la quantité de données transférées pourrait être une solution réalisable. Par conséquent, Hans et coll [58] analysent les économies d'énergie potentielles par la transmission de données compressées tout en faisant appel aux services Web. Pour leur enquête, l'UMTS est utilisé. D'une part, la compression réduit le nombre de paquets de données et le temps de disponibilité de l'interface sans fil. D'autre part, la décompression nécessite plus de puissance de calcul et donc plus d'énergie. L'évaluation montre que pour les services Web récurrents, la consommation d'énergie est plus élevée avec compression en raison de l'énergie consommée par la file d'attente. Pour les invocations de services Web uniques avec des tailles de charge supérieures à 50 ko, les économies d'énergie avec des compressions sont substantielles. Les auteurs montrent des gains plausibles allant jusqu'à 21,5 %.

En résumé, il existe plusieurs solutions pour réduire la consommation d'énergie lors de la transmission des données. Tout d'abord, la Wi-Fi doit être utilisée pour les chargements et les téléchargements volumineux. Dans la mesure du possible, les zones à forte intensité de signal devraient être préférées pour la

communication cellulaire. En outre, pour les applications de synchronisation tolérantes aux délais et le streaming, il convient d'utiliser des algorithmes de planification appropriés. De plus, la compression des données peut être utilisée pour économiser de l'énergie. Cependant, pour la transmission fréquente de petites quantités de données, les développeurs d'applications doivent être conscients du gaspillage d'énergie dû aux niveaux de puissance élevés des interfaces sans fil.

Dans nos contributions, nous proposons une gestion intelligente de l'interface Wi-Fi. Nous ne prenons pas en compte la taille des données transmises, mais nous concentrons nos efforts sur l'état de l'antenne Wi-Fi en se basant sur les besoins des applications et le contexte de l'utilisateur.

2.3 Optimisation de la consommation d'énergie des capteurs embarqués des plateformes mobiles

La perception et la connaissance de l'environnement de l'utilisateur ont joué un rôle fondamental dans le développement des applications mobiles au cours des dernières années. En effet, les applications mobiles ont souvent besoin de données de localisation pour mettre à jour les informations localement pertinentes. Cette localisation sert à fournir un service demandé par l'utilisateur, à trouver des amis proches ou à adapter le système afin de gérer efficacement les ressources. Cependant, l'accès aux ressources de géolocalisation peut être très coûteux en énergie. Par conséquent, la mesure continue de l'efficacité énergétique de la localisation est devenue un sujet de recherche incontournable, tant dans l'informatique ubiquitaire que dans les réseaux de capteurs.

Dans cette section, nous traitons les solutions au niveau du logiciel, bien que certains chercheurs tentent également de démontrer la nécessité de reconcevoir l'architecture du système au niveau hardware pour l'optimisation énergétique. Priyantha et al [104] proposent d'ajouter un micro-contrôleur à faible puissance supplémentaire dans le processeur multi-cœur responsable de la gestion des capteurs. Cette modification permet à la plupart des parties du système mobile d'entrer dans un état de veille à faible consommation d'énergie, tandis que le processeur du capteur est en train d'échantillonner et de traiter les données du capteur en continu.

Les systèmes mobiles modernes comprennent différents types de capteurs de localisation avec différentes résolutions et demandes d'énergie. Les capteurs GSM, Wi-Fi et GPS ont respectivement une erreur moyenne de précision de l'ordre de 400 m, 40 m et 10 m^{11}. Dans ces applications, le GPS est souvent préféré aux systèmes de positionnement basés sur la technologie GSM/WiFi pour sa précision élevée. En addition, les applications mobiles sont de plus en plus sensibles au contexte, en particulier à l'emplacement géographique. Par conséquent, des systèmes de localisation optimisés sur le plan énergétique sont devenus une demande des développeurs mobiles pour fournir des applications plus riches. Dans ce genre d'approches, il est possible de différencier trois types de solutions qui peuvent être complémentaires:

- Les systèmes qui combinent plusieurs capteurs pour réduire la consommation d'énergie tout en minimisant l'erreur.

- Les méthodologies qui s'appuient exclusivement sur des modèles probabilistes de localisation des utilisateurs. Ces modèles sont utilisés pour déduire les emplacements futurs et réduire l'accès en lecture des capteurs.
- Les solutions qui proposent des heuristiques pour adapter le taux d'échantillonnage des capteurs.

Ben Abdesslem et al [10] combinent les valeurs de l'accéléromètre et celles du GPS. L'idée principale est d'utiliser plus souvent des capteurs moins énergivores. En choisissant quand utiliser des capteurs plus économes en énergie, il est possible de réduire la consommation d'énergie des applications qui utilisent la localisation. Cependant, l'évaluation du système ne fournit pas de valeurs précises puisque les auteurs ont basé leurs résultats sur l'autonomie de la batterie. Ils ont fourni deux systèmes mobiles au même utilisateur. Le premier système utilise *SenseLess* qui est un module software mis en œuvre avec le *PyS60API* de *Python*. Le second système utilise périodiquement le GPS toutes les 10 secondes pendant 30 minutes environ. Les résultats indiquent que l'appareil utilisant *SenseLess* consomme 58,8% d'énergie en moins que celui qui échantillonne périodiquement le GPS. Cependant, la précision de la mesure est hautement altérée car le système considère des actions comme s'asseoir ou se lever comme des mouvements.

EnTracked [68] se penche sur le problème d'envoi des informations de localisation vers un serveur de localisation en ligne. *EnTracked* réalise cet envoi en respectant des limites d'erreur de positionnement spécifiques à l'application. *EnTracked* estime et prédit l'état du système et la mobilité de l'utilisateur pour planifier les mises à jour des positions afin de minimiser la consommation d'énergie tout en optimisant la robustesse. L' évaluation a été réalisée avec des simulations mais les résultats ont été validés dans des scénarios réels avec des appareils *Symbian*. Leurs résultats révèlent des économies d'énergie potentielles de l'ordre de 40% à 50% par rapport à un échantillonnage périodique avec une erreur maximale et peu probable de 200 m. *EnTracked* utilise l'incertitude estimée par le GPS pour programmer rapidement une nouvelle localisation si une mauvaise mesure potentielle est effectuée.

D'autres travaux comme [13] ont étendu *EnTracked* en proposant le concept de délimitation de situation [1]. Ce concept est utilisé pour améliorer et paramétrer la robustesse des stratégies de gestion des capteurs pour le suivi de trajectoire.

Un autre travail qui s'est focalisé sur les capteurs pour réduire la consommation d'énergie est *Jigsaw* [83]. Cette solution middleware améliore la résolution de plusieurs capteurs tels que l'accéléromètre, le microphone et le GPS tout en réduisant la consommation d'énergie nécessaire à la détection de l'environnement de l'utilisateur. Bien que *Jigsaw* soit centré sur les applications, il fournit des Application Programming Interfaces (APIs) spécifiques pour reconnaître les activités des utilisateurs à l'aide d'informations contextuelles et de techniques d'apprentissage automatique. Les auteurs affirment que l'une des tâches les plus consommatrices d'énergie lors de la détection de l'environnement se produit lors du traitement des données brutes des capteurs. Les auteurs affirment que leur solution peut effectuer un *tracking* GPS à long terme

1. situational bounding

peu couteux en énergie. Toutefois, l'article ne contient pas d'évaluation énergétique détaillée.

Dans [82], les auteurs se sont intéressés aux applications basées sur une détection collaborative de l'emplacement. Ces applications prennent en entrée les valeurs des capteurs à partir d'un certain nombre de systèmes mobiles, généralement répartis sur une zone géographique définie. Ces applications sont activées via le *cloud computing* en transférant des données brutes des capteurs à la logique applicative hébergée dans le *cloud*. Il en résulte un compromis entre la qualité des données reçues par les applications et l'énergie nécessaire pour le transfert des données. Les auteurs abordent ce compromis en envisageant un schéma dans lequel un middleware de détection collaboratif sert de médiateur entre les multiples applications nécessitant des données mesurées et les systèmes mobiles situés dans une zone physique particulière.

Les auteurs présentent un algorithme qui cherche à maximiser la mesure dans laquelle les données transférées d'un système mobile donné peuvent être utilisées par plusieurs applications. Les auteurs affirment que leur algorithme conduit à de meilleures performances globales en matière d'énergie consommée qu'un algorithme qui ne regroupe pas les informations détectées entre les applications.

Nous notons que ce travail peut appartenir à deux catégories qui sont l'optimisation des capteurs et l'utilisation du *cloud computing* pour la diminution de la consommation de puissance présenté dans la section 2.4. D'autres approches similaires qui tentent d'optimiser la consommation d'énergie des capteurs en exploitant les fonctionnalités du *cloud* sont proposées dans [15, 85, 140].

Dans [29, 88], Álvarez de la Concepción et al ont proposé une technique pour réduire la consommation d'énergie dans les systèmes mobile à accéléromètre. Cette réduction est obtenue en allégeant le moteur de classification des activités de l'utilisateur. Cette technique exploite le pré-traitement des données en discrétisant la plage de valeurs à l'aide de l'algorithme *Ameva*. L'étape du prétraitement génère une matrice de probabilités d'association pour chacune des activités à reconnaître. Cette matrice est utilisée par un système de vote pour sélectionner l'activité la plus probable réalisée par l'utilisateur, ainsi que sa fréquence. Si la probabilité pour toutes les activités est inférieure à un seuil prédéfini, le mécanisme peut la reconnaître comme une nouvelle activité. Comme pour de nombreux systèmes à base d'accéléromètres, le principal inconvénient est le nombre d'activités qui doivent être reconnues, puisque différentes activités peuvent montrer une corrélation dans les variables. Un second inconvénient est le capteur de position qui doit rester activé pendant l'utilisation du système. Selon l'expérimentation, la précision du système proposé peut atteindre jusqu'à 98%, épuisant la batterie après 18 h d'exécution continue.

Pour finir, nous avons dans la littérature un grand nombre de travaux qui se sont focalisés sur l'optimisation de la consommation d'énergie des capteurs dans le domaine de l'IoT [13, 53, 141]. Ces travaux ont tous en commun l'utilisation du *cloud computing* pour le traitement des données des capteurs afin de minimiser leur utilisation en local. Ces travaux sont regroupés et comparés dans le survey [99]. Le tableau 2.3 résume les travaux les plus importants de cette catégorie.

Tableau 2.3 – Optimisation des capteurs embarqués dans les systèmes mobiles

Art	Nom	Descrption
[10]	*Sens-Less*	- Géolocalisation efficace énèrgétiquement combinant l'accéléromètre et le GPS. - Il exploite le compromis entre la précision et l'énergie. - Evaluation avec un réel déploiement.
[27]	*EnLoc*	- Géolocalisation optimisée énergétiquement combinant l'accéléromètre et le GPS. - Il exploite les modèles d'arbitrage entre la précision et l'énergie et les modèles probabilistes de la mobilité humaine. - Évaluation par simulation.
[80]	*A-Loc*	Détection de localisation efficace énèrgétiquement tirant parti des modèles probabilistes et d'une combinaison de toutes les ressources de détection de localisation.
[68]	*EnTracked*	Estimation des conditions du système et de la mobilité pour planifier les mises à jour des positions aux services de localisation en ligne.
[96]	*RAPS*	Système de positionnement adaptatif utilisant un modèle probabiliste de mobilité de l'utilisateur, toutes les ressources de détection de localisation, la synchronisation opportuniste entre les utilisateurs et la surveillance de l'état des ressources.
[145]	*n/a*	Géolocalisation utilisant la suppression basée sur l'accéléromètre et l'adaptation des paramètres de détection en fonction de l'énergie.
[83]	*jigSaw*	- Cadre général pour la détection continue de toute ressource de détection. - Supporte l'accéléromètre, le microphone et le GPS.

Dans les contributions de cette thèse, nous n'avons pas travaillé de façon directe sur l'optimisation des capteurs embarqués. Néanmoins, nous avons jugé utile de présenter cette catégorie de travaux car elle est omniprésente dans la littérature. Nous nous sommes également inspirés de l'utilisation des algorithmes d'apprentissage et de classification présents dans cette catégorie comme la prédiction des futures emplacements de l'utilisateur. Les techniques d'apprentissage utilisées nous ont par exemple guidé pour réaliser la prédiction des futures applications de l'utilisateur.

Les travaux présents dans cette classe nous ont également guidé pour mettre en place un système qui permet de déduire le niveau d'activité physique de l'utilisateur. En effet, les valeurs de l'accéléromètre peuvent indiquer si l'utilisateur est immobile, en train de marcher ou courir.

Dans nos contributions, nous avons également utilisé des capteurs embarqués à faible consommation de puissance comme l'accéléromètre, le gyroscope

et l'inclinomètre. Ces capteurs ont servi à réduire la consommation d'énergie de composants qui sont plus énergivores tels que l'écran et la Wi-Fi. Cependant, nous envisageons de nous consacrer partiellement à l'optimisation du capteur GPS dans nos futurs travaux comme nous le mentionnons dans les perspectives.

2.4 Travaux centrés sur l'offloading computing

Le *cloud computing mobile* consiste à externaliser les traitements, les calculs et le stockage des données à l'extérieur des systèmes mobiles vers des plateformes puissantes et centralisées sur internet, en préservant l'interactivité des dispositifs mobiles. Le *cloud computing* peut apporter de nombreux avantages aux systèmes mobiles, comme la puissance de calcul et l'efficacité énergétique. Même à la fin des années 90, Vahdat et al [132] et Rudenko [113] ont considéré que l'énergie peut être considérée comme une autre ressource internet similaire au calcul. En effet, de plus en plus d'applications et même de services des OS s'appuient sur le *cloud computing*. Les travaux qui se sont intéressés aux systèmes mobiles et au *cloud computing* peuvent être divisés en deux classes qui sont:

- Les approches qui utilisent le *cloud* pour réduire la consommation énergétique des plateformes mobiles.
- Les travaux plus récents qui tentent d'optimiser la consommation d'énergie liée à l'utilisation du *cloud*.

Nous présentons dans cette section uniquement les travaux de la première classe qui utilisent le *cloud* comme moyen d'économie d'énergie. Comme mentionnée dans ce qui a précédé, une autre approche prometteuse pour l'économie d'énergie dans les systèmes mobiles est le transfert des tâches de calcul vers les serveurs distants. Actuellement, ce qui est très récurrent est de déplacer l'ensemble du calcul dans le *cloud* pour le traitement et renvoyer le résultat de ce traitement vers la plateforme mobile. L'envoi d'un flux vidéo à l'utilisateur via les plateformes de streaming, les jeux en ligne ou le service *Desktop-as-a-Service* en sont des exemples marquants. D'autre part, des parties d'applications spécifiques à forte intensité de calcul sont déchargées sur des serveurs distants. Ces serveurs sont proposés par des fournisseurs privés de *cloud computing*.

Kwon et Tilevich [73] étudient la possibilité de décharger des applications *Android* en déplaçant des fonctionnalités énergivores vers des serveurs distants sans partitionner les applications. Les auteurs identifient les fonctions énergivores et les envoient pour une délocalisation dans le *cloud*. Lors de l'exécution, il est possible de basculer entre l'exécution locale et distante. Cette approche se traduit par des économies d'énergie comprises entre 30 % et 60 % pour quatre applications *Android* tierces sur cinq en conservant les caractéristiques de performance d'origine. Néanmoins, l'utilisation de techniques courantes pour diviser l'application en différentes parties peut causer des problèmes en cas de panne de réseau.

Hans et al [57] analysent les effets des jeux dans le *cloud* sur la consommation d'énergie des smartphones. Les auteurs concluent que la durée de vie de la batterie est étendue lorsque des tâches de calcul intensives sont transférées vers des serveurs *cloud*. En particulier, pour les jeux à forte intensité graphique. Cette approche est prometteuse, cependant, le flux vidéo constant requis peut avoir

des effets négatifs sur la durée de vie de la batterie. Pour un environnement expérimental entièrement contrôlé, les auteurs ont développé leurs propres applications client/serveur. Ils ont comparé différentes complexités de jeu et différentes qualités vidéo en utilisant l'exécution locale et à distance. En utilisant le Wi-Fi pour la connexion à distance, les auteurs constatent des économies d'énergie pour les jeux dans le *cloud* entre 12% et 38%.

Les techniques d'offloading représentent une approche avec de nombreux atouts, en particulier pour les tâches de calcul intensives. Cependant, il y a plusieurs défis à garder à l'esprit. Premièrement, les applications peuvent nécessiter une connexion réseau disponible en permanence. Deuxièmement, en ce qui concerne l'énergie, il y a un compromis entre la réduction de la puissance de calcul et l'augmentation de l'utilisation du réseau. Enfin, en utilisant l'exécution à distance, la latence du réseau peut affecter la Qualité de Service (QoS) de l'application et par conséquent la satisfaction de l'utilisateur. Dans les contributions de cette thèse, nous ne sommes pas intéressés au *cloud computing*. Cependant, nous sommes actuellement en train de travailler sur une approche de délocalisation de nos algorithmes d'apprentissage et de classification vers un serveur basé dans le *cloud*. Cela a pour objectif de réduire les coûts des différentes solutions que nous proposons. Nous prévoyons également de proposer une approche multi-agent pour mettre en place un regroupement des utilisateurs avec les mêmes profils énergétiques.

2.5 L'importance des activités de l'utilisateur dans la consommation d'énergie

Les activités de l'utilisateur sont souvent ignorées par les approches d'optimisation de la consommation d'énergie [77]. Cependant, comme souligné précédemment, dans un système mobile, l'utilisateur est le premier responsable de la charge de travail des composants hardwares et du lancement des applications. En effet, c'est l'utilisateur qui détermine les fonctionnalités actives et les applications en exécution et en background. Par conséquent, il existe une très forte corrélation entre la consommation d'énergie et les habitudes de l'utilisateur. Partant de ce constat, quelques travaux de recherche ont initié la prise en considération des activités de l'utilisateur afin de pouvoir jauger cette concordance à des fins d'optimisations énergétiques.

Cette section est subdivisée en trois parties:

- La première partie présente les travaux de recherches qui se sont focalisés sur la collecte des données des utilisateurs. Nous y présentons également comment les données collectées sont utilisées pour guider l'optimisation de l'énergie.
- La seconde partie présente les travaux qui ont porté sur la compréhension des cycles de rechargement de la batterie.
- La troisième partie présente les travaux qui utilisent les activités de l'utilisateur pour la modélisation d'énergie.

2.5.1 Interaction de l'utilisateur avec les applications et les ressources

Comme nous l'avons déjà mentionné, une des exigences pour une gestion efficace de l'énergie réside dans la compréhension de quand et comment l'énergie est utilisée. Il est également nécessaire de savoir combien d'énergie est consommée par chaque partie du système. Les architectes des systèmes mobiles doivent tenir compte des patterns d'interaction entre l'utilisateur et le système. Cela servira à l'évaluation des différentes optimisations d'énergie et de leur impact sur l'expérience de l'utilisateur. Cependant, la surveillance de l'utilisation des ressources demeure une tâche difficile en raison de la grande diversité des applications et des composants disponibles dans les plateformes. Une autre raison qui rend cette tâche difficile est la nécessité de concevoir des outils de surveillance non intrusifs et respectueux de la vie privée des utilisateurs.

Certaines études sont basées sur les traces collectées directement sur le réseau [139]. Un des principaux avantages de ce type d'étude est qu'elles n'ont aucun impact sur les performances du système mobile. Elles peuvent également recueillir un plus grand nombre de traces. Cependant, à la différence des *loggers* d'arrière-plan fonctionnant sur le périphérique, ces solutions sont limitées quant aux informations collectées. Par exemple, Trestian et al [130] ont réalisé une étude à grande échelle sur l'utilisation du réseau mobile pour caractériser la relation entre les utilisateurs, les applications basées sur les requêtes URL et leurs schémas de mobilité. Leurs résultats démontrent que l'utilisation des applications est fortement corrélée aux habitudes et à l'emplacement des utilisateurs.

L'expérience la plus vaste réalisée d'un point de vue des utilisateurs finaux utilise des traces collectées à partir d'une application multiplateforme appelée *3GTest* [65] qui vérifie l'état des réseaux cellulaires et les performances réseaux des applications. Bien qu'il ne s'agisse pas d'un travail entièrement axé sur l'énergie, les traces de 30 000 utilisateurs mobiles dans le monde entier peuvent être pertinentes pour guider l'optimisation d'énergie. En effet, ces traces fournissent de riches informations pour comprendre la performance des réseaux cellulaires avant de concevoir des systèmes optimisés sur le plan énergétique. Avec les traces obtenues, les auteurs ont identifié des goulots d'étranglement dans le réseau sans fil ainsi que des limitations de performance et des bogues dans le système d'exploitation. Dans leurs résultats, les auteurs mentionnent également que les propriétés du réseau peuvent varier en fonction de l'heure et du lieu pour un opérateur donné. Toutes ces informations ont un impact direct sur la consommation d'énergie du système. Des résultats similaires ont été trouvés par Tan et al. [126] dans un scénario plus court et plus limité géographiquement.

Dans [47], les auteurs ont analysé les patterns de trafic mobile de 43 utilisateurs sur deux plateformes mobiles différentes à l'aide de *sniffers* de paquets. Leurs résultats indiquent que la quantité de trafic journalier générée par un utilisateur peut varier de 2 à 500 Mo et que la navigation elle même contribue pour plus de la moitié de la taille totale du trafic, impactant ainsi la consommation d'énergie du système. Ils se sont également penchés sur les inefficacités causées par la taille de transfert des paquets de taille médiane de 3 ko. Ce transfert entraîne un surcout énergétique considérable dans les protocoles de couches

inférieures qui varie de 12% à 40%, lors de l'utilisation des transferts sécurisés de données.

Les auteurs soulignent qu'une bonne connaissance des schémas de trafic dans les réseaux des systèmes mobiles peut permettre de nouvelles politiques de gestion de la radio. Ces politiques permettent d'économiser jusqu'à 35 % d'énergie en réduisant le temps d'inactivité, avec un impact minimal sur les performances du système.

D'autres solutions réalisent du monitoring sur les utilisateurs mobiles d'un point de vue plus général comme l'outil *LiveLab*. Cet outil est un enregistreur de ressources reprogrammable qui est basé sur les événements [117]. Les auteurs décrivent une méthodologie non intrusive pour mesurer l'utilisation réelle du système et des réseaux sans fil en tenant compte de la vie privée des utilisateurs. Ils ont mené des sondages pour comprendre les préoccupations des utilisateurs en matière de protection de la vie privée. Ces utilisateurs ont souligné leur besoin d'anonymat concernant les traces collectées.

Dans *LiveLab*, les auteurs décrivent comment un *logger* doit être conçu pour ne pas encourir de coûts énergétiques supplémentaires dans le système mobile. Ils estiment qu'un *logger* non intrusif doit être basé sur des événements pour éviter les sondages périodiques. Il doit également tenir compte de l'état des ressources et tirer parti de l'information déjà disponible dans le système. Le *logger* doit également profiter des mises en veille du système pour réduire les frais supplémentaires liés à la collecte de données. Cependant, l'article ne mentionne pas de détails sur le *logger* ni son coût énergétique.

Le document contient également une courte étude sur 25 utilisateurs d'*iPhone*. Les résultats montrent que l'interaction des utilisateurs avec le le système mobile et l'état des ressources dépendent d'informations contextuelles telles que le temps et l'espace. De plus, les modes d'utilisation évoluent également en fonction du temps. Cependant, l'analyse de l'interaction humaine est très simple et semble se concentrer exclusivement sur deux utilisateurs seulement.

Shye et al [119] ont mis au point un *logger* en background qui surveille périodiquement l'utilisation des ressources à 1 Hz pendant un usage normal du système mobile. Afin d'estimer la consommation énergétique spécifique à chaque application et ressource, ils ont utilisé un modèle énergétique construit en utilisant des techniques de régression linéaire comme nous le développons dans la section 2.5.3. Leurs résultats montrent que la consommation d'énergie dépend de la manière dont chaque utilisateur interagit avec son système. Les auteurs ont trouvé une grande variation dans la répartition de la puissance entre les utilisateurs et ils affirment que l'écran et le CPU sont les deux composants qui consomment le plus d'énergie, tandis que l'état inactif représente en moyenne 49,3 % de la puissance totale du système. Cependant, ces chiffres peuvent être biaisés par le *logger* en lui même et sa fréquence d'échantillonnage élevée. Ces résultats ont été présentés dans le chapitre précédent.

Une approche similaire est suggérée par Falaki et al. pour l'écran des périphériques Symbian dans [45]. Nous mentionnons également une approche décrite par Brakmo et al [17] pour configurer le CPU en mode veille pendant les petites périodes de temps où sa charge est faible, par exemple lorsque l'utilisateur lit un document ou consulte un site Web.

Vallina-Rodriguez et al. ont réalisé une étude en utilisant également un *logger* en background pour recueillir des traces de 18 utilisateurs d'Android pendant 2 semaines [134]. L'ensemble des données contient des informations contextuelles et plus de 25 statistiques sur l'usage des ressources et des applications, échantillonnées toutes les 10 secondes. Cette étude se distingue notamment des travaux précédents, car elle tente de comprendre les relations existantes entre les ressources causées par les patterns d'interaction des utilisateurs.

Les auteurs tirent profit des techniques d'apprentissage automatique pour mieux comprendre ces dépendances et voir comment l'information contextuelle et les habitudes des utilisateurs entraînent une demande de ressources. L'article démontre qu'il est possible de prédire comment les utilisateurs interagiront avec leurs appareils mobiles à partir des informations spatio-temporelles. Par conséquent, les auteurs affirment que, sur la base des résultats expérimentaux, une gestion algorithmique des ressources n'est pas réalisable pour les appareils mobiles et qu'il est nécessaire de trouver des moyens plus souples de gérer les ressources, qui peuvent s'adapter aux modes d'interaction des utilisateurs. Ils suggèrent qu'une gestion efficace des ressources dans les systèmes mobiles doit être proactive. Cette gestion se doit d'être centrée sur le système et sur l'utilisateur, tout en tirant parti des informations contextuelles et précises du système.

Ce travail réconforte notre hypothèse sur la nécessité de prendre en considération l'utilisateur, son système et son environnement pour proposer une gestion d'énergie intelligente et adaptative. En effet, une gestion algorithmique figée des ressource est à proscrire, étant donné les paradigmes d'utilisation des systèmes mobiles.

MyExperience [51] est un système permettant de capturer des données objectives et subjectives directement à partir d'appareils *Windows Mobile*. Dans cet article, les auteurs ne mentionnent pas exactement quel genre de traces ils recueillent et ne détaillent pas suffisamment leurs *logs*. Le travail décrit les étapes de conception et l'architecture du système ainsi qu'une évaluation des performances en matière de CPU, de mémoire et de durée de vie de la batterie. Les auteurs ont estimé que leur *logger* peut réduire la durée de vie de la batterie d'environ 12 %, malgré l'utilisation de déclencheurs contextuels pour échantillonner l'information. Ce qui différencie vraiment ce travail des autres, c'est qu'il demande les feedbacks des utilisateurs. Ces retours servent à comprendre l'expérience perceptible de l'utilisateur qui accède à un service spécifique. L'article présente également une brève étude sur les modes d'interaction de la batterie, la mobilité et les SMS de 14 utilisateurs pendant 1 à 2 semaines. Il est intéressant de noter que les résultats obtenus dans le présent document sont semblables à ceux de [8].

Dans nos travaux, nous avons conçu également un *logger* non intrusif. Notre collecteur de données est lancé en background pendant une période de deux semaines. Nous collectons différentes informations relatives à l'utilisateur, l'état du système et l'environnement. Comme dans [8], nous prenons également en compte les feedbacks de l'utilisateur pour ajuster nos politiques d'optimisation. Les retours de l'utilisateur sont détectés de façon implicite en analysant son comportement.

Nos phases de collecte de données sont développées dans la partie contribution. Le tableau 2.4 résume les principaux travaux mentionnés dans cette partie.

Tableau 2.4 – Travaux sur l'interaction de l'utilisateur avec les applications et les ressources hardwares

Art	Sys	Particip	Durée	Description
[130]	n/a	281 K	1 W	Pattern d'utilisation des applications et de la mobilité à partir des traces du réseau
[65]	iPhone, Android	30 K	n/a	Analyse des réseaux mobiles
[46]	Android	43	26 days	Analyse du trafic présent dans les smartphones
[117]	iPhone	25	10 W	Outil de monitoring pour iPhone et analyse d'utilisation
[119]	Android	20	12 days	Profiler de ressources et modèle de consommation d'énergie des composants hardwares
[47]	Android	33	7 - 28 W	Diversité dans les usages des systèmes mobiles et impact des données temporelles
[134]	Android	18	1 - 2 W	Impact des données spatio-temporelles dans la demande d'énergie Interdépendances des ressources causées par les patterns d'utilisation
[51]	WiMo	4-16	1 - 2 W	Profiler de ressources dans les systèmes mobiles prenant en compte les retours des utilisateurs

Dans [28], les auteurs présentent l'application *Power Monitor* qui a pour but de comprendre les patterns d'utilisation dans les systèmes mobiles et d'estimer la consommation énergétique. L'application collecte différentes données du système et les sauvegarde localement. Ensuite, un mécanisme d'apprentissage traite les données collectées pour générer plusieurs patterns d'utilisation, en prenant en compte l'aspect spatio-temporel. Le moniteur de l'application traite ensuite ces patterns d'utilisation pour générer des profils d'économie d'énergie adaptatifs. Les données qui sont collectées dans le cadre de ce travail sont : les applications exécutées, le niveau de la batterie, l'emplacement, le temps et les informations disponibles sur les interfaces de communication. Toutes ces données ont été collectées durant 7 jours. Les patterns ayant une durée de temps et une location similaire sont regroupés. Les auteurs estiment ensuite la consommation d'énergie de chaque pattern et informent l'utilisateur. Une comparaison des patterns d'utilisation en matière de consommation d'énergie est ensuite réalisée. Les informations obtenues en sortie de ce mécanisme d'apprentissage sont converties en profils d'économie d'énergie.

Les profils d'économie d'énergie contiennent différents paramètres qui sont activés pour optimiser la consommation de puissance. Ces profils sont activés

en fonction de l'état du système. Si par exemple, le pattern indique que l'application en exécution est caractérisée par une forte utilisation des ressources, alors le profil d'économie d'énergie donne le choix à l'utilisateur d'interrompre l'exécution de cette application.

Avec ces différents profils, les auteurs mentionnent une amélioration de la consommation d'énergie estimée à 20% de plus que les solutions commerciales comme Juice Defender [75]. Cependant, les stratégies proposées dans ces travaux sont toutes implémentées nativement dans les versions actuelles des systèmes d'exploitation mobiles. De plus, l'approche n'est pas toujours automatique et fait intervenir l'utilisateur pour la majorité des actions. Nous avons également noté que les actions d'optimisation proposées sont souvent basées sur les informations du systèmes d'exploitation et non pas sur les patterns générés.

Un autre travail qui s'inscrit dans le cadre des approches avec apprentissage est celui de Song et al [123], dans lequel les auteurs exploitent la durée d'interaction entre l'utilisateur et son système. Les auteurs analysent les temps de réponses du système perçus par l'utilisateur pour réduire la consommation d'énergie.

Les auteurs ont divisé les interactions entre l'utilisateur et son système en deux parties:

- Lorsque le temps de réponse du système impacte directement la satisfaction de l'utilisateur
- Lorsque un retard dans le temps de réponse du système peut être toléré.

Pour réaliser cela, les auteurs ont développé un mécanisme qui permet d'analyser le temps de réponse perçu par l'utilisateur pour réaliser une classification des applications.

Ensuite, la consommation d'énergie de chaque type de réponse est estimée selon la classe des applications. En fonction de cette estimation, les auteurs proposent de réduire la fréquence du CPU à son plus bas niveau lorsque cela n'impacte pas la satisfaction de l'utilisateur.

Les auteurs affirment qu'ils sont parvenus à minimiser la consommation d'énergie du CPU de 65% par rapport à la politique de gestion de puissance de l'OS. Cependant la classification des applications est réalisée de manière manuelle et sans algorithmes d'apprentissage, ce qui rend cette approche sujette à des erreurs de classification, impactant ainsi la satisfaction de l'utilisateur.

2.5.2 Interaction avec la batterie et les cycles de recharge

Ravi et al ont proposé un système contextuel pour la gestion de la batterie qui avertit l'utilisateur lorsqu'il détecte une limitation de puissance avant la prochaine occasion de rechargement [110]. Ils utilisent l'ensemble des applications en cours d'exécution, le débit de déchargement et les journaux d'appels téléphoniques comme entrées de leurs algorithmes de prédiction. Leur motivation est de garantir que les applications cruciales telles que le téléphone et la messagerie ne doivent pas être compromises par des applications non critiques. L'évaluation de leur algorithme a été simulée avec les traces du projet *Reality Mining* [40]. Leurs résultats indiquent que leur algorithme peut prédire la consommation de batterie et les possibilités de charge pour les utilisateurs sans entropie d'utilisation élevée. Les auteurs soulignent la difficulté de pré-

voir les appels téléphoniques en raison de l'impact des facteurs sociaux et de la grande variabilité des schémas d'appels entre les weekends et les jours de semaine.

De manière similaire, Oliver [94] utilise des méthodes de classification pour identifier trois types distincts de patterns de rechargement de la batterie chez 17 300 utilisateurs de Blackberry. Ces clusters sont définis comme des rechargeurs opportunistes, des rechargeurs de journée et des rechargeurs de nuit. Les *logs* ont été créés à l'aide d'une application en background peu couteuse en énergie. Cette application événementielle surveille l'activité de recharge, le niveau de batterie et les arrêts du système.

Dans un rapport technique qui a suivi cette étude [95], les auteurs ont utilisé des algorithmes de clustering et de classification pour prédire le niveau d'énergie restant dans les plateformes mobiles. Ils ont mis en place un *toolkit* appelé *EET*, conçu pour prédire le taux d'exécution des applications ènergivores. Cet outil permet aux développeurs d'évaluer les besoins en consommation d'énergie de leurs applications par rapport aux traces énergétiques réelles des utilisateurs. L'EET tente de déterminer le sous-ensemble des caractéristiques énergétiques de haut niveau qui différencient le mieux les utilisateurs. Leurs résultats montrent qu'il est possible de prédire le niveau d'énergie sur un système mobile avec un taux d'erreur de 7% pour l'heure suivante et un taux d'erreur de 28% pour les 24 heures suivantes.

Rahmati et al [106] ont analysé la façon dont les utilisateurs mobiles interagissent avec leurs systèmes d'un point de vue de l'interaction homme-machine. Leurs résultats sont obtenus à partir d'enquêtes directes auprès d'utilisateurs mobiles et d'un processus de suivi des données de dix utilisateurs mobiles. Ils ont trouvé des preuves qualitatives et quantitatives sur les problèmes causés par l'énergie consommée par les interfaces d'utilisateur. Ces interfaces ne tirent pas profit des paramètres d'économie d'énergie. Les auteurs ont également noté une sous-utilisation de l'énergie des batteries, causant ainsi une insatisfaction chez les utilisateurs.

Une procédure similaire a été suivie par Banerjee et al [8]. Ils ont interviewé dix utilisateurs mobiles pendant 42 jours. Dans ce cas-ci, les auteurs affirment que, malgré la grande hétérogénéité des utilisateurs, la plupart des cycles de rechargement des batteries se produisent lorsque la batterie a encore beaucoup d'énergie. Ils notent également qu'une partie considérable des recharges est pilotée par le contexte spatio-temporel. Cette information a été utilisée pour mettre en œuvre *Llama*, une gestion adaptative des ressources.

Pour conclure cette partie, il est nécessaire de mentionner les travaux antérieurs sur la prédiction du niveau des capacités restantes des batteries. Dans [137], les auteurs ont introduit une technique basée sur l'historique de la courbe de tension pour une prédiction *online* de la durée de vie batterie.

D'autres travaux [98] [109][108] décrivent quatre différents modèles stochastiques pour l'estimation de la durée de vie des batteries dans les systèmes embarqués. Le tableau 2.5 résume les études qui se sont focalisés sur les cycles de déchargement et rechargement des batteries.

Tableau 2.5 – Travaux sur l'interaction de l'utilisateur avec les batteries

Art	Sys	Description
[94, 95]	Blackberry	-Analyse des cycles de chargement de la batterie -Prédiction du niveau d'énergie par l'utilisation du clustering
[110]	Linux et Symbian	Prédiction des cycles de chargement et déchargement des batteries
[8]	Windows Mobile 5	- Monitoring de la batterie - système gestion de puissance "Llama"
[106]	Windows Mobile 5	- Survey sur les utilisateurs des systèmes mobiles - Monitoring de la batterie pour la compréhension des patterns de chargement

Dans cette thèse, nous proposons également une approche basée sur la prédiction du niveau de batterie restant. Dans la solution que nous proposons, nous donnons la possibilité à l'utilisateur d'envoyer une requête pour garder la disponibilité de son système jusqu'à une heure précise. Pour satisfaire la requête de l'utilisateur, nous prenons en compte son historique et son contexte. Contrairement aux les travaux cités dans cette section, nous n'analysons pas les cycles de rechargement. Nous partons du principe que l'utilisateur est dans l'incapacité de recharger sa plateforme jusqu'à l'heure mentionnée dans sa requête. Pour prédire si le niveau restant de la batterie permettra de satisfaire la requête de l'utilisateur, nous estimons les futures séquences d'applications et leurs temps d'exécution.

Dans la dernière partie qui suit, nous présentons comment les activités de l'utilisateur peuvent être exploitées pour mesurer et modéliser la consommation énergétique des systèmes mobiles.

2.5.3 L'exploitation des activités de l'utilisateur pour la modélisation d'énergie

Comme mentionné précédemment, les activités de l'utilisateur sont considérées comme un facteur important dans les systèmes mobiles [119, 120]. Ces activités peuvent également être utilisées pour la modélisation de la consommation d'énergie.

L'analyse de la consommation d'énergie dans un système mobile exige évidemment une connaissance approfondie de l'architecture de ce dernier. La modélisation de la consommation d'énergie d'un système mobile se fait en mettant au point un ensemble d'équations permettant d'estimer l'énergie consommée d'un système pendant une durée de temps donnée. Afin de définir ces équations, certains travaux de recherche se sont focalisés sur les activités de l'utilisateur comme dans [77, 136]. L'ensemble de ces travaux utilise les techniques de régression linéaire [71]. La figure 2.1 montre les étapes qui mènent à la définition d'un modèle de consommation énergétique.

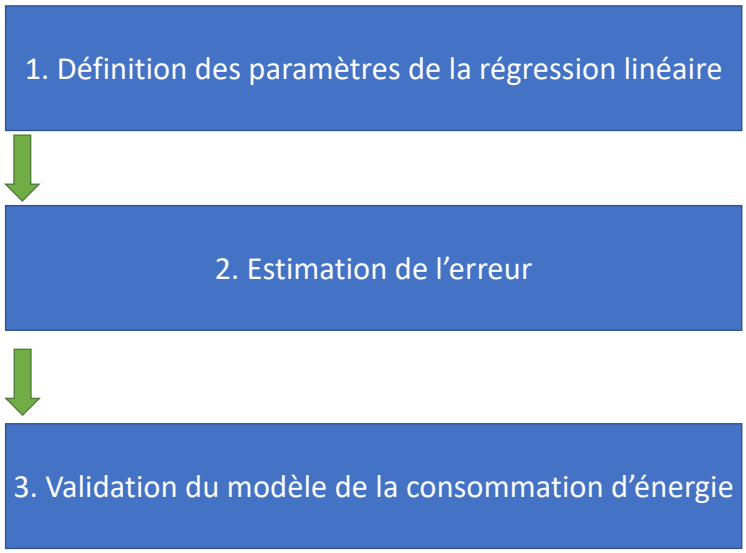

FIGURE 2.1 – Processus de définition du modèle de la consommation d'énergie

1. Définition des paramètres de la régression linéaire: après avoir analysé les activités collectées de l'utilisateur, la proposition du modèle de consommation est élaborée. Elle consiste en la définition des paramètres de régression choisis pour modéliser la consommation d'énergie. Généralement, pour chaque composant hardware, un ensemble de paramètres est fixé pour l'estimation de sa consommation. Dans le tableau 2.6, deux paramètres pour la consommation du CPU sont illustrés.

Tableau 2.6 – Exemple de paramètres fixés pour le CPU [77]

Paramètre	Description	Plage de valeurs	Consommation de puissance par unité
CPU_on	Lorsque le CPU est actif	0 ou 1	Bcpu = 152 mW
CPU_util	Pourcentage de l'utilisation CPU	0% - 100%	Butil = 7.1 mW

2. Estimation de l'erreur: après avoir fixé les paramètres de régression, une étape d'estimation d'erreur est réalisée pour calculer l'erreur introduite par le modèle [3, 119, 120, 136]. C'est en fonction de cette estimation que le modèle proposé est accepté afin de pouvoir procéder à l'étape de validation suivante.
3. La validation du modèle: afin de valider le modèle proposé, une étude de la consommation d'énergie est réalisée sur un système mobile spécifique [3, 119, 120, 136]. Un dispositif de mesure d'énergie est utilisé pour une comparaison avec les résultats du modèle. Cette étape permet de localiser l'erreur du modèle et d'assurer sa fiabilité.

Nous présentons dans ce qui suit les principaux travaux de cette catégorie. Un des travaux précurseur est celui d'Alex Shye et Al [120] qui a utilisé des activités collectées pendant six mois de la part de 25 utilisateurs.

Ce travail vise à utiliser les charges de travail des composants matériels dans la modélisation d'énergie. Le modèle proposé a été validé avec une précision qui avoisine les 90%. Le travail stipule qu'il est possible de comprendre la consommation d'énergie d'un utilisateur avec un taux d'erreur de 10%. Cependant il existe quelques limites, notamment dans la façon de construire le modèle qui se fait de manière manuelle et sa dépendance de l'architecture du smartphone HTC Dream. Cela rend l'extension du modèle aux autres systèmes mobiles très difficile, dans le sens où les paramètres utilisés dans l'étude sont variables d'un système à un autre.

D'autres chercheurs ont essayé de proposer un modèle qui ne serait pas dépendant de l'architecture du système mobile ciblé. L'un de ces travaux est de J. Lee et Al [76]. Les auteurs ont également utilisé la régression linéaire pour cette modélisation dans un processus totalement automatisé et sans aucune intervention manuelle. L'idée repose sur le fait de générer des petites activités d'utilisation appelées *chunks* dans des intervalles de temps bien précis. Ces *chunks* étaient représentés par une consommation de 1% de la capacité de la batterie. Le modèle obtenu dans ce travail a été validé avec une précision qui dépassait les 90%. Le modèle construit est également extensible à d'autres architectures. Cependant, une mauvaise précision du modèle était constatée lorsque les activités de l'utilisateur faisaient appel au GPU via les applications de jeux par exemple. La raison de cette mauvaise précision est due au fait que le composant GPU n'ait pas été étudié pour l'élaboration de ce modèle.

2.6 Conclusion

Dans ce chapitre, nous avons présenté un panorama des travaux académiques qui se sont focalisés sur l'optimisation de l'énergie dans les systèmes mobiles. Les articles dont nous avons préconisé l'étude s'intéressent aux différents niveaux de la couche software. Dans la section 2.1, nous avons présenté en premier lieu les systèmes d'exploitation soucieux de l'énergie et les différentes solutions qu'ils offrent. Nous avons deux écoles qui tentent de traiter cette problématique. La première école défend l'idée que c'est aux applications de s'adapter dynamiquement aux limitations énergétiques et la seconde préconise la centralisation et la surveillance de ces limitations au niveau de l'OS. Dans nos contributions, nous tentons de trouver un compromis entre ces deux approches qui peuvent être complémentaires. Nous avons également discuté des profileurs d'énergie et les moniteurs de ressources qui tentent de cartographier la consommation d'énergie avec les différentes applications, pour proposer des techniques d'économie d'énergie.

Dans la section 2.2, nous avons présenté les travaux de recherche qui s'intéressent à l'optimisation d'énergie des interfaces de communication. Il existe dans cette catégorie différentes solutions qui sont principalement: l'utilisation de la Wi-Fi pour les téléchargements consistants, l'utilisation des signaux à forte intensité pour la communication cellulaire et les techniques de planification pour les applications tolérantes aux délais. Une autre solution qui permet une

économie d'énergie est la compression des données transférées.

Dans la section 2.3, nous avons discuté de l'une des sous-problématiques de la consommation d'énergie la plus courante, qui est l'optimisation de la géolocalisation. Nous retrouvons dans la littérature plusieurs approches. Les plus populaires sont la combinaison de plusieurs capteurs à faible consommation de puissance, pour remplacer des capteurs énergivores comme le GPS. L'utilisation des modèles probabilistes pour prédire les futures emplacements de l'utilisateur. Nous avons également discuté les travaux qui proposent des heuristiques pour l'adaptation du taux d'échantillonnage des capteurs.

Dans la section 2.4, nous avons traité les approches basées sur le *cloud computing*. Ces solutions d'*offloading* représentent de nombreux atouts pour la réduction de la consommation d'énergie. La solution la plus préconisée est la délocalisation des calculs intensifs et des fonctions énergivores. Néanmoins, il faut être prudents avec le QoS des applications et par conséquent, la satisfaction de l'utilisateur.

La section 2.5 discute les solutions qui analysent les besoins de l'utilisateur. Nous y retrouvons des travaux dédiés à la collecte des traces de l'utilisateur pour en dégager des patterns. Ces patterns serviront par la suite à proposer des optimisations d'énergie. La majorité des travaux présentés utilisent des techniques d'optimisation sans apprentissage automatique. Les auteurs se contentent de mettre en place des heuristiques qui se basent sur les patterns observés sans aucun aspect adaptatif. Ce manque d'approches automatisées nous a fortement motivé à proposer des solutions basées sur l'apprentissage des habitudes de l'utilisateur, tout en considérant sa satisfaction et en s'adaptant aux différents changements de son contexte.

La dernière partie 2.5.3 présente l'utilisation des activités de l'utilisateur dans la modélisation de la consommation d'énergie. Ces solutions utilisent généralement une régression linéaire appliquée aux paramètres jugés pertinents pour la consommation de puissance.

Le chapitre suivant présente l'outillage commun à l'ensemble de nos contributions, ainsi que notre plateforme de test. Cela concerne en particulier l'*Intel Energy Chcecker Software Development Kit (IECSDK)*.

Chapitre 3

Intel Energy Checker SDK et Environnement d'Expérimentations

Cette thèse s'inscrit dans le cadre d'une étroite collaboration avec l'entreprise *Intel Corp* et plus précisément avec l'équipe du Dr Jamel Tayeb qui travaille sur l'acquisition des données et l'optimisation de la consommation d'énergie. Au cœur de cette collaboration, nous retrouvons un logiciel phare d'*Intel* qui est l'*Intel Energy Checker Software Development Kit* (IECSDK).

L'idée de départ de l'IECSDK était de concevoir un logiciel capable de mesurer l'efficience énergétique d'une application sans devoir utiliser un matériel coûteux et fastidieux à manipuler comme un wattmètre. Au fil du temps, l'équipe responsable de ce projet a réorienté ses travaux dans le domaine du *Big Data* pour la collecte et l'analyse de données. Les informations traitées étaient utilisées pour l'optimisation de la consommation d'énergie et la conception des nouvelles technologies d'*Intel*, comme les dernières générations de processeurs *Intel*.

Notre ambition à travers cette coopération était d'étendre l'utilisation de l'IECSDK pour collecter diverses informations qui serviront à la modélisation et l'optimisation de la consommation d'énergie. Dans le cadre de cette thèse, nous avons utilisé l'IECSDK principalement pour:

- Mesurer et journaliser la consommation de puissance et d'énergie lors de nos différentes expérimentations.
- Implémenter nos contributions et les adapter selon l'architecture de l'IECSDK et son modèle de fonctionnement.

L'objectif industriel de cette collaboration était de mettre en place des modules logiciels additionnels permettant de réduire la consommation d'énergie du système mobile. L'idée était de proposer une meilleure gestion que celle implémentée par l'OS. Les modules développés seront intégrés à l'IECSDK et seront déployés comme des sur-couches logicielles dans différentes plateformes mobiles.

Dans ce chapitre, nous présentons les composants de l'IECSDK et son utilisation. Nous développons également comment l'architecture de nos contributions s'appuie sur l'IECSDK. Pour finir, nous donnons également un aperçu de la plateforme mobile utilisée dans nos expérimentations et le matériel mis à notre disposition pour les différentes mesures de la puissance consommée. Ce chapitre est organisé comme suit:

- La section 3.1 présente un bref historique de l'IECSDK et quelques outils existants de *green computing* qui permettent de mesurer la consommation d'énergie.

- Dans la section 3.2, nous présentons l'architecture de l'IECSDK et ses composants utilisés dans le cadre de nos contributions.
- La section 3.3 présente notre plateforme de test ainsi que le wattmètre utilisé.
- La section 3.4 conclut ce chapitre.

Sommaire

3.1 Intel Energy Checker Software Development Kit . . **46**
3.2 Les composants principaux de l'IECSDK **47**
3.2.1 L'Energy Server (ESRV) 48
3.2.2 L'Intel Modeler 49
3.3 Plateforme de test et de mesure de puissance **53**
3.4 Conclusion **54**

3.1 Intel Energy Checker Software Development Kit

Comme mentionné précédemment, l'IECSDK a été conçu en premier lieu pour faciliter l'instrumentalisation du code source d'un logiciel, dans le but d'évaluer son efficience énergétique. Cette évaluation servira à l'optimiser dans sa phase de développement.

Kevin Bross, qui a travaillé sur le projet à son lancement illustre l'approche par l'exemple suivant:*"Un automobiliste qui souhaite se rendre derrière une colline consommera plus d'énergie s'il décide de passer par le sommet que s'il la contourne. Dans le même esprit, un logiciel peut être plus "efficient" s'il met en œuvre un algorithme qui "contourne la colline" plutôt qu'une approche brutale qui est peut être plus simple et plus rapide à mettre en œuvre mais consomme plus d'énergie"*.

Dans cette logique, l'IECSDK permet de mettre en place un compteur d'unités pour ce qui est appelé le "travail utile". Par exemple pour évaluer un serveur de courriels, l'indicateur du travail utile peut être un envoi d'un mail, pour un logiciel de traitement d'image, cela pourrait être le calcul du rendu d'un pixel, ou encore, pour un serveur de base de données, l'indicateur peut être une requête définie.

Étant donné que l'IECSDK est compatible avec un certain nombre de wattmètres, nous pouvons facilement obtenir un indicateur d'efficience global de la solution matérielle et logicielle. Par exemple: WH consommés par mail envoyé, WH consommés par pixel traité, etc.

Au delà de l'optimisation à priori, l'IECSDK permet également de comparer l'efficience énergétique, à posteriori, de deux solutions logicielles équivalentes. Les différences en consommation de ressources (CPU, mémoire, espace de stockage, etc.) entre deux logiciels qui fournissent le même travail peuvent être considérables. *Facebook* a par exemple divisé par 2 la consommation électrique de ses serveurs en compilant le code PHP de son site.

Toutes ces raisons font que l'idée de départ de l'IECSDK est très novatrice car la couche logicielle a un rôle prépondérant dans la performance globale d'une application. En entreprise, c'est également le facteur primordial qui pousse au renouvellement des postes de travail. Or, bien que l'optimisation de l'efficience

énergétique du matériel soit en perpétuel avancement, peu de solutions proposent l'optimisation de cette efficience au niveau du code source. Nous pouvons néanmoins en citer quelques unes comme:

- *pTop* [1], développé par l'université de Wayne (US), bibliothèque de monitoring énergétique. Cette bibliothèque est assez proche de nos besoins, dans le sens où le calcul énergétique se base au niveau des différents composants matériels tels que le CPU, mémoire, disque, etc. Cependant, cet outil n'offre aucun support de reconfiguration. Tout se faisant au niveau du code et donc avec une nécessité de recompiler l'outil après toute nouvelle configuration. La solution ne propose non plus aucun support d'interopérabilité.
- *JouleMeter* [55] est un outil de monitoring énergétique des composants matériels développé par *Microsoft*. Il permet d'obtenir la consommation énergétique d'un composant matériel et également celle d'une application donnée. L'inconvénient de cet outil est qu'il est exclusivement développé pour les plateformes Windows.
- Nous pouvons également citer *PowerTop* [2] et *PowerAPI* [135]. Le premier a été développé par *Intel* et fournit pour l'utilisateur des astuces de réduction d'énergie. *PowerAPI* quant à lui, a été développé par l'INRIA et offre une API qui permet de mesurer la consommation énergétique d'un processus système. Le calcul se base sur la consommation énergétique des composants matériels utilisés par le processus. Cela est réalisé en utilisant des formules énergétiques sans avoir recours à un wattmètre.

Dans le cadre de cette thèse, nous n'avons pas utilisé l'IECSDK pour instrumentaliser le code source des logiciels dans le but de mesurer leur efficience énergétique. Nous exploitons l'IECSDK pour implémenter notre modèle CPA (Collect - Process - Adjust) présenté dans l'introduction, ainsi que pour mesurer les gains de puissance et d'énergie apportés par nos solutions.

3.2 Les composants principaux de l'IECSDK

L'*Intel Energy Checker SDK* englobe deux composants principaux qui sont: l'ESRV et le *Modeler*.

1. L'ESRV est le principal moteur du SDK à travers lequel toutes les interactions utilisateurs sont canalisées. Il sert principalement à mesurer la consommation d'énergie de la plateforme.
2. Le *Modeler* est un module de support de périphérique dédié à l'ESRV, il est également considéré comme le framework de l'IECSDK. Il fournit des services de base comme la journalisation de la puissance et offre de multiples mécanismes d'expansion. Ces mécanismes sont exposés aux développeurs à travers des interfaces dédiées. Le *Modeler* est composé principalement par:
   - Le *Front-End* sert à implémenter la collecte de données ou d'inputs via les Input Libraries (ILs).
   - L'*Input-Bus* est une mémoire partagée entre le *Front-End* et le *Back-End*.

1. pTop, http://www.sigops.org/sosp/sosp09/papers/hotpower.do.pdf
2. https://01.org/powertop

- Le *Back-End* permet de consommer les données collectées via les Actuator Libraries (ALs).

Dans le cadre de cette thèse, nous nous sommes focalisés sur deux modules d'expansion qui sont les ILs et les ALs. Ces modules sont développés dans la section 3.2.2. Les différents composants mentionnés sont retranscris par la figure 3.1 ci-dessous.

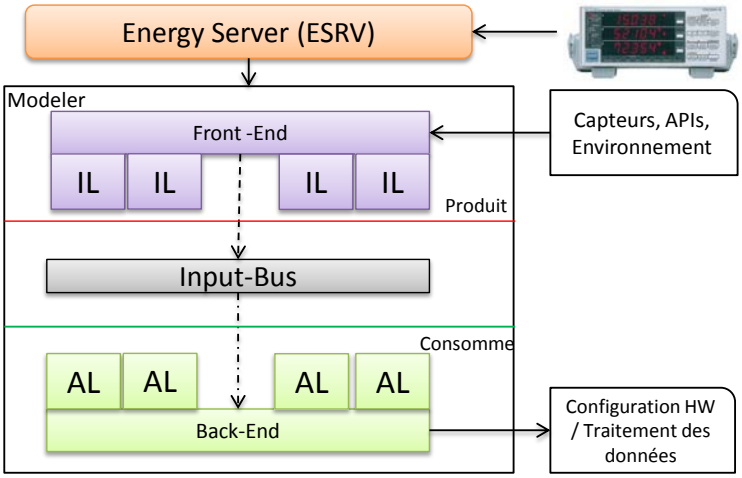

FIGURE 3.1 – Architecture simplifiée de l'IECSDK

Nous retrouvons dans qui suit ses deux composants principaux; l'ESRV et le *Modeler*.

3.2.1 L'Energy Server (ESRV)

L'ESRV réalise principalement le monitoring de la consommation de puissance d'une plateforme en l'instrumentalisant. Ce composant fournit des compteurs pour l'énergie cumulée consommée, ainsi que la consommation de puissance instantanée, directement lue de l'appareil de mesure.

Au départ, l'ESRV servait à mesurer l'efficience énergétique des logiciels qui est définie comme le rapport entre la quantité de travail utile effectuée et l'énergie consommée par le système hôte. L'ESRV jauge alors cette quantité de travail utile via des métriques prédéfinies au préalable. Par exemple, le nombre de mails envoyé peut être considéré comme une métrique pour mesurer le travail utile d'un serveur mail.

L'ESRV permet de récupérer la puissance électrique consommée par le système et calcule également l'énergie pendant une durée de temps donnée. Pour réaliser cela de manière transparente pour les utilisateurs, l'ESRV s'interface avec des wattmètres et journalise toutes les valeurs de puissance mesurées par ces appareils. L'ESRV supporte un certain nombre de wattmètres, cependant, cette liste peut être étendue à d'autres appareils. La figure 3.2 présente un schéma simplifié du mode de fonctionnement de l'ESRV.

L'ESRV est considéré comme le driver de l'IECSDK. Toutes les interactions de l'utilisateur ou des logiciels avec l'IECSDK sont canalisées à travers ce composant. L'ESRV fournit de multiples services comme:

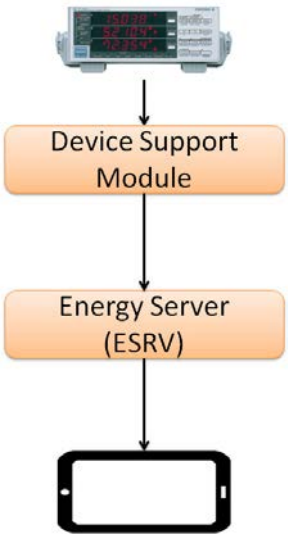

FIGURE 3.2 – Mesure de la consommation de puissance d'une plateforme via l'ESRV

- Instrumentalisation des applications;
- Communications avec le wattmètre;
- Surveillance du fonctionnement du *Modeler* et ses composants;

Étant donné que nous ne mesurons pas l'efficience énergétique des applications, nous utilisons l'ESRV dans nos contribution pour:

1. Le lancement de l'IECSDK car c'est son principal driver;
2. La retranscription de la puissance consommée et le calcul de l'énergie car c'est le seul moyen de s'interfacer avec l'appareil de mesure;
3. L'invocation du *Modeler* qui passe impérativement par l'ESRV;

Tous ces points rendent l'ESRV indispensable et principalement le troisième point, étant donné que nos implémentations reposent sur l'expansion de l'architecture du *Modeler*.

3.2.2 L'Intel Modeler

Dans la première version de l'IECSDK, le *Modeler* servait principalement à fournir et exécuter des modèles de puissance spécifiques à chaque plateforme. Un modèle de puissance est un ensemble d'équations qui décrivent la consommation de puissance d'un système donné. Avec ces modèles, le *Modeler* offrait la possibilité d'estimer la consommation de puissance sans avoir recours à un appareil de mesure externe comme le wattmètre. Cette fonctionnalité facilite l'utilisation de l'IECSDK. L'utilisateur n'est plus obligé d'acheter un matériel supplémentaire (le wattmètre). Par ailleurs, l'utilisation du *Modeler* réduit le temps de la mise en place de la plateforme. L'utilisateur n'est plus dans l'obligation de connaitre le mode de fonctionnement du wattmètre. Ces modèles de puissance étaient obtenues via l'implémentation d'une Input Library (IL) comme le montre la figure 3.3.

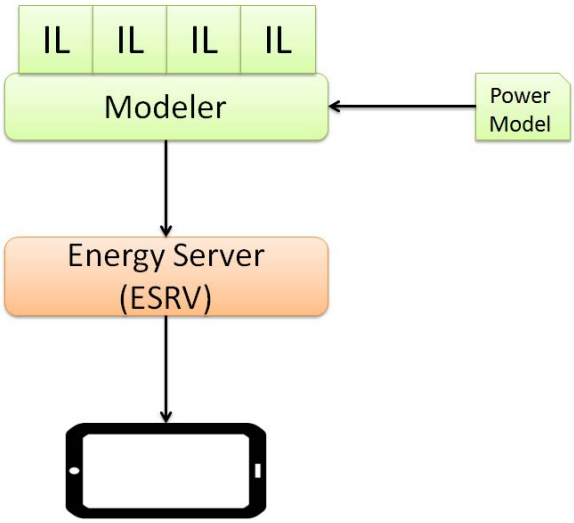

FIGURE 3.3 – L'utilisation du *Modeler* par l'ESRV

Nous détaillons le concept d'IL dans la section 3.2.2.2. Nous rappelons que le *Modeler* est composé principalement du *Front-End*, l'*Input Bus* et le *Back-End*, comme présenté dans la figure 3.1.

3.2.2.1 Front-End

Le *Front-End* sert à implémenter la collecte de données ou d'inputs. En raison de ce rôle, le *Modeler* nous sert dans notre thèse de collecteur. Le *Front-End* est modulaire et extensible via les ILs. Peu importe l'information que nous souhaitons collecter, il suffit d'implémenter une IL correspondante et l'inclure au *Modeler* via le *Front-End*.

3.2.2.2 Input Libraries (ILs)

Le concept d'IL a été mis au point pour la collecte des inputs sur les performances du système. Les ILs sont également utilisées pour la corrélation de ces inputs avec la puissance mesurée ou le modèle de puissance correspondant. Les inputs sont utilisés lors de l'exécution d'un programme donné afin de mesurer son efficience énergétique. Ainsi durant la conception d'un modèle de puissance, différents inputs sont explorés et testés.

De manière générale, les ILs représentent les unités de collecte de données du *Modeler* dans le but d'étendre le *Front-End*. La nature d'une IL est de permettre une forte réutilisation du code, réduisant ainsi le Time To Market (TTM). L'utilisation des ILs existantes permet de tirer profit de l'expertise d'autres développeurs. Les ILs peuvent être exploitées de sources multiples et intégrées par la suite. Chaque développeur peut implémenter une IL spécifique à ses besoins. Ainsi les efforts de développement peuvent être mutualisés entre différents concepteurs comme le schématise la figure 3.4.

Les ILs exposent au moins une entrée. Cependant, elles peuvent se reconfigurer à tout moment pour modifier le nombre et la nature de leurs entrées.

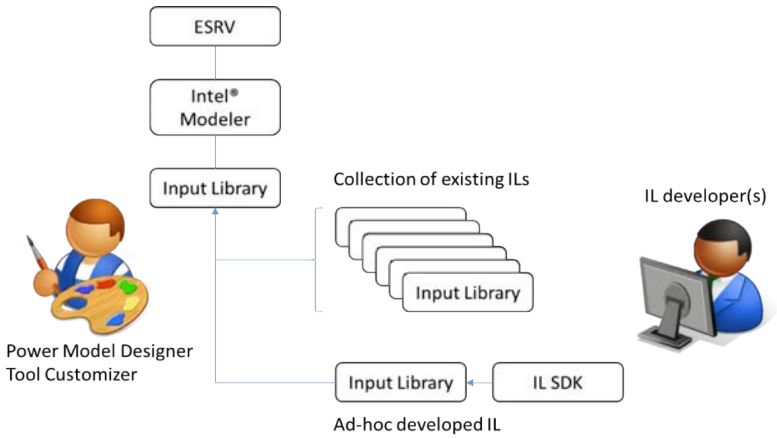

FIGURE 3.4 – L'utilisation de différentes ILs

Cette reconfiguration ne nécessite aucun arrêt ou redémarrage du *Modeler*. Cela permet le développement de modules complexe avec la possibilité de s'adapter aux différents changements qui occurrent dans la plateforme.

Dans le cadre de cette thèse, l'utilisation des ILs est primordiale. Cette utilisation a été étendue à une collecte de données plus généralistes comme nous allons le développer dans nos contributions. Après l'implémentation des ILs, le *Front-End* produit les différentes inputs et les expose dans l'*Input Bus*.

3.2.2.3 Input-Bus

Une fois dans l'*Input-Bus*, les données des ILs sont disponibles pour être consommées par les composants du *Back-End* après une phase de renommage, comme le montre la figure ci-dessous.

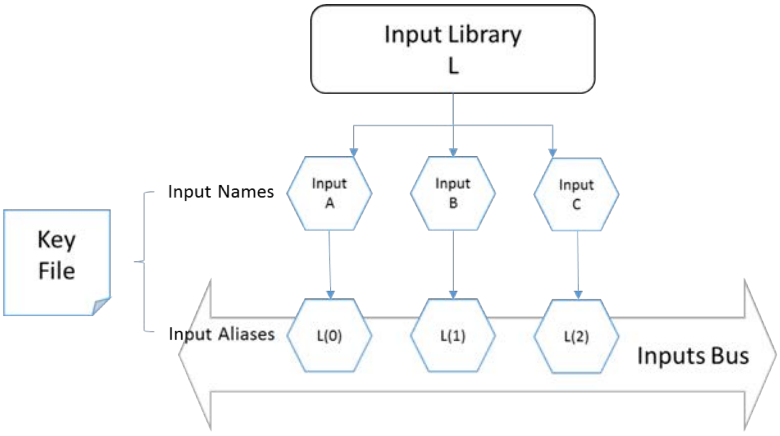

FIGURE 3.5 – Renommage des inputs dans l'*Input-Bus*

L'*Input Bus* peut être considéré comme un *buffer* entre le *Front-End* et le *Back-End*. Les inputs disponibles dans l'*Input Bus* sont automatiquement renommés

via des alias. Ces alias servent à faciliter le référencement de ces inputs dans les composants du *Back-End*. L'alias d'un *input* est dérivé de son rang et du nom de l'IL. Par exemple, un input appelé *Instructions Per Cycle* d'une IL nommée *CPU* peut être renommé *CPU (7)*. L'association entre le nom des inputs et l'alias est obtenue via un service de *logging* disponible dans le *Back-End*.

3.2.2.4 Back-End

Le *Back-End* fournit un ensemble de services tels que le *logging*, une corrélation entre la puissance et les inputs et l'évaluation des équations. De manière similaire au *Front-End*, le *Back-End* est modulaire et extensible via les ALs.

3.2.2.5 Actuator Libraries ALs

Les ALs sont les homologues des ILs et représentent les unités de traitement et les actionneurs du *Modeler*. Les ALs peuvent être utilisées pour créer des actions personnalisées qui doivent être lancées pendant l'exécution. Avec les ALs, nous pouvons implémenter le chemin le plus court et le plus rapide possible entre la collecte, l'analyse et le traitement des données. Les ALs ont des avantages similaires aux ILs en matière de réutilisation, de partage et de TTM.

Dans le cadre de nos travaux de thèse, les ALs sont utilisées pour implémenter les différentes actions de traitement de données comme les techniques d'apprentissage supervisé et la classification. Nous nous en servons également pour mettre en place nos actions de gestion du hardware.

La communication entre les ILs et ALs est schématisée dans la figure 3.6.

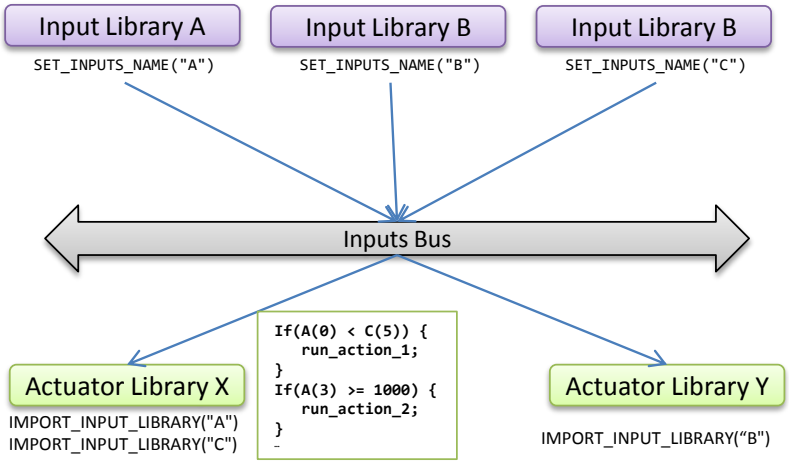

FIGURE 3.6 – Exemple de communication entre ILs et ALs

Cette figure nous montre trois ILs A, B et C qui exposent leurs inputs dans l'*Input-Bus*. Nous avons également deux ALs X et Y. L'Actuator Library (AL) X importe les inputs fournis par l'IL A et C pour réaliser un traitement dessus, tandis que l'AL Y importe les inputs de l'IL B.

La section suivante présente notre plateforme de test.

3.3 Plateforme de test et de mesure de puissance

Le déploiement des solutions, les tests et les mesures de la consommation de puissance ont été réalisés sur un *Ultrabook Intel*. Ce système est un 2en1 entre la tablette et le *laptop*. L'*Ultrabook* s'exécute sous Windows 8.1 avec un processeur Intel dual-core i7-u3667U cadencé à 2.50 GHz et 4 GB de RAM. Le tableau 3.1 présente les caractéristiques hardwares de l'*Ultrabook*.

Tableau 3.1 – Intel 2in1 Ultrabook Features

Plateforme analysée	2in1 Intel Ultrabook
Average Battery Life (Hours)	8 Hours
Maximal Power Consumption (W)	23 W
Minimal Powe 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 Go

Les solutions proposées dans cette thèse sont génériques et peuvent s'exécuter sur différents types de plateformes telles que les smartphones, les tablettes et les *Ultrabooks*. La raison principale pour laquelle l'*Ultrabook Intel* a été choisi réside dans la simplicité à le connecter avec notre appreil de mesure, le Yokogawa WT210 [61] comme illustré dans la figure 3.7.

Figure 3.7 – Installation de l'Ultrabook avec le Yokogawa WT210

L'*Ultrabook* contient également un port pour carte SIM et un écran tactile. En addition aux caractéristiques hardwares, avec *Windows Store*, nous avons accès à un large choix d'applications *Metro* [5] comme *Facebook*, *Instagram*, *Shazam*, etc. Ces applications sont largement utilisées dans les smartphones et seront exploitées avec l'*Ultrabook* comme des applications à part entière sans les utiliser via un navigateur Web. Toutes ces caractéristiques rendent légitime l'utilisation de

l'*Ultrabook* qui est un système mobile à part entière. Nous justifions également ce choix par le fait que la version actuelle de l'IECSDK pour les smartphones est encore sous développement.

À titre indicatif, la figure 3.8 présente la consommation d'énergie moyenne des composants principaux de notre plateforme mobile de test lorsque le système est actif. Pendant l'utilisation de la plateforme, nous avons mesuré la consommation de puissance 10 fois et chaque mesure dure 10 secondes. La moyenne de la consommation de chaque composant est ensuite calculée et rapportée par la figure ci-dessous. Cette consommation d'énergie peut différer suivant les applications et l'interaction de l'utilisateur avec sa plateforme, mais ces différences sont minimes, en comparaison avec les résultats de la figure 3.8.

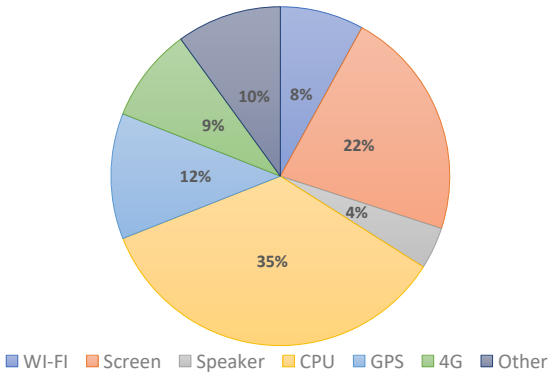

FIGURE 3.8 – Consommation d'énergie moyenne des composants hardware de l'Ultrabook

Nous notons que le processeur et l'écran représentent les composants les plus énergivores. La partie *other* dans le graphique représente la consommation d'énergie des capteurs embarqués tels que l'accéléromètre, le gyroscope, le bluetooth, etc. Les résultats ci-dessus nous aident à cibler les composants matériels qui offriront un gain énergétique important. La répartition de la consommation d'énergie entre les différents composants hardwares de l'*Ultrabook* est assez similaire dans les autres types de systèmes mobiles. En effet, dans les smartphones et les tablettes, les composants hardwares qui consomment le plus d'énergie sont également l'écran et le processeur.

3.4 Conclusion

Dans ce chapitre, nous avons présenté l'Intel Energy Checker SDK (IECSDK) et ses composants principaux qui sont : l'ESRV et le *Modeler*.

Dans cette thèse, l'ESRV nous permet de mesurer la puissance et l'énergie consommées par notre système mobile en s'interfaçant avec le wattmètre. L'ESRV nous est également indispensable pour lancer le *Modeler* et étendre son utilisation.

Le *Modeler*, quant à lui, nous donne la possibilité d'implémenter des ILs pour la collecte de données et des ALs pour mettre en place nos algorithmes de

traitement et la configuration hardware. Pour ce faire, tous les modules logiciels de nos contributions sont implémentés sous formes d'ILs et ALs et sont intégrés au sein de l'IECSDK.

L'utilisation de l'IECSDK nous offre plusieurs avantages comme la modularité et la généricité de nos contributions. L'IECSDK nous permet également de réduire considérablement le TTM et nous donne la possibilité d'inclure d'autres ILs et ALs pour enrichir nos solutions.

Nous avons également présenté dans ce chapitre notre plateforme de test et nous avons donné un aperçu de notre appareil de mesure, le wattmètre. La partie suivante englobe les quatre chapitres qui détaillent nos différentes contributions.

Deuxième partie

Contributions

Chapitre 4

Sensor Based Optimization Component

Nous présentons dans ce chapitre la première contribution réalisée dans le cadre de cette thèse. Mené à terme, ce travail nous a permis de mettre en place le composant SBOC (Sensor Based Optimization Component). Ce composant est une sur-couche logicielle responsable de l'optimisation de la consommation de puissance d'une catégorie de systèmes mobiles tels que les *Ultrabooks*. Cette optimisation se base sur les données des capteurs embarqués disponibles dans la plateforme dans le but de configurer dynamiquement la luminosité de l'écran et le volume des hauts-parleurs. Le premier objectif de SBOC est de démontrer qu'il existe des opportunités de réduction de puissance améliorant la gestion proposée par l'OS. Avec SBOC, nous prouvons que ces opportunités ne sont pas exploitées par les concepteurs des systèmes d'exploitation mobiles actuels.

L'idée de base est que l'utilisateur et son système peuvent se trouver dans des situations où il n'est pas nécessaire de garder certains composants hardwares actifs. De ce fait, une réduction de la puissance consommée est possible, mais demeure inexploitée par la gestion de puissance de l'OS.

Les situations offrant des économies de puissance sont principalement liées à la position et l'environnement dans lesquels se trouve le système. Ces circonstances sont détectables via les valeurs exposées par les capteurs embarqués. Les situations traitées par SBOC sont liées à la lumière ambiante, le bruit ambiant, l'inclinaison et le mouvement du système.

Comme toutes les approches proposées dans cette thèse, SBOC obéit au modèle CPA, présenté dans le chapitre 1. En comparaison avec les autres contributions, le composant de ce chapitre présente un certain nombre d'avantages comme un faible coût en matière de consommation de ressources, un impact mémoire négligeable, une simplicité d'implémentation et un aspect dynamique et reconfigurable. Tous ces points vont être développés tout au long de ce chapitre. SBOC représente également la première solution qui exploite l'architecture du *Modeler* de l'IECSDK présentée dans le chapitre 3. En effet, la mise en place de ce composant nous a permis de prendre en main le SDK d'*Intel* et d'implémenter des APIs destinées à collecter les données des capteurs embarquées.

Ce chapitre est organisé comme suit : nous commençons par présenter le contexte et les motivations de ce travail. Ensuite, nous détaillons l'architecture de SBOC avec tous les modules et les phases qui aboutissent à son fonctionnement. Nous finissons avec l'évaluation de notre composant suivie de la conclusion.

Sommaire

4.1 Contexte et motivation **60**

4.2 Architecture modulaire de SBOC **62**
4.2.1 Sensor Collection Module SCM 64
4.2.2 Étude des données des capteurs et SPM 66
4.2.3 Ressource Manager Module (RMM) 74
4.3 Évaluation et coût de SBOC **76**
4.3.1 Scénarios, configuration et jeu de test 76
4.3.2 Coût de SBOC 82
4.4 Conclusion **84**

4.1 Contexte et motivation

Comme mentionné dans l'introduction générale, le nombre des capteurs embarqués est en augmentation exponentielle dans les systèmes mobiles. Ces capteurs sont indispensables au fonctionnement optimal de la plateforme et sont porteurs de diverses informations. Les capteurs embarqués permettent également de répondre à de nombreuses questions concernant l'utilisateur, le système et l'environnement dans lequel ces derniers se trouvent.

Ces capteurs embarqués peuvent répondre par exemple aux interrogations suivantes:

- Quelle est la position de l'utilisateur ?
- À quelle allure l'utilisateur se déplace ?
- Dans quelle position se trouve le système ?
- Quelle est la lumière ambiante de la pièce dans laquelle l'utilisateur se trouve ?
- L'utilisateur se trouve t-il dans un environnement bruyant ou clame ?
- Quel est l'emplacement de l'utilisateur ?

Cette liste n'est pas exhaustive et les capteurs embarqués permettent de répondre à davantage de questions. Ces capteurs collectent des quantités considérables de données en temps réel et en continu, suivant un taux d'échantillonnage prédéfini. Ce mode de fonctionnement les rend certes énergivores mais permet d'avoir des informations susceptibles de réduire la puissance consommée par le système. Ainsi que nous l'avons vu dans la section 2.3, de nombreux travaux de recherche ont été orienté exclusivement sur l'optimisation énergétique des capteurs embarqués. Contrairement à ces travaux, nous ne nous concentrons pas sur la consommation des capteurs, mais nous exploitons leur omniprésence pour configurer à la baisse des composants hardwares encore plus énergivores. Avec SBOC, nous tirons profit de ces données massives pour mettre au point des heuristiques permettant de réduire la consommation de puissance totale du système.

Comme introduit précédemment, toutes les approches mises en places dans cette thèse ont pour but ultime d'améliorer la consommation de puissance proposée par le système d'exploitation. La différence entre SBOC et les autres contributions réside principalement dans l'aspect dynamique des traitements. Avec SBOC, nous ne réalisons pas de stockage. Les données sont consommées à la volée pour une reconfiguration *inline* des ressources matérielles. Cet aspect en fait une solution peu coûteuse en ressources de calcul, en puissance, en temps de déploiement et avec un faible impact mémoire.

Toutes ces caractéristiques donnent la possibilité d'embarquer SBOC dans des systèmes mobiles limités d'un point de vue matériel.

La figure 4.1 présente une vision schématisé des trois phases de SBOC basée sur le modèle CPA.

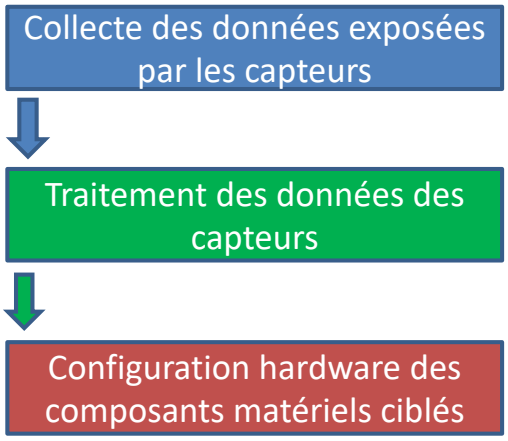

FIGURE 4.1 – Abstraction de l'architecture globale de SBOC basée sur le modèle CPA

Le fonctionnement de SBOC est concrétisé en trois phases chronologiques:

1. **Collecte des données exposées par les capteurs**: cette première étape consiste à collecter les valeurs exposées par les capteurs en temps réel. Dans un premier temps, nous collectons les données de tous les capteurs pour les filtrer et en garder uniquement les capteurs utiles à l'élaboration de notre approche. Ce filtrage se fait à travers la seconde phase.
2. **Traitement des données des capteurs**: les données capturées sont traitées pour détecter l'état et l'environnement du système. Cette détection sert à la génération d'une base de connaissance et des politiques de gestion de puissance. Cette étape est cruciale et se traduit par la désignation des capteurs les plus pertinents, ainsi que le filtrage de leurs données. Le traitement des données se fait lors d'une phase expérimentale en prenant en compte chaque capteur et les valeurs qu'il expose. Cette phase expérimentale nous sert à différencier entre les différentes situations dans lesquelles le système se trouve. Les résultats de cette phase sont utilisés lors de la phase suivante.
3. **Configuration des composants matériels ciblés** : les résultats obtenus dans la phase précédente nous permettent de mettre en place nos politiques d'optimisations. Ces politiques sont appliquées aux composants matériels ciblés en ajustant leurs configurations selon l'état du système et l'environnement. Avec SBOC, nous nous focalisons sur le niveau de la luminosité de l'écran et le volume des hauts parleurs. Néanmoins, le même processus peut être utilisé pour cibler d'autres composants tels que le GPS et la Wi-Fi.

Dans la section suivante, nous présentons l'architecture détaillée de notre composant et les modules logiciels qui l'englobe.

4.2 Architecture modulaire de SBOC

Comme mentionné dans la section précédente, SBOC agit à travers trois phases communicantes. Pour assurer ces trois phases, trois modules et une étude expérimentale sont mis en place comme le montre la figure 4.2.

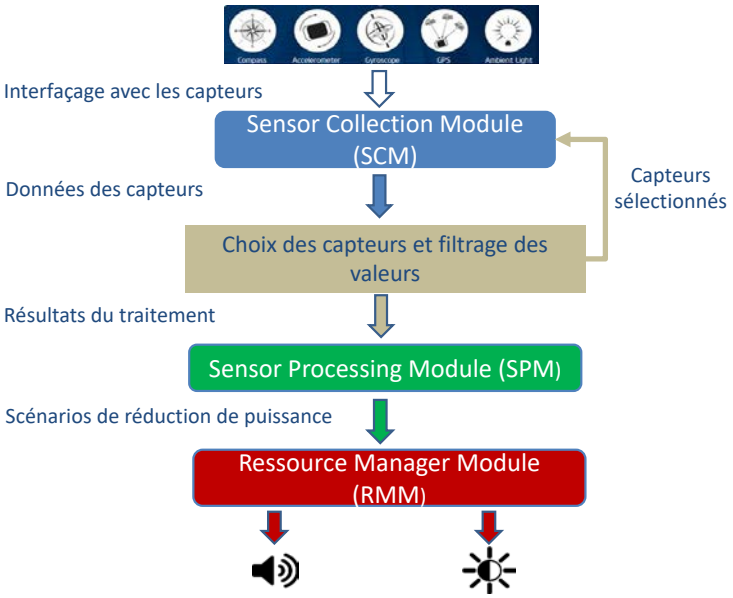

FIGURE 4.2 – Architecture modulaire de SBOC

Dans la figure ci-dessus, nous avons:

1. Le Sensor Collection Module (SCM) s'interface avec tous les capteurs disponibles dans la plateforme pour collecter toutes les valeurs exposées. Ces valeurs sont par la suite filtrées et traitées à travers une phase expérimentale. À l'issue de cette phase, le SCM est notifié pour s'interfacer uniquement avec les capteurs sélectionnés. Le SCM a été implémenté sous forme d'une *input library* (IL). Ce module est détaillé dans la section 4.2.1.
2. Le Sensor Processing Module (SPM) assure la seconde phase de traitement. Il est considéré comme la partie intelligente de SBOC. Nous réalisons une étude préliminaire à travers ce module pour sélectionner les capteurs les plus utiles à l'élaboration de notre approche. Le SCM est ensuite notifié des capteurs sélectionnés . Parmi ces capteurs, nous filtrons également leurs métriques. Les résultats de cette étude permettent au SPM de détecter en temps réel les scénarios susceptibles d'offrir des opportunités de réduction de puissance. Les informations de ces détections sont ensuite transmises au Resource Manager Module (RMM). Cette phase d'études et le rôle du SPM sont détaillés dans la section 4.2.2.
3. Le Ressources Manager Module (RMM) réalise la configuration dynamique des ressources hardwares ciblées. Cette configuration est réalisée via les drivers des composants matériels. Ces optimisations se font en fonction des scénarios détectés par le SPM. Nous détaillons le mode de

fonctionnement du RMM dans la section 4.2.3. Pour démontrer la faisabilité de SBOC, nous nous sommes intéressés à deux ressources spécifiques qui sont la luminosité de l'écran et le volume des hauts parleurs. Ces deux ressources ont été sélectionnées pour illustrer l'aspect dynamique de SBOC qui configure la luminosité et le volume à la volée. Néanmoins, la solution peut être étendue à d'autres composants hardwares.

Nous nous sommes limités à ces deux ressources pour plusieurs raisons qui sont principalement:

- **La rapidité de configuration**: la latence provoquée par le changement de la luminosité de l'écran et le volume des hauts parleurs est négligeable et se mesure en microsecondes. Cette latence n'impacte nullement les performances de la plateforme et n'est pas discernable par l'utilisateur.
- **La disponibilité des interfaces de programmation (APIs)**: les systèmes d'exploitations exposent des APIs qui permettent la modification de l'intensité de la luminosité et le niveau du volume des hauts parleurs de façon dynamique, contrairement à d'autres ressources où l'OS n'offre aucun moyen pour les contrôler.
- **L'expérience de l'utilisateur**: en gérant la luminosité et le volume des hauts parleurs, la satisfaction de l'utilisateur est moins impactée, en comparaison avec d'autres composants plus critiques tels que le processeur ou encore le GPS. SBOC est un composant qui agit de façon dynamique en fonction des valeurs des différents capteurs. La luminosité et le volume ne sont pas considérés comme des ressources critiques et peuvent être contrôlés avec cette démarche dynamique.
- **Le gain de puissance élevé**: le gain de puissance résultant d'une bonne gestion de ces deux composants dépasse les 30%. En effet, une réduction de la luminosité à son plus faible niveau permet un gain avoisinant les 28%. De manière similaire, une réduction totale du volume génère un gain supérieur à 4 %. En pratique, ces économies d'énergie sont rarement atteignables uniquement en gérant la luminosité et le volume. Cependant, une bonne gestion de ces deux ressources permet un gain d'énergie dépassant les 20 % dans certains cas de figure.
- **La disponibilité des capteurs**: ce dernier point est celui qui a orienté le plus notre choix vers la luminosité et le volume. En effet les capteurs embarqués disponibles dans les plateformes mobiles fournissent principalement des informations sur :
  — **La lumière ambiante**
  — **Le mouvement**
  — **L'orientation**
  — **La localisation**
  — **Le bruit ambiant**

  Le type de données fourni par les capteurs limite les champs d'action possibles.
  Parmi les composants matériels que nous pouvons gérer et configurer en se basant uniquement sur les données des capteurs et sans apprentissage, nous avons la luminosité de l'écran et le volume des hauts parleurs. Nous pouvons également gérer la Wi-Fi en se basant sur les capteurs de mouvement mais cette gestion peut être sujette à des erreurs de précision. La corrélation entre les capteurs disponibles et les composants

hardwares ciblés est développée dans le section 4.2.2.

4.2.1 Sensor Collection Module SCM

Le SCM est une IL qui s'interface avec les capteurs embarqués illustrés sur la figure 4.3.

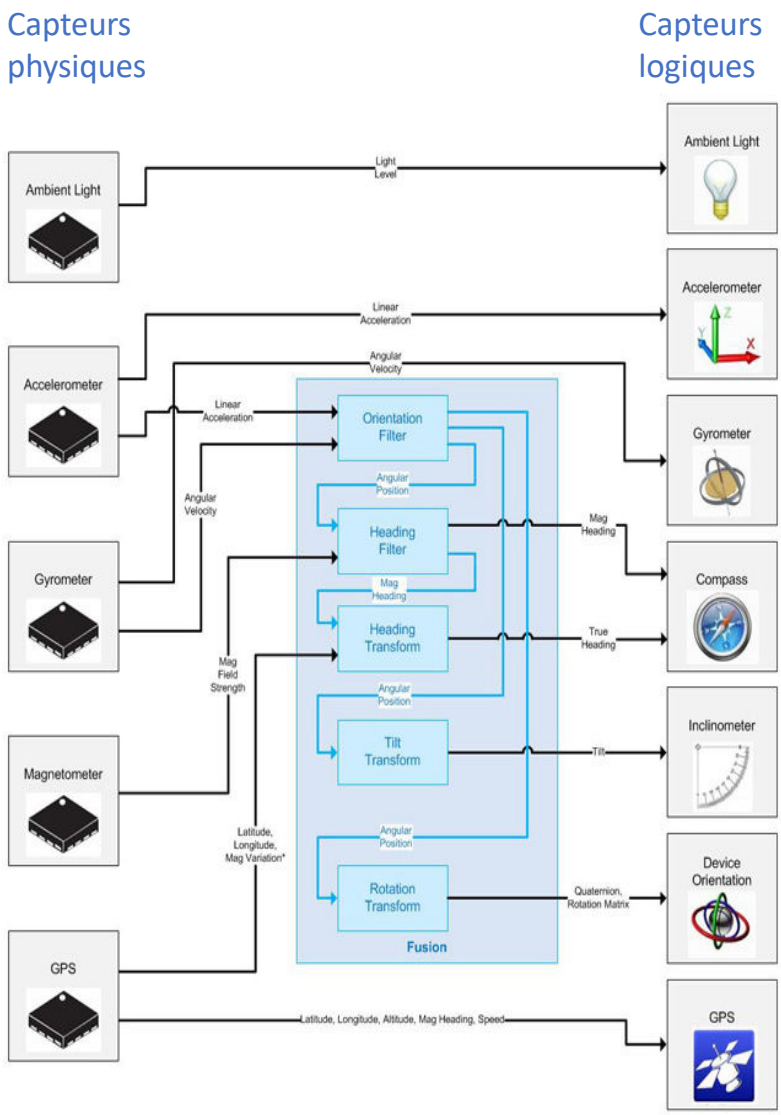

FIGURE 4.3 – Correspondance entre capteurs physiques et abstractions logicielles

Cette figure expose:

1. Les capteurs physiques auxquels correspondent des puces *ad-hoce*. Ces puces hardwares sont représentées à gauche dans la figure 4.3.

2. Parmi les capteurs logiques présents à droite, nous distinguons deux types:
   (a) Les capteurs logiques représentant des abstractions logicielles des capteurs physiques.
   (b) Les capteurs logiques obtenus par la fusion des capteurs physiques.

Les capteurs logiques qui représentent des abstractions des puces *ad-hoc* sont:

1. **Ambient Light Sensor(ALS)**: ce capteur de lumière ambiante est présent dans toutes les plateformes mobiles de nouvelle génération. Il indique le niveau de la lumière ambiante de l'environnement où se trouve le système. La métrique de ce capteur est le *Lux* qui est l'unité de mesure de l'éclairement lumineux. Il caractérise le flux lumineux reçu par unité de surface.
2. **Accélomètre - gyromètre**: L'accéléromètre et le gyromètre sont les capteurs qui mesurent les 6 degrés de liberté d'un système mobile [129].
   - L'accéléromètre enregistre de façon simultanée les accélérations dites statiques comme la gravité et les accélérations dites dynamiques telles que le choc et le mouvement. L'accéléromètre nous fournit la direction dans laquelle le mobile se déplace et par conséquent le déplacement de l'utilisateur. Il ne détecte pas une position, mais une accélération sur les trois axes X, Y et Z. Un accéléromètre mesure les variations de vitesse et les changements de position. Les accéléromètres sont généralement utilisés pour mesurer des petits mouvements. Par exemple, grâce à l'effet de la gravité, sur un mobile, l'accéléromètre nous permet de passer du mode portrait au mode paysage et vice versa.
   - Le gyromètre quant à lui, ne détecte pas un déplacement linéaire le long d'un axe, mais une accélération de la rotation autour d'un axe [54]. Il mesure deux types d'orientation qui sont les mouvements angulaires et les changements de vitesse de rotation. Il ne fournit que des données relatives telles qu'une rotation de 30 degrés vers la droite ou la gauche.
     Le gyromètre peut déterminer l'orientation du système mobile par rapport à la verticale en utilisant l'accéléromètre. Les jeux de course de voitures où le mobile fait office de volant ou encore le fait de secouer son appareil pour éteindre sa musique sont de parfaits exemples de l'utilisation du gyromètre.
3. **Magnétomètre**: ce capteur mesure les champs magnétiques. La terre possède en effet un champ magnétique significatif, ce qui nous permet d'utiliser le magnétomètre comme une boussole. Le magnétomètre possède trois capteurs agissant sur trois axes et permet de disposer d'une véritable boussole embarqué.
4. **Global Positionning System (GPS)**: c'est sans doute le capteur le plus connu et le plus utilisé. Le GPS révolutionne notre quotidien et facilite les déplacements d'un point A à un point B. Le principe de fonctionnement de ce capteur repose sur la mesure de la distance d'un récepteur par rapport à plusieurs satellites [50]. Les satellites sont répartis de telle manière que 4 à 8 d'entre eux soient toujours visibles. Chaque satellite émet un

signal, capté sur terre par le récepteur. Ce signal permet ainsi de mesurer très précisément la distance séparant l'émetteur du récepteur grâce au temps de parcours. Avec la réception des signaux de quatre satellites qui sont répartis en trois pour obtenir le point d'intersection des trois sphères et un quatrième pour la synchronisation du temps, le récepteur mobile est capable de calculer sa position géographique par triangulation.

Les capteurs logiques obtenus par fusion ne possèdent pas de puces dédiées. La fusion des données qui émanent des différentes capteurs physiques est réalisée en quatre étapes:

1. La combinaison des données brutes de plusieurs capteurs physiques, en particulier l'accéléromètre, le gyromètre et le magnétomètre.
2. L'application de filtres et de transformations mathématiques tels que les filtres de *Kalman* [78]. Ces filtres servent à corriger les limitations naturelles des capteurs.
3. Le calcul des données qui seront plus facilement exploitables par l'homme.
4. La représentation sous formes d'abstractions logiques.

Parmi ces capteurs obtenus par fusion, nous en recensons:

1. **Inclinomètre**: ce capteur logique résulte d'un filtre d'orientation provenant de l'accéléromètre qui subit une transformation d'inclinaison.
2. **Compass**: la boussole numérique est un capteur indispensable car cette dernière est directement liée au GPS. Cela permet de naviguer par exemple via des applications tel que *Google Maps*. Les valeurs de la boussole numérique sont générées à partir d'une combinaison entre le magnétomètre, le gyromètre et le GPS.
3. **Device Orientation**: ce capteur fournit des informations sur l'orientation du mobile. Il est le résultat de la combinaison entre l'accéléromètre et le gyromètre.

Le SCM récupère toutes les valeurs des capteurs existants en temps réel. Entre deux collectes, nous avons un *time-out* fixé à *1000 ms* par défaut. Ce *time-out* est évidemment ajustable à la hausse ou à la baisse selon nos besoins. Néanmoins, la réduction de cette période d'entre deux collectes n'est pas pertinente et n'offrira aucun gain énergétique, car l'utilisateur ne se rend pas compte d'un changement qui opère au moins d'une seconde. L'augmenter peut entrainer des latences, impactant ainsi négativement la satisfaction de l'utilisateur.

Dans un premier temps, le rôle du SCM se limite à s'interfacer avec toutes ces abstractions logiques des capteurs présents à droite dans la figure 4.3. Cet interfaçage est réalisé via une API que nous avons développée en collaboration avec *Intel*. Les données de tous les capteurs sont analysées et filtrées à travers notre étude préliminaire comme cela est présenté dans la section suivante. À l'issue de cette étude, le SCM sera notifié pour s'interfacer uniquement avec les capteurs sélectionnés.

4.2.2 Étude des données des capteurs et SPM

. La phase de collecte de données est suivie par une phase d'étude des données des capteurs. Cette étude permet de sélectionner les capteurs les plus pertinents et filtrer les métriques à utiliser dans notre approche. Le fonctionnement

du SPM se base sur les résultats de cette étude pour la détection des différents scénarios offrant des opportunités de réduction de puissance. Avant de détailler la phase d'études, nous présentons les scénarios détectables par le SPM:

- **Un écran difficile ou impossible à regarder**: cela se produit dans le cas où le mobile est trop incliné ou en mouvement excessif. Nous en déduisons que l'utilisateur ne peut pas être dans une position confortable pour utiliser son système. Par conséquent, il subsiste une opportunité de réduire la consommation de puissance en baissant la luminosité de l'écran.
- **Lumière ambiante basse**: quand le système se trouve en position normale, nous gérons la luminosité de l'écran en fonction de la lumière ambiante de l'environnement. Le niveau de luminosité sélectionné est inversement proportionnel à la lumière ambiante.
- **Un environnement peu bruyant**: dans le cas où l'environnement dans lequel la plateforme mobile se trouve est un environnement calme, nous procédons à la baisse du volume qui n'impactera en rien l'expérience de l'utilisateur. Si le système est dans un environnement avec des nuisances sonores, le contrôle du volume des hauts parleurs est laissé à l'utilisateur ou à l'OS.

La figure 4.4 reprend ces scénarios mentionnés ci-dessus.

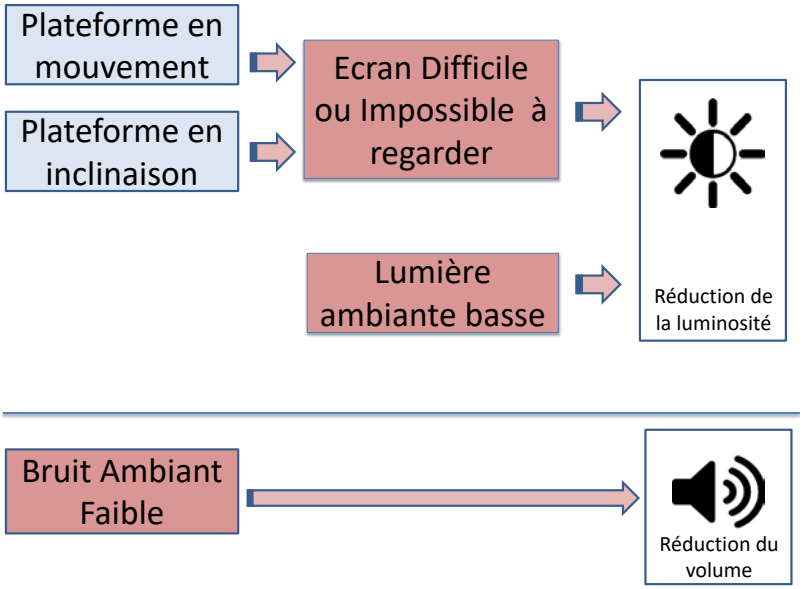

FIGURE 4.4 – Scénarios détectés par le SPM

Pour pouvoir corréler les scénarios cités ci-dessus avec les données des capteurs, nous consultons leurs valeurs pour en garder que ceux qui serviront à la détection des scénarios. Cette phase d'étude comporte deux actions:

1. Choix préliminaire des capteurs.
2. Comparaison des données des capteurs.

4.2.2.1 Choix préliminaire des capteurs

Dans un premier temps, l'étude des capteurs et de leurs données a pour objectif:

1. Éviction des capteurs qui ne seront pas exploités pour la gestion de la luminosité et le volume: certaines capteurs ne fournissent pas d'informations pertinentes pour la détection des scénarios susmentionnés. Collecter les données exposées par ces capteurs engendrera des latences et un surplus en matière de puissance consommée.
2. Regroupement des capteurs qui fournissent le même type d'information: certains capteurs fournissent des informations similaires pour la détection de nos scénarios comme l'accéléromètre et le gyromètre. Utiliser deux capteurs qui fournissent la même information ne représente aucune plus value. Bien au contraire, cela augmente la sollicitation des ressources de calcul et de mémoire.

Nous procédons à un filtrage préliminaire pour garder un nombre restreint de capteurs qui sont:

1. **Ambient Light Sensor (ALS)**
2. **M**icrophone: nous utilisons ce périphérique pour la collecte du bruit ambiant. Bien que le microphone ne soit pas un capteur à proprement dit, il permet toutefois de quantifier le niveau du bruit ambiant.
3. **Accéléromètre Vs gyromètre**: ces deux capteurs fournissent des informations sur le mouvement du système.
4. **Inclinomètere Vs boussole (*compass*)**: ces deux capteurs indiquent le niveau d'inclinaison de notre plateforme.
5. **Device Orientation**: ce capteur logique a été ignoré car le système d'exploitation n'expose aucune de ses métriques. Nous pouvons nous passer de l'utilisation de ce capteur car il fournit le même type d'information que le gyromètre.
6. **GPS**: ce capteur est également ignoré dans SBOC à cause du caractère privé des données de l'emplacement. En effet, les utilisateurs sont très réticents à l'idée de se faire géolocaliser.

Après avoir regroupé les capteurs en fonction du type d'information fourni, nous procédons aux choix concernant les deux couples de capteurs suivants: **Inclinometere - compass** et **Accéléromètre - gyromètre**.

4.2.2.2 Comparaison des données des capteurs

Pour départager entre l'inclinomètre et la boussole, nous n'avons pas eu à réaliser une étude comparative, notre choix s'est logiquement orienté vers l'inclinomètre pour les contraintes que possède la boussole. En effet, cette dernière fournit des données relatives à la force du champ magnétique existant autour du système. La boussole fournit des données qui sont constamment variables peu importe l'inclinaison du système. À contrario, l'inclinomètre expose des données absolues à l'orientation. Cette différence cruciale fait porter notre choix vers l'inclinomètre. Ce dernier expose deux métriques distinctes qui informent de l'inclinaison en degrés sur les deux axes (X,Y).

La figure 4.5 représente les variations d'inclinaison sur les axes X et Y ainsi que la somme de leurs valeurs absolues. La figure 4.6 illustre le seuil d'inclinaison.

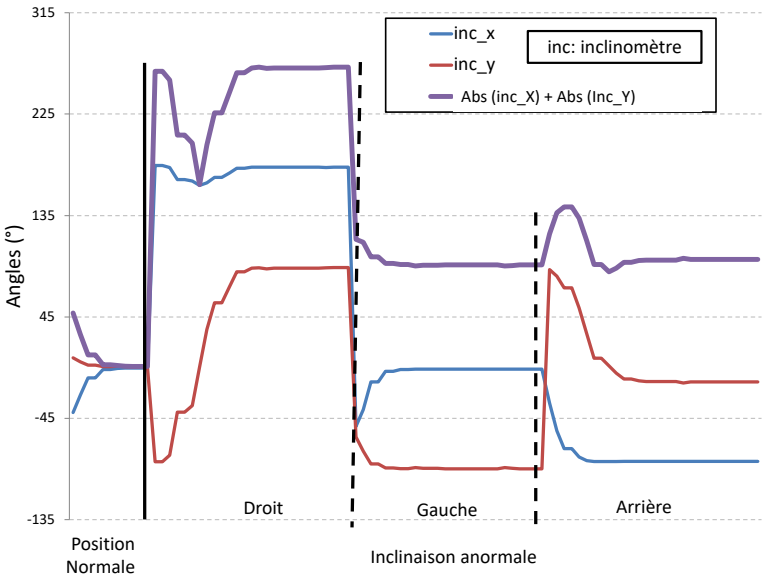

FIGURE 4.5 – Les variations des métriques de l'inclinomètre

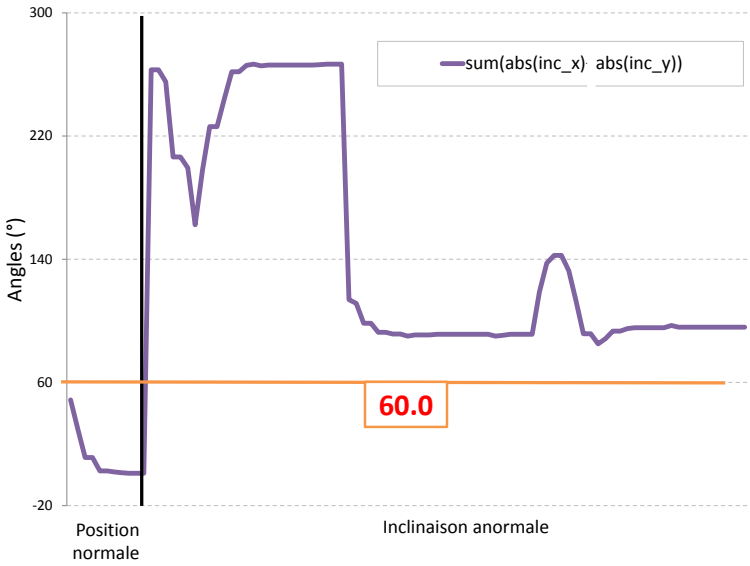

FIGURE 4.6 – Seuil de l'inclinaison

La figure 4.5 montre les variations d'inclinaison en fonction de la position du système. Les données fournies par l'inclinomètre sont stables lorsque le système

est en position normale, tandis que nous remarquons de grandes variations lors des différentes inclinaisons anormales. Dans la figure 4.6, nous avons regroupé tous les types d'inclinaisons pour avoir uniquement deux états, l'état normal et l'inclinaison anormale. Nous avons fixé le seuil d'inclinaison à 60° en prenant en compte la somme des valeurs absolues des inclinaisons X et Y. Ce seuil de 60° nous aide clairement à différencier entre une position normale et une inclinaison anormale du système.

Le second choix se tenait entre le gyromètre et l'accéléromètre comme l'illustre la figure 4.7.

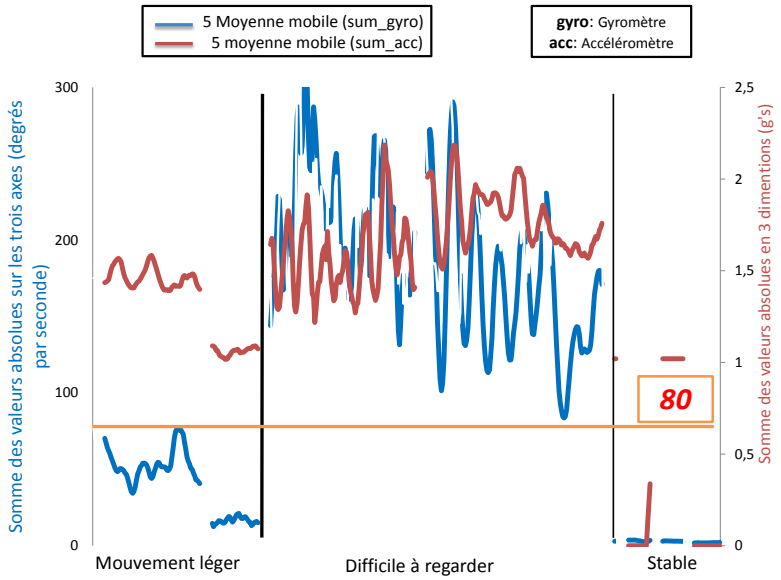

FIGURE 4.7 – Comparaison entre les métriques du gyromètre et l'accéléromètre

Choisir entre l'accéléromètre et le gyromètre pour déterminer le mouvement a nécessité une étude comparative contrairement au premier choix entre l'inclinomètre et la boussole. Dans ce qui suit, nous comparons les valeurs de l'accéléromètre avec le gyromètre lorsque le système est dans trois états: "mouvement léger", "difficile à regarder" et "stable".

La figure 4.7 expose la moyenne mobile d'ordre 5 pour la somme des valeurs absolues en *degrés/seconde* sur les trois axes X,Y et Z pour le gyromètre (Bleu). Les variations en rouge représentent également la moyenne mobile d'ordre 5 pour la somme des valeurs absolues de la gravité terrestre *(G's)* sur les trois axes X,Y et Z pour l'accéléromètre. Nous avons utilisé un lissage par moyenne mobile pour faire apparaître l'allure de la tendance des valeurs du gyromètre et l'accéléromètre. Cela nous permet de gommer les variations transitoires de manière à en souligner les tendances à plus long terme.

La comparaison des moyennes mobiles des métriques exposées par les deux capteurs nous montre clairement que le choix le plus approprié est celui du gyromètre illustré en bleu dans la figure. En effet, nous constatons que lors du passage d'un état à un autre, les valeurs du gyromètre sont plus sensibles. L'accélromètre ne différence pas entre les deux états "mouvement léger et "difficile

à regarder". Contrairement à l'accéléromètre, le gyromètre fournit des valeurs qui permettent la distinction entre les trois états.

Nous comparons également à titre indicatif les écarts-types des métriques des deux capteurs:

- Gyromètre: (x, y, z) =(89.42, 57.80, 54.95)
- Accéléromètre (x, y, z)) =(0.57, 0.37, 0.52)

Nous notons là aussi, que l'écart type retourné par le gyromètre est plus sensible et par conséquent plus informatif. Cette comparaison conforte notre choix concernant l'utilisation du gyromètre pour la détection du mouvement de la plateforme.

Après avoir sélectionné le gyromètre, nous déterminons un seuil qui sépare les deux états "mouvement léger" et "système en position difficile à regarder". Ce seuil est fixé à 80, ce qui représente la valeur absolue de la somme des valeurs du gyromètre en degrés/seconde. La seuil est de 80 car comme le rapporte la figure 4.7, cette valeur est celle qui sépare le plus clairement les différents états de mouvement. Lorsque les valeurs du gyromètre sont supérieures à 80, nous en concluons que le système est dans un état " difficile à regarder".

Après avoir sélectionné les capteurs et les seuils relatifs à la position du système, nous nous intéressons au capteur de lumière ambiante. En effet, dans le cas où notre plateforme mobile est dans une position normale, nous devons détecter la lumière ambiante de l'environnement du système comme le montre le tableau 4.1.

Tableau 4.1 – Détection du niveau de la lumière ambiante en fonction de la métrique de ALS

lux	État	lux	État
100000	Lumière soleil	251.189	intérieur sombre
31622.8	Lumière soleil	223.872	intérieur sombre
28183.8	ciel couvert	199.526	intérieur sombre
100000	ciel couvert	177.828	intérieur sombre
8912.51	intérieur lumineux	112.202	intérieur sombre
1778.28	intérieur lumineux	100	intérieur sombre
1584.89	intérieur lumineux	79.4328	intérieur sombre
1412.54	intérieur lumineux	70.7946	intérieur sombre
1258.93	intérieur lumineux	39.8107	intérieur sombre
1122.02	intérieur lumineux	35.4813	intérieur sombre
1000	intérieur lumineux	31.6228	intérieur sombre
891.251	intérieur lumineux	28.1838	intérieur sombre
794.328	lumière intérieur	25.1189	intérieur sombre
707.946	lumière intérieur	22.3872	intérieur sombre
630.957	lumière intérieur	19.9526	intérieur sombre
562.341	lumière intérieur	12.5893	intérieur sombre
501.187	lumière intérieur	11.2202	intérieur sombre
446.684	lumière intérieur	10	intérieur sombre
398.107	lumière intérieur	8.91251	obscurité
354.813	lumière intérieur	7.07946	obscurité
316.228	lumière intérieur	6.30957	obscurité

Dans le tableau ci-dessus, la case *lux* représente les valeurs fournies par

l'ALS. Chaque valeur est corrélée à différents états allant du plus lumineux "lumière soleil" jusqu'à l'état le plus sombre "obscurité".

Après avoir corrélé les données des capteurs avec la positon du système et la lumière ambiante, des études expérimentales pour déterminer le seuil du bruit ambiant ont été menées. Ce seuil servira à la gestion du volume des hauts parleurs. Pour ce faire, nous exploitons un bruit ambiant dont les amplitudes varient. Nous avons ensuite analysé les données fournies par le microphone. Les trois métriques exposées sont *mic min, mic avg et mic max* comme le montre la figure 4.8.

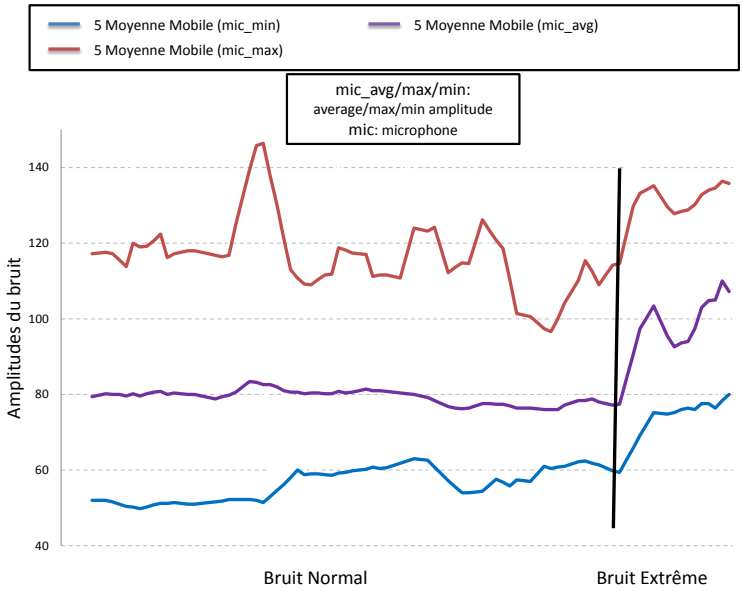

FIGURE 4.8 – Résultats de l'étude expérimentale sur les données du microphone

Nous notons dans la figure ci-dessus que la courbe rouge correspondante à *mic max* contient plusieurs oscillations. Son évolution est peu stable lorsque le niveau du bruit ambiant est normal. Les deux courbes correspondantes à *mic avg* et *mic min* suivent le même schéma d'évolution. Elles sont stables lors du bruit normal et augmentent à la même allure lorsque le bruit ambiant devient extrême. L'instabilité de la courbe générée par *mic max* nous pousse à l'écarter pour la détermination du seuil du bruit extrême. Nous ne gardons que *mic min* et *mic avg*.

La figure 4.9 indique les seuils de l'amplitude du bruit ambiant. Cette figure montre que lorsque *mic min* est au dessus d'une amplitude de 65 et que *mic avg* indique une amplitude supérieure à 85. Nous en concluons que le système est dans un environnement bruyant.

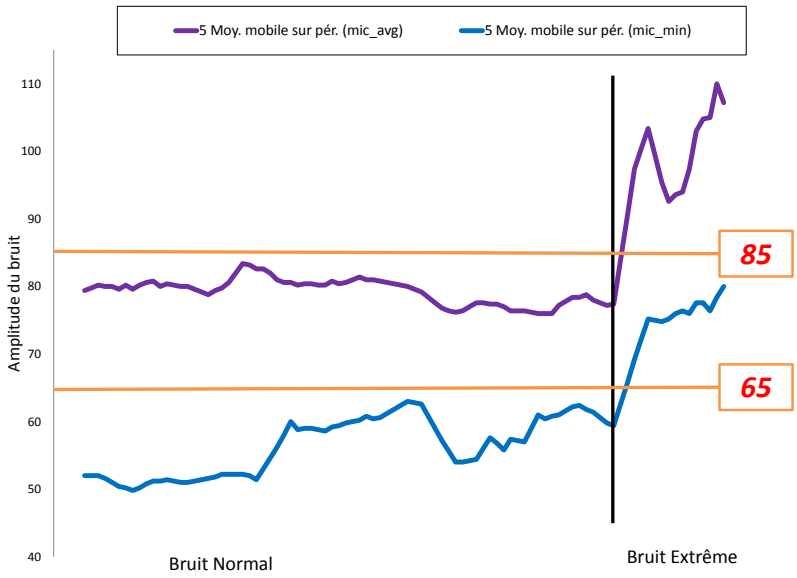

FIGURE 4.9 – Seuils des amplitudes du bruit ambiant

Ces expérimentations permettent de choisir entre les différents capteurs disponibles dans notre plateforme. Le SCM est reconfiguré pour collecter que les données du microphone, l'ALS, l'inclinomètre et le gyromètre. Ces expérimentation permettent également de déterminer les seuils d'inclinaison, de mouvement, de lumière ambiante et de bruit ambiant. Les valeurs de ces différents seuils sont stockées dans la mémoire interne du SPM et sont comparés en temps réel avec chaque nouvelle métrique collectée par le SCM. Les résultats de cette comparaison déterminent dans quel état se trouve le système, ainsi que son environnement à un instant T. La figure 4.10 schématise les entrées et les sorties du SPM.

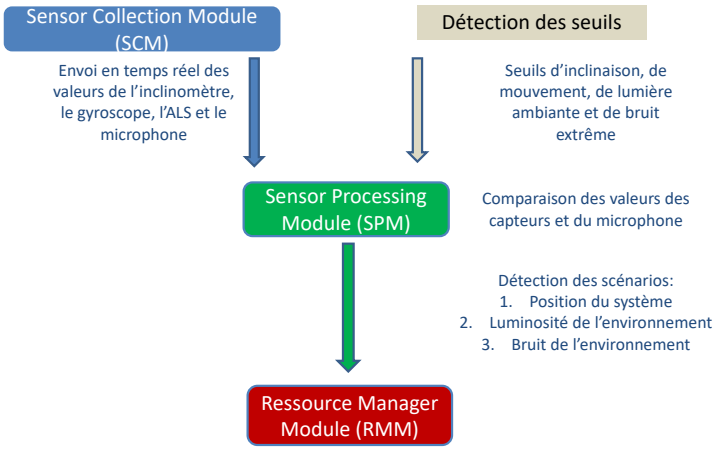

FIGURE 4.10 – Les entrées / sorties du SPM

Le SPM a été implémenté sous forme d'AL et son fonctionnement est comme suit:

- Stockage des différents seuils résultants de la phase d'expérimentation.
- Importation des valeurs collectées par le SCM.
- Comparaison des valeurs des capteurs avec les seuils d'inclinaison, de mouvement et de bruit ambiant.
- Si la plateforme est en position anormale, c'est à dire incliné ou en mouvement, l'information est transmise au RMM pour baisser la luminosité.
- Sinon, la gestion de la luminosité est réalisée en fonction du capteur de lumière ambiante comme montré dans le tableau 4.1.
- Pour la gestion du volume, le SPM compare également en temps réel les valeurs du microphone pour notifier le RMM de l'action à réaliser.

4.2.3 Ressource Manager Module (RMM)

Les actions réalisées par ce module représentent la dernière phase de SBOC. Le RMM est implémenté sous forme d'AL dont le rôle demeure simpliste dans cette première contribution. La gestion de la luminosité et du volume des hauts parleurs se fait sur la base des résultats de la comparaison réalisée par le SPM. La configuration des ressources hardwares à la baisse est réalisée lorsque le système n'est pas dans un état normal. Par "état normal", nous désignons les conditions habituelles d'utilisation du système en matière d'inclinaison, de mouvement, de bruit ambiant, etc. Lorsque le système passe d'une position anormale à une position normale, le RMM reconfigure la luminosité en se basant sur la lumière ambiante. La gestion du volume se fait également de la même façon, le volume est reconfiguré à son niveau initial lors du passage d'un environnement calme à un environnement bruyant. La figure 4.11 schématise les actions du RMM.

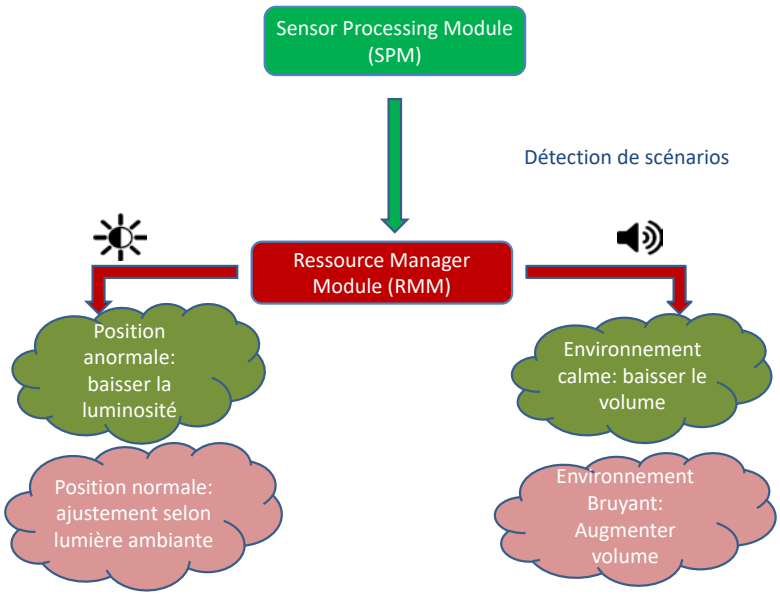

FIGURE 4.11 – Mode de fonctionnement du RMM

Pour ajuster les niveaux de la luminosité et du volume, le RMM manipule les drivers de l'écran et du haut parleur pour appliquer les heuristiques mises en place. Le tableau 4.2 montre le mode de fonctionnement du RMM basé sur le SPM pour la configuration de la luminosité de l'écran.

Tableau 4.2 – Heuristiques pour la gestion de la luminosité

Seuil	Catégorie	État	Lum
Gyro >= 80	Mouvement	Écran difficile à regarder	20%
Gyro >80 et incl >8	Mouvement	Mouvement léger	30%
Gyro <8 et incl >= 60	Immobile	Inclinaison anormale	20%
Gyro >8 et incl <60	Immobile	Position normale	ALS

En se basant sur la comparaison des valeurs des capteurs avec les seuils enregistrés dans le SPM, ce dernier détermine l'état du système. Lorsque le système est en mouvement avec un écran difficile à regarder ou anormalement incliné, nous supposons que l'utilisateur n'est pas entrain d'utiliser son système et nous baissons par conséquent la luminosité à son niveau minimal de 20 %. Lorsque le système est en mouvement léger, la luminosité est baissée à 30 %, compte tenu de la nature du mouvement. Finalement, lorsque la position du système est normale, le RMM configure la luminosité selon la lumière ambiante comme le montre le tableau 4.3.

Tableau 4.3 – Table de correspondance pour la gestion de la luminosité en fonction de la lumière ambiante

lux	État	Lum	lux	État	Lum
100000	Lumière soleil	100	251.189	intérieur sombre	36
31622.8	Lumière soleil	100	223.872	intérieur sombre	33
28183.8	ciel couvert	100	199.526	intérieur sombre	33
100000	ciel couvert	100	177.828	intérieur sombre	30
8912.51	intérieur lumineux	100	112.202	intérieur sombre	30
1778.28	intérieur lumineux	100	100	intérieur sombre	25
1584.89	intérieur lumineux	93	79.4328	intérieur sombre	25
1412.54	intérieur lumineux	90	70.7946	intérieur sombre	25
1258.93	intérieur lumineux	86	39.8107	intérieur sombre	25
1122.02	intérieur lumineux	75	35.4813	intérieur sombre	25
1000	intérieur lumineux	65	31.6228	intérieur sombre	25
891.251	intérieur lumineux	55	28.1838	intérieur sombre	20
794.328	lumière intérieur	50	25.1189	intérieur sombre	20
707.946	lumière intérieur	46	22.3872	intérieur sombre	20
630.957	lumière intérieur	46	19.9526	intérieur sombre	20
562.341	lumière intérieur	43	12.5893	intérieur sombre	20
501.187	lumière intérieur	43	11.2202	intérieur sombre	20
446.684	lumière intérieur	40	10	intérieur sombre	20
398.107	lumière intérieur	40	8.91251	obscurité	20
354.813	lumière intérieur	40	7.07946	obscurité	20
316.228	lumière intérieur	36	6.30957	obscurité	20

Dans le tableau 4.3 ci-dessous, nous avons repris les différents états relatifs à la lumière ambiante en y ajoutant le niveau de luminosité adéquat. Ces niveaux de luminosité y sont présentés dans un un ordre décroissant car la luminosité de l'écran est inversement proportionnelle à la lumière ambiante de l'environnement. La luminosité de l'écran est à son maximum quand le système est exposé à la lumière du soleil et est configurée à son niveau minimal quand le système est dans un environnement obscure.

Pour la gestion du volume, le tableau 4.4 expose les heuristiques utilisées.

Tableau 4.4 – Heuristiques pour la gestion du volume

Seuil	Bruit ambiant	Volume
mic_min <65 et mic_avg <85	Bruit normal	20%
mic_min >= 65 ou mic_avg >= 85	Bruit extrême	100%

La gestion du volume dans cette contribution est assez simpliste. Nous nous contentons de baisser le volume lorsque l'environnement est bruyant. Toutefois, la fonctionnalité proposée demeure intéressante car aucune adaptabilité n'est proposée par l'OS en matière de gestion de volume. Le gain de puissance offert par SBOC est proportionnel au niveau de la luminosité de l'écran et au volume des hauts parleurs comme le détaille la section suivante.

4.3 Évaluation et coût de SBOC

Dans la partie évaluation de cette première contribution, nous mettons l'accent sur deux points:

1. Le premier point est l'évaluation de SBOC dans deux scénarios : le premier est spécifique à la gestion de la luminosité et le second pour la gestion du volume. L'objectif est de montrer les éventuelles économies d'énergie apportées par SBOC.
2. Le second point représente le coût de SBOC : nous nous focalisons sur la phase de collecte réalisée par le SCM. Nous étudions uniquement le coût de la collecte car c'est la seule phase qui provoque un surplus de consommation de puissance discernable.

4.3.1 Scénarios, configuration et jeu de test

Nous présentons en premier lieu des résultats préliminaires obtenus en gérant la luminosité de l'écran avec SBOC. Cette gestion se fait uniquement en fonction de la position du système sans prendre en considération la lumière ambiante.

La figure 4.12 illustre les résultats de ces expérimentations.

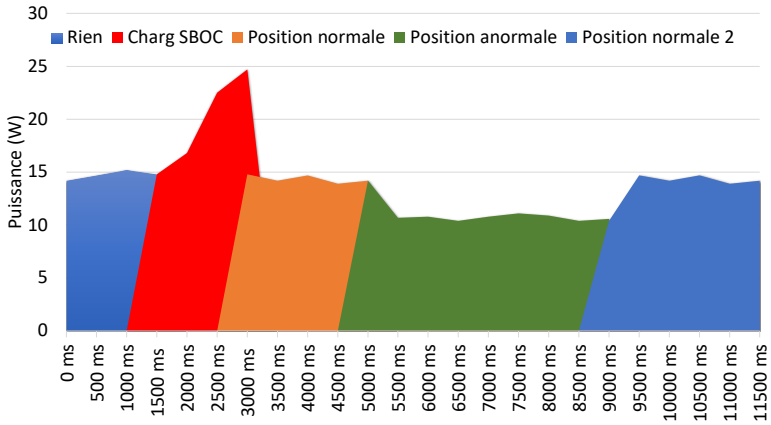

FIGURE 4.12 – Consommation de puissance avec la gestion de luminosité proposée avec SBOC

Nous avons cinq parties dans la figure ci-dessus, chaque partie correspondant à une position du système:

- La première partie du graphe en bleu représente notre système mobile en position normale avant le déploiement de SBOC.
- La partie rouge illustre le chargement de SBOC. Nous notons un pic de puissance qui résulte de l'interfaçage avec les différents capteurs et le démarrage des modules. Ce surplus de puissance est étudié dans la section 4.3.2.
- La partie orange représente la consommation de puissance du système en position normale, avec SBOC chargé et lancé en arrière plan. Nous remarquons qu'une fois SBOC déployé, ce dernier ne génère aucun surplus de puissance lorsque le système est en position normale.
- La partie verte désigne la consommation de puissance quand le système est en position anormale pour l'utilisateur. Nous rappelons qu'une position anormale correspond à une inclinaison ou un mouvement brusque. Le RMM réduit la luminosité de l'écran et par conséquent, une réduction de la puissance consommée en résulte.
- La dernière partie en bleu représente la consommation de puissance lorsque le système est à nouveau dans une position normale avec SBOC fonctionnant en arrière plan.

Cette première expérimentation est préliminaire et permet de montrer comment SBOC réagit en fonction de la position du système. Nous exposons plus de détails sur les actions de SBOC dans les parties suivantes.

4.3.1.1 Scénario et configuration pour la gestion de la luminosité

Dans le but d'estimer l'économie de puissance apportée par SBOC en configurant la luminosité, nous simulons le scénario illustré dans le tableau 4.5. Ce scénario est composé de cinq phases, dont chacune dure 30 secondes. Nous sélectionnons pour chaque phase une position pour le système et un niveau de lumière ambiante.

Tableau 4.5 – Scénario pour la gestion de la luminosité

Phases	Durée (S)	Orientation / Mouvement	Lumière ambiante
1/5	[1 - 31]	Position normale	Intérieur sombre
2/5	[31 - 61]	Position normale	Intérieur lumineux
3/5	[61 - 91]	Écran difficile à regarder	Intérieur lumineux
4/5	[91 - 121]	Mouvement moyen	Intérieur lumineux
5/5	[121 - 151]	Inclinaison anormale	Intérieur lumineux

Nous appliquons à ce scénario les quatre configurations suivantes illustrées dans le tableau 4.6.

Tableau 4.6 – Configurations appliquées au scénario de gestion de la luminosité

Configurations	Niveau de luminosité
MaxBlight	100%
DefaultBlight1	Basé sur la luminosité ambiante: 36% – 100%
SBlight	20% – 100 %
MinBlight	20%

Ces configurations sont:

- *MaxBlight* configure la luminosité de l'écran à son niveau maximal de 100 %.
- *DefaultBlight* est une configuration basée uniquement sur la lumière ambiante proposée par le système d'exploitation. Le niveau de luminosité y varie entre 36% et 100%.
- *SBlight* est la configuration proposée avec SBOC présentée tout au long de ce chapitre. *SBlight* est basée sur la position du système mobile, son mouvement et la lumière ambiante de l'environnement.
- *MinBlight* attribue au système un niveau de luminosité minimal de 20%.

Les configurations *MinBlight* et *MaxBlight* sont utilisées uniquement pour montrer le gap en matière de puissance consommée entre le niveau de luminosité le plus faible et le plus élevé. En pratique, il est rare de maintenir la luminosité en permanence à ces niveaux là.

Pour mesurer avec précision la consommation de puissance des quatre configurations et l'impact de la luminosité sur la consommation totale, nous activons le profil d'économie d'énergie. Nous gardons la fonctionnalité de la luminosité adaptative proposée par l'OS uniquement durant l'utilisation de la configuration *DefaultBlight*.

La figure 4.13 présente les résultats de la consommation de puissance obtenus.

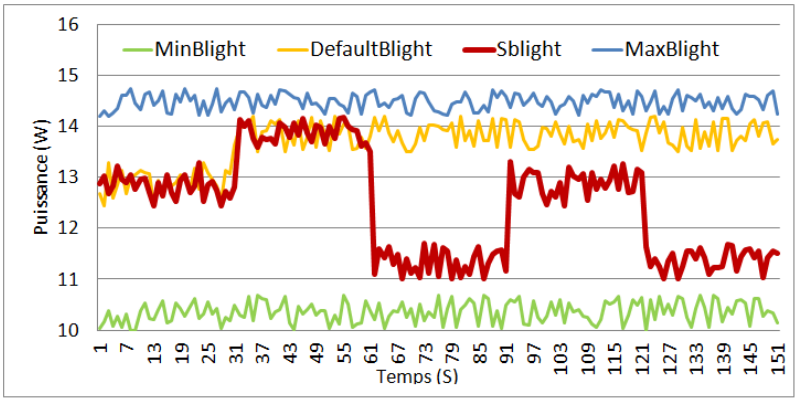

FIGURE 4.13 – Résultats de la consommation de puissance pour le scénario de la luminosité

Nous notons principalement que la consommation de puissance durant la première et la seconde phase avec *SBlight* est similaire à celle engendrée par *DefaultBlight*. Cette similarité se justifie par la la position du système. En effet, lorsque le système est en position normale, SBOC adapte la luminosité en fonction de la lumière ambiante et de manière similaire à celle du système d'exploitation. Lors des trois dernières phases du scénario, le système est en position anormale. *SBlight* offre des économies de puissance importantes, en comparaison avec *DefaultBlight*. Lorsque l'écran du système est difficile à regarder ou anormalement incliné, le RMM baisse la luminosité à son niveau le plus bas. Lorsque le système est en mouvement léger, la luminosité est baissée à 30 %. La raison de cette différence en matière de puissance est que l'OS ne prend pas en considération la position du système pour la gestion de la luminosité. Parmi toutes les configurations présentées, la seule configuration qui propose une plus faible consommation que *SBlight* est *MinBlight* qui maintient le niveau de luminosité à 20%. La figure 4.14 présente la consommation d'énergie normalisée qui résulte des quatre configurations.

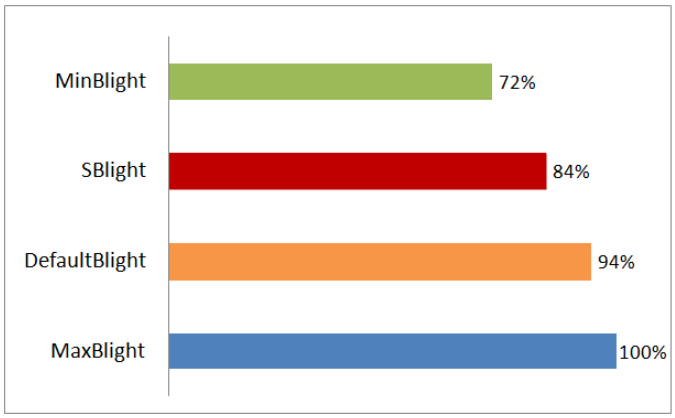

FIGURE 4.14 – Consommation d'énergie normalisée pour la gestion de la luminosité

Pour le scénario de gestion de luminosité, la réduction d'énergie totale offerte par SBOC avec la configuration *SBlight* est de 16%. Ce gain est plus élevé de 10% par rapport à *DefaultBlight* qui est basée uniquement sur la lumière ambiante. Lorsque le système est en position anormale, SBOC offre un gain total évalué à plus de 20% . Nous utilisons la même démarche pour statuer de l'efficacité de SBOC en matière de gestion du volume.

4.3.1.2 Scénario et configurations pour la gestion du volume

Pour jauger l'efficacité de SBOC pour la configuration du volume des hauts-parleurs, nous proposons le scénario illustré dans la figure 4.7. Ce scénario est composé de trois phases qui durent également 30 secondes chacune. Nous avons un bruit ambiant extrême pour la première et la troisième phase, et un bruit ambiant normal pour la seconde phase. Pour simuler le bruit extrême, nous avons sélectionné un son de chutes d'eau [1]

Tableau 4.7 – Scénario pour la gestion du volume

Phases	Intervalle de temps	Niveau du bruit ambiant
1/3	[1 s - 31 s]	Bruit extrême
2/3	[31 s - 61 s]	Environnement calme
3/3	[61 s - 91 s]	Bruit extrême

Nous appliquons à ce scénario trois configurations présentées dans le tableau 4.8. Chacune de ces configurations va être appliquée durant les trois phases définies ci-dessus.

Tableau 4.8 – Configurations appliquées au scénario pour la gestion du volume

Configurations	Niveau de volume
DefaultVol	100%
SVol	20%–100%
MinVol	20%

Les configurations spécifiques à la gestion du volume sont:

1. *DefaultVol* représente la configuration par défaut du volume proposée par l'OS. Le niveau de ce dernier est de 100%. L'OS ne fournit aucune gestion optimisée pour le volume des hauts parleurs.
2. *Svol* est la configuration que nous proposons avec SBOC basée sur le seuil du bruit ambiant. Le RMM configure le volume des hauts parleurs selon le niveau de bruit ambiant comme illustrée dans la figure 4.3.
3. *MinVol* représente le niveau de volume minimal des hauts parleurs qui est fixé à 20%.

1. www.youtube.com/watch?v=Qaw0x0QF2HM& feature=g-hist

Dans le but de mesurer avec précision la consommation de puissance engendrée par chaque configuration, nous sélectionnons le profil d'économie d'énergie proposé par le système d'exploitation. Nous désactivons également la fonctionnalité d'adaptation de la luminosité et nous fixons son niveau à 20%. Cela permet d'éviter toute interférence dans l'étape de mesure. La figure 4.15 représente les consommations de puissances obtenues avec les trois configurations.

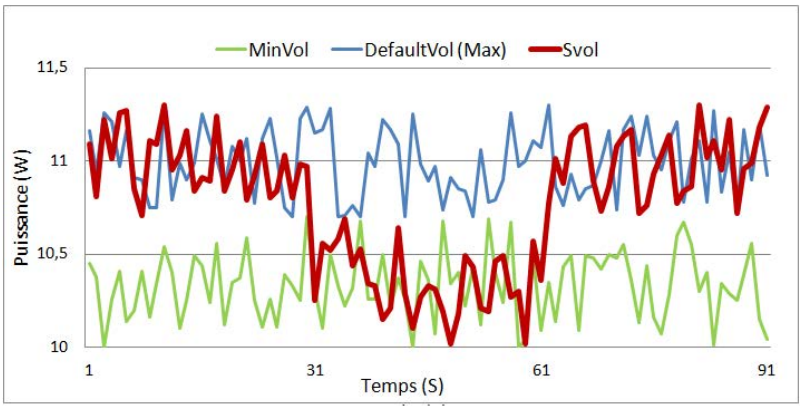

FIGURE 4.15 – Consommation de puissance des configurations du volume

Nous notons à partir de la figure ci-dessus que la consommation de puissance avec *Svol* varie en fonction du niveau de bruit ambiant. Lorsque le niveau de bruit ambiant est extrême, la puissance consommée avec notre configuration est similaire à *DefaultVol*. En effet, nous pouvons remarquer que les courbes bleue et rouge sont quasiment au mêmes niveaux durant la première et la troisième phase. Dans la seconde phase, avec un bruit ambiant faible les résultats de la consommation de puissance avec *Svol* et *MinVol* sont similaires. Cette similarité est due à la réduction du niveau de volume à 20% compte tenu du calme de l'environnement.

La figure 4.16 représente la consommation d'énergie normalisée des trois configurations.

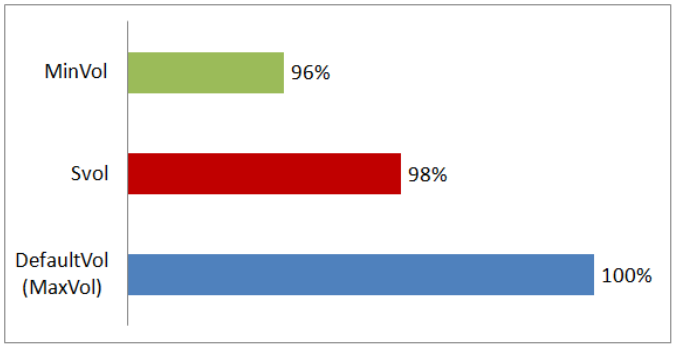

FIGURE 4.16 – Consommation d'énergie normalisée pour les configurations du volume

Nous notons que la configuration *Svol* de SBOC offre un gain de 2.70% alors qu'en réduisant le volume au maximum, l'économie d'énergie totale atteignable est de 4%. En pratique, il est très rare pour un utilisateur de maintenir le volume de son système à un niveau aussi bas. Malgré que la configuration proposée pour la gestion du volume demeure simpliste, nous arrivons néanmoins à combler le manque d'adaptabilité de l'OS en fonction du bruit ambiant. Si l'utilisateur sélectionne un niveau inférieur à 100% lorsqu'il se trouve dans un environnement bruyant, SBOC configure le volume à la valeur sélectionnée par l'utilisateur.

Nous évaluons dans la partie suivante le coût de la collecte de données de SBOC.

4.3.2 Coût de SBOC

Dans cette dernière partie, nous présentons deux graphes qui illustrent le coût de SBOC en matière de consommation de puissance.

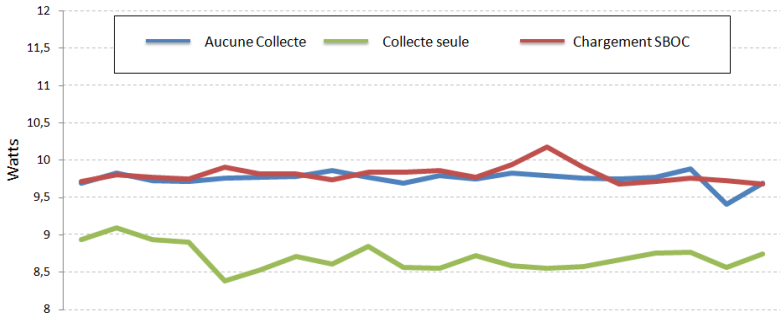

FIGURE 4.17 – Coût de SOC

Dans la figure 4.17:

- Le graphe bleu indique la consommation de puissance lorsqu'aucune donnée n'est collectée. La puissance électrique consommée par le système est d'environ 8.7 W.
- Le graphe rouge représente la consommation de puissance lorsque les données sont collectées sans l'intervention de SBOC. Cela correspond à l'utilisation du système par défaut avec les capteurs embarqués actifs. La moyenne de la puissance consommée est de 9.75 W.
- Le graphe vert englobe la consommation de puissance qui résulte de la gestion réalisée par SBOC. La consommation de puissance est de 9.85 W.

Nous remarquons là aussi quelques fluctuations durant les mesures causées par d'autres ressources comme le processeur et la connexion réseau. La principale information apportée par ces mesures est que SBOC engendre un surplus négligeable de puissance électrique. Lorsque nous comparons la puissance consommée par la collecte seule et la puissance consommée avec SBOC en arrière plan, nous concluons que SBOC a un impact minime sur les performances du système et la satisfaction de l'utilisateur.

La figure 4.18 représente le coût engendré par la collecte des métriques du bruit ambiant via le microphone. Pour la collecte de ces métriques, aucun sur-

plus de puissance n'a été observé. Cela s'explique par le fait que le microphone reste généralement actif durant l'utilisation du système.

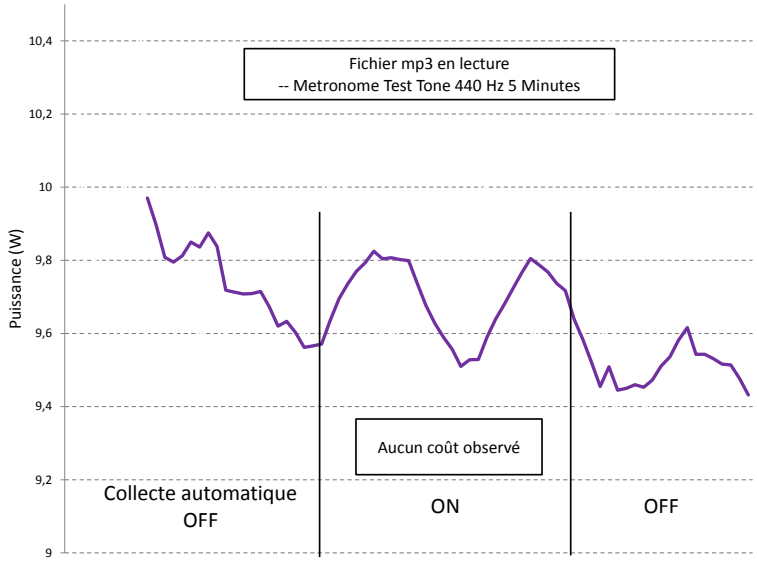

FIGURE 4.18 – Coût de la collecte des métriques du bruit ambiant

La figure 4.19 illustre le cout de la collecte des métriques utilisées pour la gestion de la luminosité en fonction de la position.

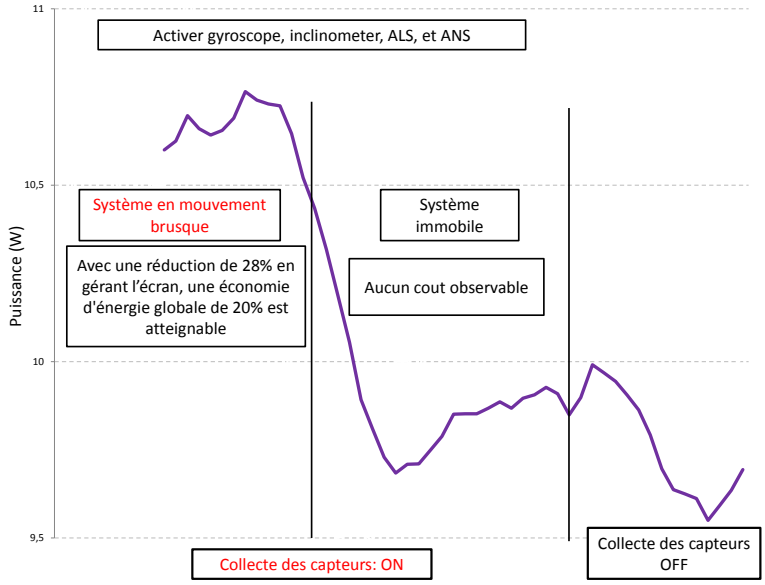

FIGURE 4.19 – Coût de la collecte des métriques pour la gestion de la luminosité

Pour la collecte des données des capteurs de mouvement et d'inclinaison, nous avons un surplus de puissance qui s'élève à 6% lorsque le système est en position anormale. Ce surplus est compensé par le gain engendré en configurant la luminosité à 20%. Cette réduction génère un gain de 28%. Menant ainsi à un gain total atteignant les 22%. Lorsque le système est immobile, nous n'observons aucun cout lié à la collecte.

4.4 Conclusion

Dans ce premier chapitre nous avons présenté la première contribution de cette thèse SBOC. Ce composant est est une sur-couche logicielle chargée au niveau de l'OS dans le but d'améliorer la consommation de puissance du système dans certains cas de figure. Comme la totalité de nos approches, SBOC obéit au modèle CPA présentée dans le chapitre 1. En comparaison avec les autres contributions, SBOC présente un certain nombre d'avantages comme un faible coût en consommation de ressources, un impact mémoire négligeable, une simplicité d'implémentation et un aspect dynamique et reconfigurable.

Les économies d'énergie obtenues avec ce premier composant subviennent dans certains scénarios liés à la position du système et son environnement. SBOC agit en trois étapes:

1. Collecte des données des capteurs.
2. Détection des scénarios offrant des opportunités de réduction de puissance.
3. Application d'heuristiques pour la configuration des niveaux de luminosité et du volume.

Dans cette première version, SBOC agit principalement sur la luminosité de l'écran et le volume des hauts parleurs mais le même *modus operandi* peut être exploité pour la gestion d'autres ressources hardwares. SBOC peut offrir des économies d'énergie avoisinant les 26% avec un surcout négligeable. Les économies d'énergie atteignables en gérant la luminosité et le volume sont respectivement de 22% et 4 %.

Pour finir, SBOC participe également à l'extension de l'Intel Energy Checker SDK (IECSDK) en lui ajoutant de nouvelles fonctionnalités par l'implémentation d'une IL et deux ALs qui correspondent respectivement au SCM, au SPM et au RMM.

SBOC est la solution la moins complexe de cette thèse axée uniquement sur l'environnement de l'utilisateur sans l'utilisation de techniques d'apprentissage. Dans le chapitre 5 suivant, nous présentons notre seconde contribution axée sur les applications.

Chapitre 5

Application Needs Based Optimization Component

Nous présentons dans ce chapitre la seconde contribution de cette thèse. Mené à terme, ce travail nous a permis de mettre en place le composant ANBOC pour Application Needs Based Optimization Component. Comme toutes les contributions de cette thèse, ANBOC est également une sur-couche logicielle qui tend à améliorer la consommation énergétique des systèmes mobiles en se basant sur le modèle CPA.

Contrairement à son prédécesseur SBOC, le composant présenté dans ce chapitre est axé sur les applications et les interactions de l'utilisateur. Avec SBOC, nous nous limitions à l'environnement de l'utilisateur, tandis que dans le composant actuel, nous prenons en compte les applications exécutées et leurs besoins en ressources hardwares.

ANBOC a été mis au point dans le but d'analyser l'interaction entre l'utilisateur et ses applications. Cette analyse nous sert par la suite à assimiler comment les différents composants hardwares sont exploités par les applications. Cette compréhension du comportement de l'interaction entre l'utilisateur et ses applications permet de cerner leurs besoins. Pour ce faire, nous exploitons des algorithmes d'intelligence artificielle et de fouille de données. Ces algorithmes nous servent dans un premier temps à classifier les applications en fonction de leurs besoins en ressources hardwares et à prédire les futures applications qui seront exécutées. La classification combinée à la prédiction permettent d'anticiper les futures actions de l'utilisateur, tout en lui offrant la configuration hardware nécessaire au fonctionnement optimal de sa plateforme mobile.

Cette configuration doit répondre à deux critères majeurs: une réduction de la consommation d'énergie et un impact minime sur la qualité de service, les performances du système et la satisfaction de l'utilisateur. Pour une proportion d'utilisateurs, ANBOC nous permet d'avoir un gain d'énergie considérable, avoisinant les 30%, en comparaison avec la gestion offerte par l'OS. Ce chapitre présente en détail les contributions apportées avec ANBOC. La section 5.1 présente le contexte et la motivation de ce travail. Nous illustrons dans la section 5.2 la phase de collecte de données réalisée par ANBOC. La section 5.3 explicite la classification des applications et la section 5.4 approfondit la prédiction des futures actions de l'utilisateur. Dans la section 5.6, nous présentons les résultats expérimentaux et nous concluons ce chapitre dans la section 5.7.

Sommaire

5.1 Contexte et Motivation **86**
5.2 Collecte de données **88**
5.2.1 Architecture du DCM 88

5.3 Classification offline des applications **93**
5.3.1 Classification binaire des applications en fonction des besoins en connectivité 93
5.3.2 Classification des applications selon les besoins en fréquence CPU 94
5.4 Prédiction des futures applications de l'utilisateur **100**
5.4.1 Prédiction des applications 101
5.5 Fonctionnement de l'OA **106**
5.5.1 Configuration des ressources hardwares 108
5.5.2 Satisfaction de l'utilisateur 111
5.6 Résultats expérimentaux **112**
5.6.1 Résultats de la prédiction et de la classification . . . 112
5.6.2 L'évaluation d'ANBOC 114
5.6.3 Coût d'ANBOC 116
5.7 Conclusion **117**

5.1 Contexte et Motivation

Comme mentionné dans l'introduction générale, à notre époque, la société connait un réel engouement pour les systèmes mobiles sous leurs différentes déclinaisons, avec la large panoplie d'applications qu'ils incluent. Malgré le fait que nous sommes nombreux à utiliser ces mêmes plateformes, dotées de fonctionnalités similaires et fournissant les mêmes applications, chaque utilisateur reste un cas particulier avec une utilisation et des attentes bien spécifiques.

Cette utilisation spécifique se traduit par différents paramètres comme le type et le nombre d'applications utilisées, les besoins en luminosité, les besoins de connectivité, les attentes en matière de puissance de calcul, etc. Ajoutons également à cette liste non exhaustive l'ultime problématique de cette thèse qui est la consommation énergétique. En effet, la consommation d'énergie diffère d'un utilisateur à un autre et varie également pour ce même utilisateur en fonction de ses habitudes et son comportement. Comme mentionné dans le chapitre 2, nombreux travaux académiques ont souligné l'importance d'analyser la consommation énergétique en considérant l'utilisateur final comme l'épicentre de l'étude [20, 23, 119, 120].

Une des motivations majeure de cette contribution demeure dans le fait que de nombreux utilisateurs ont tendance à utiliser leurs plateformes en suivant un enchainement logique dans les exécutions des applications. Cet enchaînement est appelé pattern d'utilisation. Au fur et à mesure qu'il manie son système mobile, l'utilisateur prend des habitudes d'utilisation en exécutant par exemple *Twitter* après avoir exécuté *Facebook* et cela se produit à des moments et dans des contextes bien précis. Ces enchaînements diffèrent d'un utilisateur à un autre et sont fortement corrélés au temps, le style de vie, la profession, etc.

Un autre point crucial est que les utilisateurs n'exploitent pas tous les niveaux de performances offerts par un composant hardware comme annoncé dans [120]. En d'autres termes, les performances d'un composant hardware donné ne sont pas toujours sollicitées à leur configuration maximale. Prenons l'exemple de la luminosité de l'écran : cette ressource est rarement utilisée à ses 100 niveaux disponibles.

D'autre part, le marché des applications mobiles est en évolution exponentielle, tant sur le plan quantitatif que qualitatif. Or, les développeurs des applications mobiles prennent rarement l'aspect de l'efficacité énergétique lors de l'étape de conception de l'application. Cette omission donne naissance à des applications énergivores qui n'arrangent en rien la durée de vie limitée des batteries.

Démarrant de ces postulats, nous avons jugé qu'il était primordial de proposer aux utilisateurs une gestion de la consommation énergétique qui soit personnalisable et adaptative à leurs besoins, leurs comportements et leurs applications. Pour ce faire, nous avons mis au point ANBOC. Notre composant tire profit du fait que les systèmes mobiles dans toutes leurs déclinaisons représentent une mine d'informations considérables. En effet, cet afflux de données en continue malgré qu'il soit coûteux d'un point de vue énergétique n'en demeure pas moins très indicatif et porteur d'informations sur l'utilisateur, ses préférences, ses applications et ses besoins en matière de configuration hardware. Exploitées à bon escient, ces informations nous permettent de proposer des politiques d'optimisations qui améliorent la gestion d'énergie proposée par l'OS.

ANBOC est basé sur le modèle CPA et incorpore trois phases principales qui sont illustrées dans la figure 5.1.

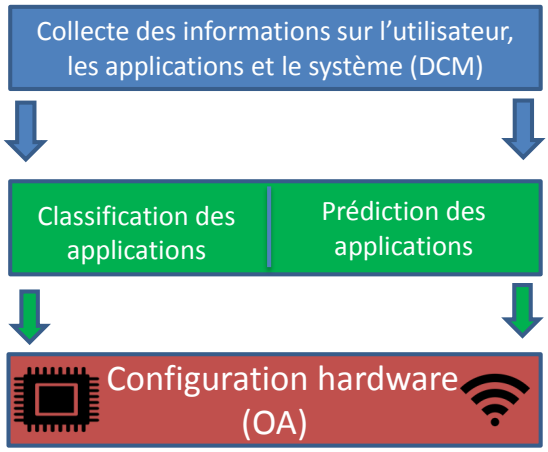

FIGURE 5.1 – Abstraction de l'architecture globale de ANBOC basée sur le modèle CPA

1. **Collecte d'informations**: la première étape consiste en la récupération des données relatives aux applications exécutées, les informations sur l'utilisation des ressources hardwares et les informations temporelles. Cette étape est assurée par le Data Collection Module (DCM). Ce module réalise un traitement sur les données collectées et les stocke. Les informations stockées sont ensuite consommées lors de la seconde étape.
2. **Traitement des données collectées**: les données fournies par le DCM sont utilisées de deux manières différentes:
   - Classification des applications: ce mécanisme nous permet de classifier les applications en fonction de leurs besoins en ressources hard-

wares, telles que la connectivité Wi-Fi et la puissance de calcul. Cette classification a pour but d'allouer moins de ressources matériels à l'application, en comparaison avec les ressources allouées par l'OS. Toutefois, cette classification doit impérativement préserver la qualité de service et la satisfaction de l'utilisateur. La classification d'une application A selon ses besoins en ressources $B_{ressources}$ aura pour but de proposer une configuration hardware consommant moins d'énergie que la configuration réalisée par l'OS. Cette configuration proposée sera appliquée à chaque fois que l'application sera lancée.

- Prédiction des futures applications exécutées: ce second mécanisme a pour but de prédire les futures applications qui seront lancées par l'utilisateur, sur la base des informations fournies par le DCM. Le rôle majeur de la prédiction dans ANBOC est de maintenir la satisfaction de l'utilisateur et anticiper ses besoins.

3. **Configuration hardware**: les mécanismes de classification et de prédiction de la phase en amont, sont exploités pour reconfigurer les composants matériels ciblés au moment opportun. Cette reconfiguration consiste à baisser les ressources hardwares dans le but d'offrir à l'utilisateur une réduction d'énergie. Dans le cas où aucune opportunité de réduction d'énergie n'est détectée, l'OS reprend la main sur la gestion des ressources. Cette étape est assurée par l'Optimizer Actuator (OA).

La section 5.2 suivante présente la phase de collecte.

5.2 Collecte de données

Toutes les contributions de cette thèse obéissent au schéma CPA, agissant en trois étapes communicantes présentées dans le chapitre 1. Dans cette section, nous développons l'étape de collecte de données dans ANBOC réalisée par le DCM. Les données collectées servent à la classification et à la prédiction. La classification combinée à la prédiction nous permettent de proposer des politiques de gestion de la consommation de puissance offrant des économies d'énergie, tout en prenant en compte la satisfaction de l'utilisateur.

Contrairement à SBOC, où nous nous étions contraints de gérer uniquement la luminosité de l'écran et le volume des hauts parleurs à cause des données des capteurs, ANBOC se place comme une solution générique, applicable à toutes les ressources hardwares du système. Toutefois, dans la version actuelle de notre composant, nous nous sommes intéressés à la gestion du processeur et de l'interface Wi-Fi. Nous avons choisi ces deux ressources pour illustrer la faisabilité des mécanismes de classification et de prédiction. Ajoutons à cela que le processeur représente une des ressources les plus énergivores du système mobile. Une bonne gestion du processeur peut réduire la consommation énergétique de 30%. L'interface Wi-Fi a été choisie quant à elle car c'est une ressource que l'utilisateur désactive rarement, même quand ce dernier n'est pas connecté. Cette omission peut coûter jusqu'à 8% de batterie.

5.2.1 Architecture du DCM

La collecte de données est réalisée pendant une période de temps ajustable. Dans un premier temps, cette période a été fixée à deux semaines. Nous pensons

que cette durée représente une période de temps suffisamment longue pour capter les habitudes de l'utilisateur, les informations sur l'usage du système et les applications, sans que cette étape ne soit fastidieuse. Ces deux semaines représentent un bon compromis dans le sens où le DCM collectera toutes les informations deux fois pour les jours de semaine et deux fois pour les weekends. Permettant ainsi d'éviter une classification approximative basée sur une seule semaine d'utilisation. Cela permet également de vérifier notre hypothèse concernant le changement de comportement de l'utilisateur entre les jours de semaines et les weekends.

Pour ce faire, nous avons distribué le DCM à des étudiants qui se sont portés volontaires. Le module de collecte est exécuté en *offline* à chaque démarrage du système. Le DCM est implémenté sous forme d'une *input library*. Cette IL est lancée en background et de façon transparente à l'utilisateur. En cas d'insatisfaction de l'utilisateur des politiques d'optimisations proposées, ce module est relancé pour collecter davantage d'informations et corriger les mécanismes de classification et de prédiction. Le DCM collecte deux catégories d'informations : la première est destinée à la classification des applications et la seconde est utilisée pour réaliser la prédiction.

5.2.1.1 Données collectées pour la classification

La figure 5.2 schématise le fonctionnement du DCM pour la classification des applications.

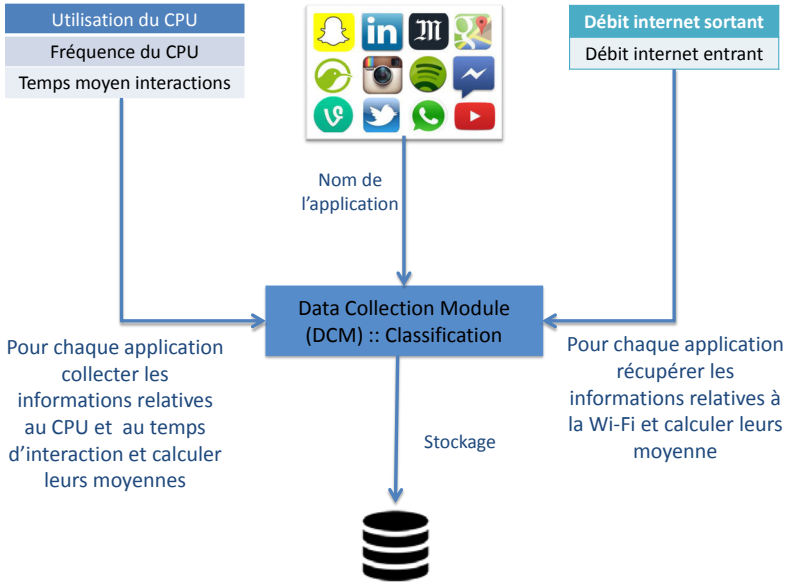

FIGURE 5.2 – Collecte de données pour la classification des applications

Pour un jour donné, le DCM collecte pour chaque application ciblée par la classification les informations suivantes:

1. Informations sur la Wi-Fi:
   - Débit entrant (ko/s): pendant l'exécution d'une application donnée en premier plan, le DCM s'interface avec l'API *WLAN*[1] exposée par l'OS pour collecter le débit de téléchargement entrant de la connexion Wi-Fi.
   - Débit sortant (ko/s): Le même procédé est réalisé pour le débit de téléchargement sortant.
2. Informations sur le CPU:
   - Charge d'utilisation du CPU (%): pendant l'exécution d'une application donnée, le DCM capture la charge de travail du CPU (%).
   - Fréquence CPU (MHz): la fréquence d'horloge allouée par l'OS est également collectée pendant l'exécution de l'application ciblée par la classification.
   - Temps entre deux interactions utilisateur/apps ($T_{interaction}$ (S)): durant l'exécution de l'application donnée, le DCM collecte également le temps entre deux interactions utilisateurs. Une interaction utilisateur peut être un appui tactile sur l'écran, un clic ou l'utilisation du clavier.

Concernant les informations susmentionnées, l'intervalle entre deux collectes est fixé arbitrairement à 40 secondes. Contrairement au module de collecte des capteurs du chapitre précédent, le DCM réalise un traitement sur les données collectées. En effet, pour la Wi-Fi, le DCM réalise pour chaque application:

- Le calcul de la moyenne des débits entrants.
- Le calcul de la moyenne du débit sortants.
- La somme des moyennes des deux débits.

Le DCM calcule également pour l'application ciblée par la classification CPU:

- La moyenne de la fréquence d'horloge allouée pendant l'exécution de l'application.
- La moyenne et l'écart type de la charge d'utilisation.
- Le ratio de la fréquence allouée pour chaque application, par rapport à la fréquence d'horloge maximale.

Le DCM calcule également le temps moyen entre deux interactions utilisateurs. Cette information sert à jauger la satisfaction de l'utilisateur suite aux politiques d'optimisations comme illustré dans la section 5.5.2.

5.2.1.2 Données colletées pour la prédiction

Pour la prédiction des futures applications, le DCM capture l'application lancée par l'utilisateur en premier plan, ainsi que les applications en background. Nous calculons également les différentes durées d'utilisation de chaque application. Ces durées sont calculées en fonction de l'ordre de l'application dans une séquence d'exécution donnée. Par exemple, le durée d'utilisation de l'application *Facebook*, quand elle est utilisée entre *Youtube* et *Instagram* est différente de la durée d'utilisation de *Facebook* entre *Skype* et *2048*. Toutes les données collectées sont corrélées au jour de semaine, la date et l'heure.

1. https://msdn.microsoft.com/fr-fr/library/windows/desktop/aa816369(v=vs.85).aspx

Pour la prédiction, chaque collecte représente un tuple dans la base de donnée sous la forme suivante :

- Jour de semaine;
- Date et heure;
- Application en premier plan;
- Temps moyen d'exécution de l'application en fonction de l'ordre de la séquence;
- Application en background;

Contrairement à la collecte de données pour la classification, où nous fixons un intervalle de 40 secondes entre chaque collecte, pour la prédiction, la collecte est déclenchée par le changement de l'application exécutée en premier plan.

La figure 5.3 illustre les fonctionnalités du DCM pour la prédiction des applications.

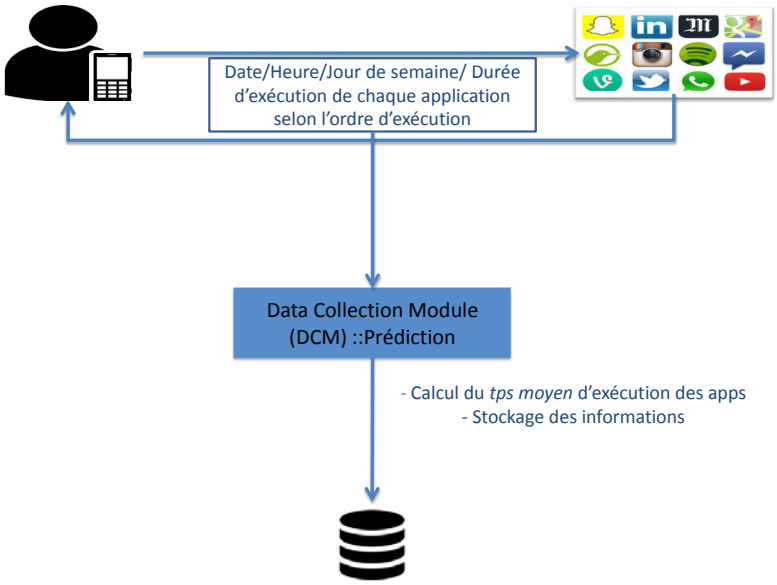

FIGURE 5.3 – Collecte de données pour la prédiction

Rappelons qu'un des éléments cruciaux de cette étude est l'aspect temporel. En effet, notre hypothèse est basée sur le fait que l'utilisateur a tendance à suivre le même schéma d'utilisation de ses applications en fonction du jour de semaine, la date et l'heure. Les enchaînements d'applications au cours d'un Lundi par exemple à 8 H du matin ne sont pas forcément les mêmes à 23 H. Les applications que nous aurions tendance à utiliser pendant les jours de travail ne seront pas les mêmes que celles utilisées pendant les weekends.

L'algorithme suivant présente le fonctionnement du DCM.

Algorithme 1 : Algorithme de fonctionnement du DCM

```
1  Données:
2  T,t, tempsInteraction : Doubles qui représentent la durée de temps en
   secondes;
3  Date: Structure pour stocker les informations relatives à la date;
4  WiFiDo, WiFiUp: Double pour stocker les informations du débit
   montant et descendant de la connexion sans fil;
5  CpuUt: Double pour stocker l'utilisation du CPU en pourcentage;
6  CpuFr: Double pour stocker la fréquence du CPU;
7  AppName, BackgroundProcess: Chaine de caractères pour stocker le
   nom des applications en foureground et background;
8  /* Récupération des données qui seront utilisées pour la classification
   et la prédiction */
9  /* Lancement de l'Input Library permettant la collecte des données */
10 Début
11 tant que Vrai faire
12     Pour chaque t seconde faire
13         Date = la date, jour de semaine et heure;
14         AppName = le nom de l'application en premier plan;
15         BackgroundProcess = Récupérer les processus en background;
16         CpuFr = la fréquence du CPU();
17         CpuUt = l'utilisation du CPU();
18         lancement du hook permettant de collecter les interactions de
           l'utilisateur;
19         WiFiDo = le débit entrant de la connexion sans fil (Download
           Rate);
20         WiFiUp = le débit sortant de la connexion sans fil (Upload Rate);
21         si la durée de collecte = 40 secondes alors
22             Calcul et sauvegarde de la moyenne et l'écart type des
               données relatives au CPU;
23             Calcul et sauvegarde du temps moyen entre deux interactions;
24             Calcul et sauvegarde la moyenne des débits entrant et sortant
               de la Wi-Fi;
25             Sauvegarde du nom de l'application en foureground et des
               applications en background;
26         si l'application en premier plan change alors
27             Initialisation des tableaux de collecte à 0;
28             Initialisation de la durée de collecte à 0 seconde;
```

Comme mentionné dans l'introduction, la classification des applications se fait en *offline*. Dans un premier temps, les applications seront classifiées en fonction des données collectées pendant les deux semaines, où le DCM était déployé. Les données collectées sont figées mais peuvent être enrichies dans le cas où l'utilisateur n'est pas satisfait de la gestion proposée. La section suivante présente la classification des applications.

5.3 Classification offline des applications

L'objectif de la classification est de détecter des opportunités de réduction de puissance lors de l'utilisation de chaque application. Cette classification dépend des ressources utilisées par l'application et dépend également du comportement de l'utilisateur. Le s résultats de la classification sont directement pris en considération par l'OA dans la phase d'optimisation. Pour illustrer la faisabilité d'ANBOC, les applications sont classifiées selon leurs besoins en connectivité Wi-Fi et en puissance de calcul (CPU). Néanmoins, ce procédé de classification demeure généraliste et peut être étendu à d'autres ressources hardwares.

Nous avons les deux types de classification suivants:

- **Classification binaire pour la Wi-Fi**: cette classification détermine si l'application requiert une connectivité Wi-Fi ou pas. L'objectif étant de pouvoir désactiver l'interface Wi-Fi, lorsqu'aucune application ne requiert de connectivité sans fil.
- **Classification supervisée pour le CPU**: cette classification détermine un seuil maximal de fréquence CPU à allouer pour l'application ciblée. Cette borne supérieure de fréquence permet de répondre juste aux besoins de l'utilisateur et de l'application. Le but est de réduire la fréquence d'horloge allouée par l'OS.

Nous commençons par présenter la classification Wi-Fi.

5.3.1 Classification binaire des applications en fonction des besoins en connectivité

Le but de cette classification est de contribuer à la gestion intelligente de l'interface Wi-Fi selon les besoins des applications exécutées par l'utilisateur. Pour achever cette classification *offline*, nous avons réalisé des expériences préliminaires et des mesures concernant les débits de téléchargement entrants et sortants. Le procédé est le suivant:

1. Durant le test, nous analysons chaque application à part. Nous nous assurons qu'aucun processus ou application en background est en train d'utiliser la connexion sans fil. Nous nous assurons également qu'aucune mise à jour nécessitant une connexion internet n'est lancée. Ces vérifications assurent la précision des résultats de la classification.
2. Pour chaque application notée A, nous estimons le débit internet qu'elle requiert noté IR. Ce débit représente la somme du débit de téléchargement entrant *Upload Rate (UR)* et sortant *Download Rate (DR)* pendant l'exécution de l'application fournie par le DCM.
3. Le système d'exploitation *Windows* utilisé fournit une gestion de connectivité qui envoie des trames de tests dans le but de s'assurer de la pré-

sence d'une connexion à un réseau. Même dans le cas où aucune application exécutée ne requiert une connexion internet, les tests sont réalisés en permanence par l'OS. La taille de ces trames de test est estimée aux alentours de 9.70 ko, ce qui nous pousse à fixer le seuil de test de connectivité *Windows* à 10 ko.

4. Pour finir, nous réalisons une comparaison entre le débit Internet *IR* de chaque application et le seuil de connectivité *Windows* estimé à 10 ko. Chaque application dont le débit internet est supérieur au seuil, est classifiée comme une application nécessitant la connectivité sans fil et vice versa.
5. Si $UR + DR <= 10$ ko alors l'application est classifiée comme non-nécessitant une connectivité Wi-Fi. La configuration dynamique de l'interface Wi-Fi est effectuée sur la base de cette classification.

Évidemment, le point crucial est d'évaluer l'exigence complète de l'état du système. Cela a pour objectif d'éviter l'insatisfaction de l'utilisateur lorsque la connexion sans fil est désactivée alors que celle-ci est requise par l'utilisateur.

Le tableau 5.1 donne un exemple de classification pour trois applications.

Tableau 5.1 – Exemple de la classification Wi-Fi des applications

Apps	IR (kO/s)	DR (kO/s)	UR (kO/s)	Classe Wi-Fi
Messeneger	33.55	12.93	19.62	ON
Youtube	366.09	325.81	40.28	ON
2048	1.56	1.02	0.54	OFF

La partie suivante présente la classification pour les besoins en calcul.

5.3.2 Classification des applications selon les besoins en fréquence CPU

5.3.2.1 Ajustement de la fréquence et du voltage du processeur (DVFS)

Dans cette contribution, nous nous intéressons à l'ajustement de la fréquence du processeur pour tenter de réduire la consommation d'énergie de ce composant. Dans ce qui suit, nous présentons l'ajustement dynamique du voltage et de la fréquence d'horloge du processeur. Comme mentionné précédemment, le processeur est un des composants le plus énergivore dans un système mobile. La consommation de puissance du CPU est subdivisée en trois parties [67], la consommation dynamique P_{dyn}, la consommation des courts circuits P_{cc} et la consommation de puissance due aux fuites des différents transistors P_{leak}. Cette puissance est définie par l'équation c-dessous:

$$P_{processeur} = P_{dyn} + P_{cc} + P_{leak}$$

La consommation dynamique représente la partie la plus importante [67], elle est caractérisée par sa dépendance aux activités du processeur.

Elle peut être exprimée en fonction de la capacité électrique, sa fréquence f et son voltage v.

$$P_{dyn} = c * f * v^2$$

Compte tenu de la constance de P_{cc} et P_{leak}, l'optimisation de la consommation du processeur consiste principalement à réduire sa consommation dynamique. Plusieurs approches ont été proposées pour la réduction de la consommation dynamique principalement basées sur la réduction du voltage et de la fréquence. Une des approches les plus courante et utilisée est le DVFS. Cette solution réalise l'ajustement de la fréquence et le voltage du processeur en fonction de la charge d'utilisation de ce dernier. [24].

Afin de supporter le DVFS, les processeurs disposent d'un ensemble de fréquences et de voltages. Dans cet ensemble, à chaque fréquence correspond un voltage, le couple fréquence/voltage est appelé performance state (p-state). Chaque p-state correspond à une quantité de puissance consommée. Le rôle du DVFS est de basculer entre les différents p-states supportés par le processeur afin d'en optimiser la consommation de puissance. Plusieurs algorithmes et stratégies DVFS sont disponibles, chaque OS en implémentant sa propre version. Un algorithme basé sur le DVFS est composé de deux phases, une phase d'augmentation de la fréquence et une phase de diminution.

La figure 5.4 illustre le fonctionnement du DVFS.

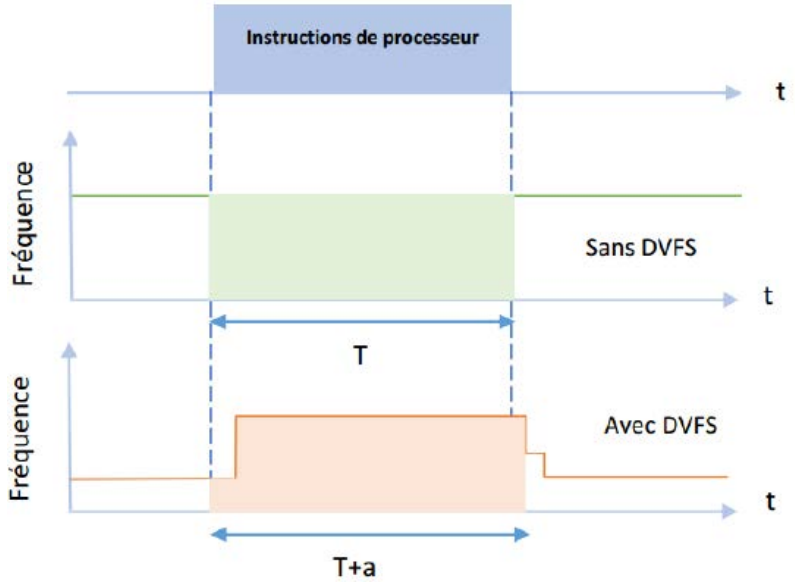

FIGURE 5.4 – Exemple de fonctionnement du DVFS

Dans cet exemple, nous remarquons que sans l'ajustement dynamique, la fréquence est fixe. Tandis qu'avec le DVFS, la fréquence varie selon la charge de travail. Cette fréquence et la charge de travail du CPU sont généralement proportionnelles. Dès qu'un ensemble d'instructions est détecté, la fréquence augmente, puis elle revient à sa valeur minimale à la fin du traitement de ces instructions. Le DVFS augmente le temps d'exécution d'une constante a, sans compromettre les performances du système [72].

5.3.2.2 Présentation des classes

Les systèmes d'exploitation utilisent les paramètres du processeur pour adapter la fréquence de ce dernier à la volée [118]. Ces paramètres sont principalement:

- Le nombre d'instructions par seconde.
- Les fréquences supportées par le processeur.
- La charge d'utilisation du processeur.

Cependant, l'utilisateur final est souvent négligé. Cela est paradoxal sachant que le but final de tout système interactif est de satisfaire cet utilisateur final. Dans [34], les auteurs affirment que l'utilisateur peut être satisfait avec une fréquence moins élevée que celle proposée par l'OS sans que cela n'altère sa satisfaction. Partant de ce postulat, nous proposons une classification supervisée des applications en fonction de leurs besoins en CPU. L'objectif de cette classification est principalement de définir des règles permettant de classer des objets dans des classes à partir de variables qualitatives ou quantitative, caractérisant ces objets. Par conséquent, notre classification CPU nous sert à classifier des applications dans des classes prédéfinies à partir des données collectées par le DCM.

Afin de choisir les seuils de fréquences de chaque classe, nous recourons à un modèle de consommation. Ce modèle permet de mesurer la puissance consommée par le processeur durant son fonctionnement. Nous utilisons le modèle proposé par Alex Shye et al [119, 120]. Dans ce modèle la consommation du processeur $P_{processeur}$ est liée à son utilisation $Util_{processeur}$ et sa fréquence. L'équation suivante exprime ce modèle, avec $\beta_{processeur}$ comme constante de consommation de chaque fréquence pour une charge d'utilisation CPU de 1%. θ constante, commune à toutes les fréquences, estimée à 826 mW.

$P_{processeur} = \beta_{processeur} * Util_{processeur} + \theta$, où $\beta_{processeur}$ est obtenue par expérimentation et mesure. $Util_{processeur}$ est collectée via des compteurs de performances accessibles via l'API Performance Data Header (PDH).

Le tableau 5.2 présente les différentes fréquences allouées par le processeur de l'*Ultrabook*. Pour chaque fréquence, nous donnons le voltage correspondant ainsi que la constante $\beta_{processeur}$. Ces différentes valeurs ont été obtenues avec le Yokogawa WT 210 [61] et l'ESRV.

Tableau 5.2 – Les différentes P-states de l'*Ultrabook*

P-state	Freq (MHz)	Volt (V)	$\beta_{processeur}$
P0	709	1.65	50
P1	800	1.89	59
P2	1000	2.31	65
P3	1250	2.4	71
P4	1400	2.5	80
P5	1500	2.59	92
P6	1600	2.66	104
P7	1750	2.74	113
P8	1900	2.93	122
P9	2000	3.05	131
P10	2200	3.2	149
P11	2300	3.33	158
P12	2400	3.41	166
P13	2500	3.56	175

Parmi les différentes fréquences supportées par notre processeur, nous sélectionnons quatre fréquences dont chacune représente une classe. Le choix de ces quatre classes s'est fait de manière à avoir quatre niveaux de fréquences différents, allant d'un faible besoin en puissance de calcul à un besoin élevé. Le critère de ce choix repose sur le fait d'avoir quatre constantes β qui sont le plus éloignées possible. Cela garantit d'avoir un gain énergétique notable selon le choix des classes. Ces quatre classes sont:

1. **Classe** C_1 **- Faible**: représente les applications qui requièrent une faible fréquence CPU ($<$ 800 MHZ). Les applications comme *Word* ou *Excel* peuvent en faire partie. Nous avons également dans cette catégorie des jeux peu complexes comme *Imperial Sudoku*.
2. **Classe** C_2 **- Moyenne**: les applications qui ont besoin d'une puissance de calcul moyenne. La fréquence varie entre 800 MHz et 1.25 GHz. Les navigateurs WEB comme *Google Chrome* peuvent éventuellement appartenir à cette classe.
3. **Classe** C_3 **- Élevé**: cette classe contient les applications qui nécessitent une puissance de calcul élevée variable entre 1.25 GHz et 1.75 GHz. Nous pouvons y trouver des jeux plus complexes que ceux de la première catégorie comme *2048*.
4. **Classe** C_4 **- Élevé**: cette classe correspond aux applications qui ont un besoin de fréquence CPU très élevée. Les applications de traitement d'images peuvent appartenir à cette classe. Notons que durant nos expérimentations, aucune application n'a été affectée à cette classe.

Nous précisons que la fréquence de chaque classe détermine une borne supérieure et non pas une fréquence figée. Nous explicitons ce point dans la sec-

tion 5.5. Dans le cas où l'application exécutée n'a pas été classifiée au préalable, la gestion de la fréquence revient entièrement à l'OS. Après avoir défini nos 4 classes, nous procédons à la classification avec les paramètres collectés par le DCM.

5.3.2.3 Algorithme de classification

Rappelons qu'avec ANBOC, pour classifier une application, la charge d'utilisation du CPU et sa fréquence sont mesurées pendant l'exécution de l'application ciblée. Nous utilisons pour caractériser chaque application, la moyenne (m), l'écart type (e) et le ratio de la fréquence allouée pour l'application par rapport à la fréquence maximale (f). Nous déterminons ensuite la probabilité de chaque application d'appartenir à une des classes c_i, en fonction des paramètres (m), (e) et (f) qui ont été calculés.

Pour réaliser cette classification, il existe dans la littérature de nombreuses techniques. Parmi les solutions existantes, nous citons: les arbres de décision [127], la régression logistique [87], la régression linéaire [71], la classification bayésienne [138], machines à vecteur de support (SVM) [111], méthodes des K plus proches voisins [30] et les réseaux de neurones artificiels [114]. Pour divers raisons en relation avec la nature de nos données en entrée, nous avons opté pour un modèle probabiliste, en utilisant la classification bayésienne naïve basée sur le théorème de Bayes. Un classifieur bayésien naïf est un algorithme supervisé qui permet de classifier un ensemble d'observations selon des règles déterminées par l'algorithme lui même. Le principe de la classification bayésienne est d'inférer des quantités décrites par des probabilités. Ces probabilités seront utilisées pour encadrer l'inférence. À chaque hypothèse, nous associons une probabilité. L'observation d'une ou plusieurs instances peut modifier cette probabilité et par conséquent, modifier le résultat de la classification.

Nous avons dans le survey [70] une comparaison entre les différents algorithmes de classification supervisée. Parmi les avantages d'une classification bayésienne, nous avons :

- Une économie en puissance de traitement.
- En comparaison avec d'autres techniques de classification, les classifieurs bayésiens naïfs sont plus précis avec des échantillons de taille réduite sur lesquels ils convergent plus rapidement, pour estimer les paramètres nécessaires à la classification. Ce qui nous permet de classifier nos applications avec de faibles quantités de données en entrée.
- L'avantage du classifieur bayésien naïf est qu'il requiert relativement peu de données d'apprentissage pour estimer les paramètres nécessaires à la classification, à savoir moyennes et variances des différentes variables. En effet, l'hypothèse d'indépendance des variables permet de se contenter de la variance de chacune d'entre elles, pour chaque classe, sans avoir à calculer de matrice de covariance.
- Une convergence plus rapide qu'un modèle de discrimination comme la régression logistique, ceci implique une consommation de puissance très faible. Ce cout est présenté dans la section 5.6.

Pour résumer, une classification bayésienne est une approche probabiliste basée sur les probabilités conditionnelles qui supposent l'indépendance des attributs. Dans notre cas, nous estimons la probabilité qu'une application A appartienne à une classe c_i sachant qu'une hypothèse préliminaire est vérifiée.

Cette hypothèse préliminaire peut être l'appartenance d'une application à la classe CPU faible si la moyenne de la fréquence collectée est inférieure à 800 MHz.

Notre classificateur opère comme suit:

$$P(c_i|m,e,f) = max_{1\leq j\leq 4} P(c_j|m,e,f)$$

Avec

$$P(c_i|m,e,f)$$

la probabilité pour qu'une application soit classifiée dans la classe i étant donné m, e et f

$$P(c_i|m,e,f) = \frac{P(c_i) * P(m,e,f|c_i)}{P(m,e,f)}$$

Où

$$P(m,e,f|c_i) = P(m|c_i) * P(e|c_i) * P(f|c_i)$$

et

$$P(m,e,f) = \sum_{i=1}^{4} P(m|c_i) * P(e|c_i) * P(f|c_i)$$

Pour calculer les différentes probabilités simples, nous considérons les distributions m, e et f comme des distributions gaussiennes [66]. Nous choisissons des distributions gaussiennes car nous n'avons pas réalisé d'apprentissage, nous nous contentons d'utiliser des probabilités marginales.

Par exemple avec μ la moyenne définie par

$$\mu = \frac{1}{N}\sum_{1}^{n} x_i$$

et σ^2 la variance du ratio définie par:

$$\sigma^2 = \frac{1}{(N-1)}\sum_{i}^{N}(x_i - \mu)^2$$

$$P(f|c_i) = \frac{1}{\sqrt{2 * \pi * \sigma^2}} * \exp\left(-\frac{(f-\mu)^2}{2 * \sigma^2}\right)$$

Sur la base du résultat de cette classification bayésienne, nous affectons nos applications à nos quatre classes. Le tableau 5.3 nous montre les résultats de la classification concernant un seul utilisateur pour trois applications. Selon la classe de l'application, l'OA sélectionne le seuil maximal de la fréquence du processeur comme cela va être présenté dans les sections suivantes.

Dans le cas où une application non classifiée est exécutée, la gestion de la fréquence du CPU revient au système d'exploitation. Pendant l'exécution de cette dernière, les informations nécessaires à sa classification sont collectées et stockées pour aboutir à une classification offline. La classification réalisée par ANBOC est dite statique car elle repose sur des données collectées et stockées

par le DCM. Cette classification est réalisée à chaque fois que le DCM collecte de nouvelles données. Pour ne pas impacter la consommation d'énergie du système, la classification est réalisée lorsque le système est en charge. Cependant, cette classification peut être réalisée de façon dynamique en fonction de la satisfaction de l'utilisateur et du mécanisme de prédiction que nous présentons dans la section suivante.

Tableau 5.3 – Exemple de classification CPU pour un utilisateur labmda

Apps	Utiliz CPU Moy	Ecart Type	Ratio Freq	Classe
Chrome	25.88	7.65	0.71	Moyenne
Foxit	7.45	4.07	0.55	Faible
2048	46	10.58	0.87	Élevée

5.4 Prédiction des futures applications de l'utilisateur

Comme mentionné dans les sections précédentes, l'idée principale d'ANBOC est de proposer un système de gestion d'énergie qui soit personnalisable selon les besoins de chaque utilisateur et des applications exécutées. Les politiques d'optimisations proposées doivent apporter des améliorations au gestionnaire d'énergie standard du système d'exploitation. ANBOC dépend de facteurs comme les habitudes de l'utilisateur, ses préférences et ses interactions avec les applications. Le moment où ces éléments se produisent est également pris en compte. Nous supposons que l'utilisateur se comporte différemment vis à vis de son mobile pendant les jours de semaines et les weekends. Cette différence de comportement se fait également ressentir entre différentes période de la journée. Par exemple: les applications lancées pendant un Lundi matin dans un lieu de travail sont différentes des applications lancées pendant un Samedi soir. Les comportements de l'utilisateur sont également différents d'un utilisateur à un autre. Dans notre approche, nous considérons par exemple qu'un utilisateur installé dans un bureau a des besoins différents d'un utilisateur se déplaçant dans un chantier.

Le DCM collecte chaque fenêtre d'application en premier plan lancée par l'utilisateur pour capturer le nom de l'application via l'API *GetFourgroundWindow()*.

Il collecte également les applications s'exécutant en arrière plan, avec le jour de la semaine, la date et l'heure. À chaque fois que l'utilisateur lance une nouvelle application, le temps passé dans l'application précédente est calculé et enregistré. Cette information servira à calculer le temps moyen passé sur chaque application en le corrélant avec le jour de semaine, la période de la journée et l'ordre de l'application dans la séquence d'exécutions. Ces informations serviront ensuite à élaborer la prédiction des futures applications.

Dans la littérature, aucune solution de réduction d'énergie basée sur la prédiction des futures applications n'a été proposée. En prédisant les futures applications qui seront exécutées, nous serons en mesure de cerner le comportement

de l'utilisateur et ses différents patterns d'utilisation. Ceci permet également d'anticiper les futures demandes en ressources du couple utilisateur/application. Un autre intérêt de la prédiction est de pouvoir allouer à la future application les ressources nécessaires à son fonctionnement avant son début présumé, sans que l'utilisateur s'en aperçoive.

La classification des applications présentée précédemment est réalisée statiquement. Les résultats fournis sont stockés et utilisés pour gérer les ressources lors de l'exécution d'une application donnée. Comme mentionné précédemment, cette classification statique permet d'éviter de classifier l'application à chaque fois qu'elle est exécutée. Par conséquent, nous avons mis en place le mécanisme de prédiction pour pouvoir utiliser les résultats de la classification sans la réitérer. Le principal atout de la prédiction dans cette contribution est d'éviter tout impact négatif sur la satisfaction de l'utilisateur. La prédiction permet également d'ajuster les ressources graduellement avant le début présumé de l'application prédite. Cela sert à éviter tout ajustement brusque, impactant là aussi la satisfaction de l'utilisateur. Un exemple sur ce cas est donnée dans la section 5.5.

La prédiction peut également servir dans le cas d'une séquence d'applications composée d'une application A suivie d'une application B, l'application A nécessite une connectivité Wi-Fi durant toute son exécution, tandis que l'application B en a besoin uniquement pendant le début de son exécution. En prédisant l'application B en fonction de l'application A, nous pouvons précharger les données de l'application B avant son début et pendant l'exécution de l'application A. Cela permettrait de désactiver la Wi-Fi pendant l'exécution de l'application B et d'en générer une économie d'énergie.

La prédiction réalisée dans ANBOC est également utilisée pour prédire la durée de vie restante de la batterie dans le chapitre 7.

5.4.1 Prédiction des applications

Dans le but de prédire les futures applications qui seront lancées par l'utilisateur, nous supposons qu'il existe une dépendance dans l'enchainement des différentes applications pour un jour et une heure donnés. L'objectif est de découvrir quand l'utilisateur a pour habitude de lancer certaines de ses applications et dans quel ordre. Les techniques de Sequential Pattern Mining (SPM) [86] sont dédiées à extraire des éléments de séquences fréquentes parmi de larges données corrélées temporellement. En d'autres termes, le sequence mining détecte des motifs dans les flux de données dont les valeurs sont délivrées par séquences.

5.4.1.1 Sequence Mining

Notre motivation en utilisant ces techniques est de trouver des régularités dans les schémas d'utilisation du système mobile en se basant sur le jour de semaine, l'heure et les applications en background. Plus précisément, les éléments traités par ces techniques sont les applications en *foreground* lancées pendant la même tranche horaire d'un jour donné, pendant plusieurs semaines. Ces tranches horaires ont une durée de deux heures, en commençant à minuit. Par exemple, ces techniques peuvent détecter si les mêmes applications sont

fréquemment lancées dans le même ordre chaque Lundi entre 8 H et 10 H et nous fournissent également les séquences d'applications fréquentes.

Pour illustrer ces techniques, le formalisme suivant est utilisé:

Soit $I = \{i_1, i_2, ...i_m\}$ un ensemble de m éléments distincts. Un évènement est une collection non vide et non ordonnée de m éléments notée $(i_1, i_2, ...i_m)$. Une séquence α est une liste ordonnée d'évènements α_i notée $(\alpha_1 \longrightarrow \alpha_2 \longrightarrow ... \longrightarrow \alpha_q)$. Une séquence avec K éléments $(k = \sum_j \mid \alpha_j|)$ est appelée une K-séquence. Enfin, α (de cardinal n) est une sous séquence de β ssi $\alpha \subseteq \beta$, ie. il existe des entiers $i_1 < i_2 < ...i_n$ tels que $\alpha_1 \subseteq \beta_{i1}, \alpha_2 \subseteq \beta_{i2}, ...\alpha_n \subseteq \beta_{in}$. Une sous-séquences c de s est dite contiguë si elle résulte d'une suppression à la fois du premier et du dernier élément de s.

Il existe plusieurs algorithmes de sequence mining dans la littérature. Parmi les plus connues, nous avons:

- Generalized Sequenced Pattern (GSP) [124] est basé sur l'algorithme *apriori* [2] dont le but est de découvrir des règles d'associations séquentielles. GSP améliore son ascendant en définissant une fenêtre d'évènements ainsi qu'un intervalle.
- Sequential PAttern Discovery using Equivalence classes (SPADE) [143] est un algorithme basé sur la théorie des Treillis afin de limiter son espace de recherche. Pour chaque ensemble, SPADE utilise une borne supérieure et une borne inférieure. Pour rechercher les séquences fréquentes, cet algorithme utilise les stratégies Beadth-First Search (BFS) [19] et Depth First Search (DFS) [128].

Les résultats fournis par GSP et SPADE sont similaires. Pour les mêmes informations en entrée, les deux algorithmes présentent en sortie les mêmes séquences. La différence réside dans le temps de calcul de SPADE qui est meilleur que celui de GSP, lors de l'utilisation de bases de données de taille importante. Ceci s'explique par le fait que SPADE utilise des listes verticales temporaires pour les jointures. Malgré que SPADE peut être plus performant que GSP d'un point de vue temporel, il nécessite cependant, un formatage complet de la base de données en entrée. Ce travail en amont nous a dissuadé d'utiliser cet algorithme et a fait porter notre choix vers GSP. En effet GSP prend en entrée directement les informations collectée par le DCM sans la nécessité de modifier le format de la base de données. De plus, la différence des temps d'exécution pour GSP et SPADE n'est pas observable compte tenu de la taille de notre base de données, qui est d'environ 14 Mo pour chaque utilisateur. Nous pourrons envisager dans le futur d'améliorer GSP ou recourir à SPADE si la taille de la base de données augmentait considérablement. Pour finir, un dernier facteur qui a orienté notre choix vers GSP réside dans la simplicité de l'implémentation de cet algorithme.

5.4.1.2 L'algorithme GSP

GSP est un algorithme multipasse. La première passe détermine le support qui est le nombre de séquences contenant l'application recherchée. GSP permet de travailler uniquement sur les éléments contenant le support minimum et générer les séquences d'applications candidates. Le support de ces candidats étant déterminé lors des passages suivants.

La génération des candidats se fait en deux phases: la jointure et l'élagage. La jointure se fait en joignant l'ensemble des $(K-1)$ - séquences fréquentes L_{k-1}

avec lui même, ce qui consiste - en considérant les séquences $s_1, s_2 \in L_{k-1}$ - à rajouter le dernier élément de s_2 à s_1. Si l'élément constituait un évènement à lui tout seul dans s_2, alors il sera considéré comme un évènement dans s_1. De manière similaire, s'il faisait parti d'un évènement dans s_2, alors il sera rajouté au dernier évènement de s_1. La jointure est valable uniquement si en enlevant le premier élément de s_1, on obtient la même sous-séquence qu'en retirant le dernier élément de s_2.

Enfin, l'élagage consiste à retirer les candidats ayant des sous-séquences ou des sous-séquences contiguës dont le support serait inférieur au *minSupp*. Une fois l'ensemble des K-séquences candidates généré, chaque candidat est dénombré dans la base de données. Afin d'optimiser le dénombrement, une table de hachage est utilisée ainsi que la mise en place d'une durée maximale entre éléments d'une séquence, ce qui permet de limiter les recherches.

5.4.1.3 Application de GSP pour la prédiction

Une séquence d'applications est considérée fréquente quand son occurrence dans la base de donnée dépasse un seuil spécifié. GSP commence par collecter les applications dont la fréquence est supérieure à un seuil de support minimal, pour créer un ensemble de séquences fréquentes de longueur 1. Ensuite, GSP analyse de manière itérative la base de données des applications pour collecter le nombre de supports, afin de sélectionner les séquences d'applications fréquentes de longueur $K + 1$ parmi les séquences fréquentes de longueur K. Ce processus de sélection est répété jusqu'à ce qu'aucune séquence fréquente ou aucune séquence candidate ne soient détectées.

A la fin du traitement de GSP, nous avons en sortie les occurrences des applications allant d'une séquence formée par une application jusqu'aux séquences formées de K applications.

Le tableau 5.4 illustre un exemple des données en entrée pour le traitement de GSP.

Tableau 5.4 – Exemple de données en entrée pour GSP

Lundi [8H - 10H]	Séquences d'applications
Semaine 1	Excel, Mozilla, Spotify, 2048, VLC
Semaine 2	Excel, Mozilla, Word, Calc, VLC
Semaine 3	Notepad, Mozilla, Word, 2048, VLC
Semaine 4	Mozilla, Word, Spotify, 2048, VLC

Ces données présentent les séquences des applications lancées pendant 4 Lundis consécutifs entre 8 H et 10 H. Les données en sortie obtenues après l'application de GSP représentent le nombre des occurrences des applications entre 8 H et 10 H avec différentes longueurs K, comme l'illustre la figure 5.5.

K représente la longueur de la séquence, plus précisément le nombre d'applications que comporte la séquence. R représente la fréquence de répétition, qui indique le nombre de fois ou cette séquence d'applications se produit.

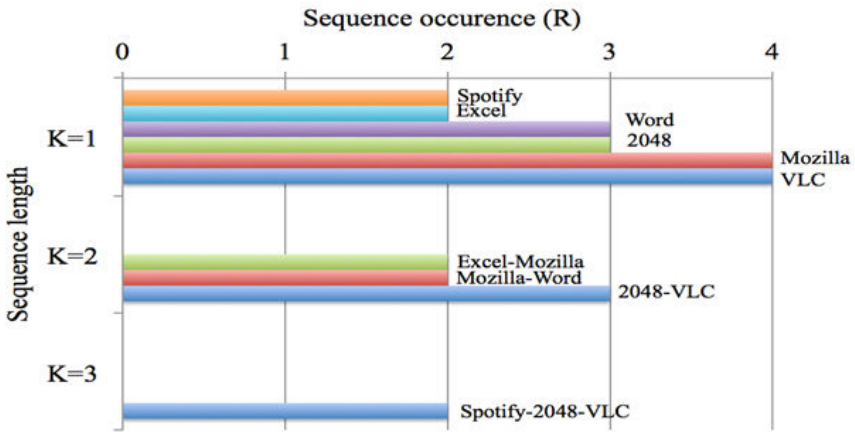

FIGURE 5.5 – Les sorties de GSP

L'analyse des données en sortie permet de déduire:

1. Avec $K = 1$, l'occurrence R nous donne le nombre de fois ou chaque application apparait. Nous notons que *Mozilla* et *VLC* sont les applications les plus fréquentes.
2. Avec $K = 2$, l'occurrence R concerne les séquences composées de deux applications comme la séquence (*2048, VLC*) qui apparait 3 fois.
3. Avec $K = 3$, l'occurrence R nous donne les séquences composées de 3 applications : nous notons que la séquence (*Spotify, 2048, VLC*) apparait deux fois et les autres séquences apparaissent une seule fois.

En se basant sur la sortie de notre algorithme, l'application en cours d'exécution, la date et l'heure collectées en temps réel, les applications susceptibles d'être lancées le 5 ème Lundi peuvent être prédites. Différentes prédiction peuvent être établies selon K:

- $K = 1$: l'application la plus utilisée est prédite sans prendre en considération l'application qui est en cours d'exécution.
- $K = 2$: nous prédisons l'application qui suivra l'application en cours d'exécution. Si l'application en cours est *2048*, l'application suivante sera *VLC*.
- Les résultats de l'analyse pour $K = 3$ peuvent être utilisés dans deux situations:
  — Pour prédire les deux prochaines applications connaissant l'application courante.
  — Pour prédire la prochaine application connaissant l'application courante et l'application précédente.

Dans cette contribution, nous choisissons d'utiliser la prédiction basée uniquement sur l'application en cours d'exécution. Dans ce cas de figure, le choix du facteur K le moins élevé permet d'avoir des prédictions précises. Toutefois, cela augmente la fréquence de consultation des sorties de GSP, ce qui cause une surplus en temps d'exécution et en consommation de puissance. Même si ce surplus est négligeable, un compromis doit être trouvé entre une grande valeur de K, permettant de réduire le surcoût et une petite valeur de K ($K = 2$

par exemple) qui permet d'avoir une meilleure précision sur la prédiction des applications.

Dans le but de déterminer le facteur K le plus approprié, nous avons mesuré le coût de la consultation des règles de GSP. Prenons l'exemple de la séquence suivante: *Mozilla: 20 min, Word: 17 min, VLC: 47 min, 2048: 7min*. Avec *Mozilla* comme application en cours d'exécution, les règles GSP prédisent *Word*, *Word et VLC*, ou *Word, VLC et 2048* en fonction de la valeur du facteur K. Le coût de la consultation a été mesuré par notre wattmètre et s'élève à 2 W/S.

- En choisissant K = 2, nous serons obligés de consulter les sorties de GSP à chaque changement d'application. Nous aurons à réaliser trois consultations, le surplus en terme d'énergie est d'environ 6 W/S.
- En choisissant K = 4, le surplus d'énergie est de 2 W/S car nous consulterons les règles de GSP une fois seulement pour prédire les trois applications suivantes.

Cette différence relativement faible nous pousse à choisir le facteur $K = 2$ pour sa précision.

Nous pouvons être dans le cas de figure où nous prédisons deux applications différentes qui ont le même nombre d'occurrences R. Nous expliquerons dans la partie suivante comment interagir avec ce cas de figure. La précision de la prédiction est relative à la quantité de données collectées. C'est pour cela que la précision de notre algorithme ne fera que croitre avec le temps. Le second paramètre qui impactera la prédiction est le comportement de l'utilisateur et sa régularité. En effet, quand un utilisateur exécute pendant un grand nombre de fois les applications dans le même ordre, la précision de la prédiction est plus élevée et vice versa. Nous pouvons tout de même avoir des cas où il n'est pas possible de prédire les futures applications. Cela arrive quand l'utilisateur ne suit aucun motif régulier dans l'exécution de ses applications.

La figure 5.6 synthétise comment la prédiction se fait de la part de l'OA.

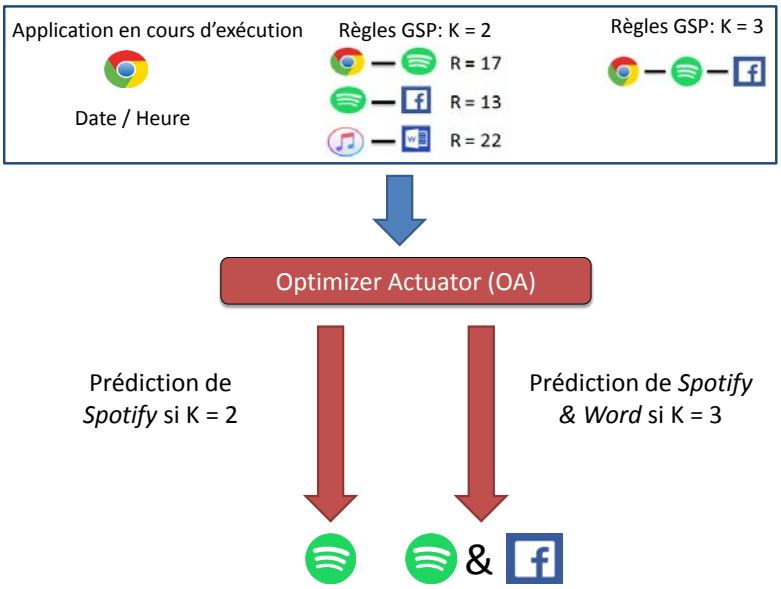

FIGURE 5.6 – Prédiction des applications par l'OA

5.5 Fonctionnement de l'OA

La dernière phase de notre composant ANBOC est d'adapter la consommation d'énergie du système mobile en se basant sur les informations collectées en temps réel, la classification et la prédiction. Cette phase est assurée par l'Optimizer Actuator (OA). L'OA a été implémenté sous forme d'une AL. La figure 5.7 résume le fonctionnement de l'OA.

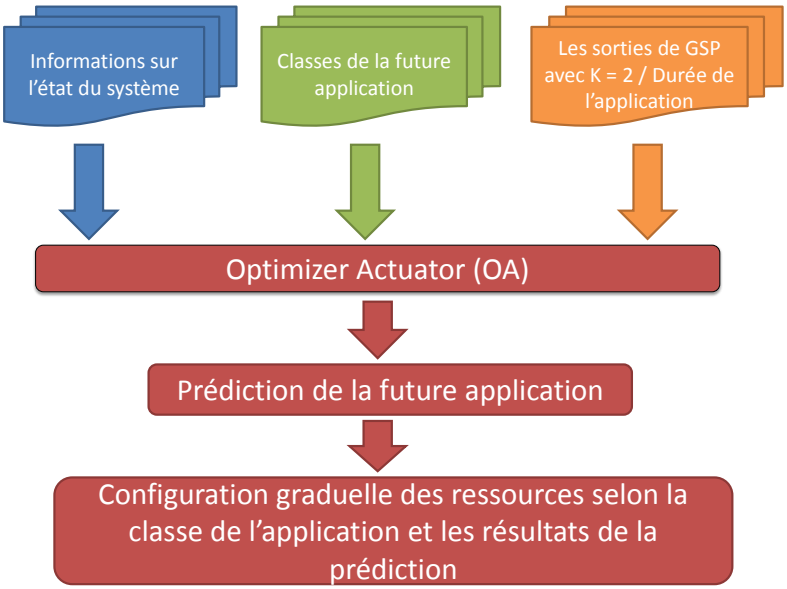

FIGURE 5.7 – Fonctionnement de l'OA

Le mode de fonctionnement de l'OA est comme suit:

1. Les informations en temps réel sur l'état du système sont collectées en premier lieu. Ces informations représentent l'application en cours d'exécution, les applications en background et les ressources en cours d'utilisation.
2. L'OA consulte ensuite les résultats de la prédiction en fonction de l'application en cours d'exécution, ainsi que sa durée d'utilisation.
3. Après ces deux premières actions, l'OA est en mesure de connaitre la future application qui sera lancée, ainsi que le moment où cela va se produire via le mécanisme de prédiction.
4. L'OA consulte les résultats de la classification *offline* pour connaitre les besoins en ressources de l'application prédite.
5. Avant le début présumé de l'application prédite, l'OA commence à ajuster les ressources graduellement en fonction des besoins de l'application.
6. L'OA s'assure de ne pas désactiver les ressources si elles sont utilisées par les applications en background.
7. Dans le cas où l'application prédite n'a pas été classifiée, les informations nécessaire à la classification sont collectées et la gestion des ressources est réalisée par l'OS.

8. Nous pouvons également être dans le cas où les résultats de la prédiction indiquent le même nombre d'occurrence pour deux applications différentes. Dans ce cas là, l'OA prend en compte la classe la plus élevée. Prenons l'exemple d'un utilisateur qui lance *Facebook* et les résultats de la prédiction indiquent que l'application suivante est soit *Skype* soit *2048*. L'OA ne désactive pas la Wi-Fi dans ce cas là et alloue la fréquence $Max(fc_{2048}, fc_{Skype})$.

Dans la phase de configuration des ressources, l'OA réalise l'ajustement de façon graduelle. Dans le cas où, l'application A est suivie par l'application B et les ressources exigées par ces deux applications sont différentes, l'OA configure les ressources nécessaires pour B avant son début supposé. La figure 5.8 présente un aperçu sur l'ajustement réalisée par l'OA.

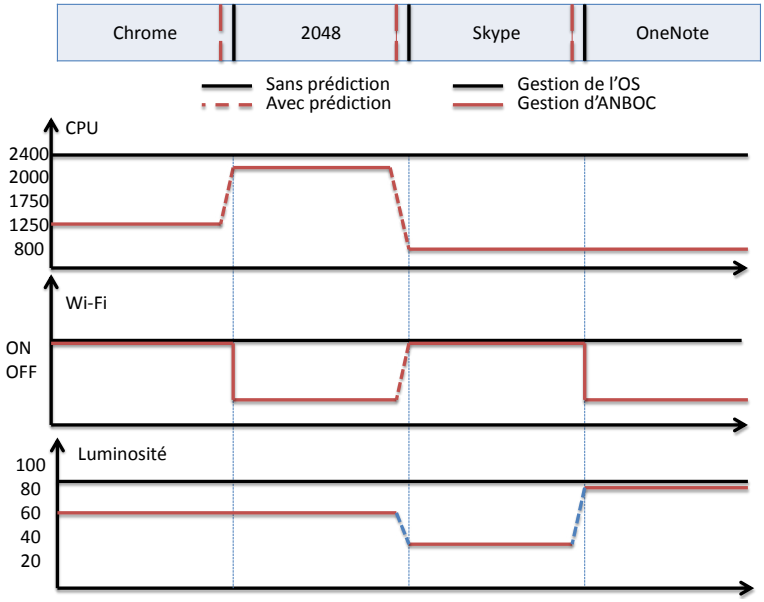

FIGURE 5.8 – La phase d'ajustement réalisée par l'OA

La gestion de l'OS est représentée en noir et la gestion proposée avec ANBOC est représentée en rouge. La barre verticale noire représente la fin de l'application, la barre verticale rouge représente l'instant où ANBOC commence à ajuster les ressources pour l'application suivante.

Ce changement graduel est réalisé avant la fin supposée de l'application. Cette fin représente 1% à 10% de la durée totale de l'application. Cette variation est encore une fois ajustable et dépend de la durée de l'application et du comportement de l'utilisateur. Plus la durée de l'application est supposée être longue, plus la valeur du pourcentage est petite.

Cet ajustement se fait sur la base de la classification et peut se faire à la hausse ou à la baisse, dépendamment de la classe de l'application. Une configuration de ressources soudaine et brusque peut impacter négativement la satisfaction de l'utilisateur, qui réagira dans la majorité du temps par l'augmentation maximale de toutes ses ressources. Sur la figure 5.8, nous avons la gestion de trois ressources qui sont le CPU, la Wi-Fi et la luminosité. Dans la partie expé-

rimentale ; par manque de temps, nous n'avons pas appliqué notre approche sur la luminosité. Mais nous rappelons qu'ANBOC est une solution générique applicable à toutes les ressources hardwares à condition de réaliser la classification au préalable. Concernant le CPU, un changement brusque au niveau de la fréquence du processeur risque de ralentir les performances du système et impacter l'expérience de l'utilisateur. Enfin, concernant la gestion de l'interface Wi-Fi, étant donné que nous avons uniquement deux états possibles, l'ajustement graduel n'est possible que de l'état ON à l'état OFF.

5.5.1 Configuration des ressources hardwares

- Pour la Wi-Fi, lorsque nous nous passons d'une application qui nécessite une connectivité à une application qui n'en nécessite pas, la prédiction n'est pas utilisée. Nous nous contentons de désactiver l'interface Wi-Fi en se basant uniquement sur la classification de l'application en cours. Nous nous assurons tout de même que les processus en background n'utilisent pas la connectivité sans fil. La prédiction est utilisée lors du passage de l'état OFF à l'état ON. L'OA active l'interface Wi-Fi avant le début de l'application appartenant à la classe Wi-Fi ON, ce qui permet d'éviter toute intervention manuelle de l'utilisateur.
- Pour le CPU, nous avons modifié l'algorithme qui contrôle la fréquence de l'horloge nommé *ondemand governor*, en l'enrichissant par les techniques de classification.

Nous présentons dans ce qui suit l'algorithme *ondemand governor* de *Windows* suivi de notre modification basée sur la classification CPU. Dans l'algorithme *ondemand governor*, on applique le principe du DVFS présenté précédemment.

Les deux phases du *ondemand governor* de *Windows* sont:

- Phase d'augmentation de fréquence: l'algorithme récupère chaque 150 ms la charge d'utilisation du processeur, il augmente ensuite la fréquence si la charge d'utilisation a augmenté d'au moins 20% par rapport à la charge précédente.
- Phase de diminution de fréquence: l'algorithme récupère l'utilisation du processeur 500 ms après le dernier ajustement. La fréquence est diminuée si la charge d'utilisation a baissé d'au moins 20%.

Algorithme 2 : Pseudo-code de l'algorithme ondemand governor de Windows

1 **Données**: **p-states** : Tableau de P-State supportés par le processeur;
2 **current-p-state**: Un entier représentant le p-state actuel du processeur;
3 **p-min**: Un entier représentant le p-state minimal supporté par le processeur;
4 **Début**
5 **/* La phase d'augmentation*/**
6 **si** *150 ms se sont écoulées depuis le dernier ajustement de la fréquence* ***ET*** *la charge d'utilisation du CPU a augmenté de 20% depuis la dernière évaluation* **alors**
7 current-p-state++;
8 ajuster la fréquence à p-states[current-p-state] dans les 10 prochaines ms;
9 **/* La phase de diminution*/**
10 **si** *Si 500 ms se sont écoulées depuis le dernier ajustement de la fréquence* ***ET*** *la charge d'utilisation du CPU a diminué de 20% depuis la dernière évaluation* ***ET*** *la fréquence est supérieur à* ***p-min*** **alors**
11 current-p-state–;
12 ajuster la fréquence à p-states[current-p-state] dans les 10 prochaines ms;

L'algorithme 3 représente l'ajustement des fréquences basé sur la classification d'ANBOC. Nous reprenons le DVFS de *Windows* en prenant en compte les résultats de la classification. L'ajustement de la fréquence dépend alors de la classe de l'application en plus de la charge d'utilisation du CPU.

Algorithme 3 : Pseudo-code de l'algorithme ondemand governor d'ANBOC

1 **Données**: **p-states-ANBOC** : Tableau des 4 P-States;
2 **current-p-state**: Un entier représentant le p-state actuel du processeur;
3 **p-min**: Un entier représentant le p-state minimal supporté par le processeur;
4 F_c: La fréquence de la classe de l'application; **Début**
5 **/* La phase d'augmentation*/**
6 **si** *La fréquence du processeur* < F_c **alors**
7 **si** *150 ms se sont écoulée depuis le dernier ajustement de la fréquence* ***ET*** *la charge d'utilisation du CPU a augmenté de 20% depuis la dernière évaluation* **alors**
8 current-p-state++;
9 ajuster la fréquence à p-states[current-p-state-ANBOC] dans les 10 prochaines ms;
10 **/* La phase de diminution*/**
11 **si** *500 ms se sont écoulées depuis le dernier ajustement de la fréquence* ***ET*** *la charge d'utilisation du CPU a diminué de 20% depuis la dernière évaluation* ***ET*** *la fréquence est supérieur à* ***p-min*** **alors**
12 current-p-state–;
13 ajuster la fréquence à p-states[current-p-state-ANBOC] dans les 10 prochaines ms;

Nous rappelons qu'avec la classe de l'application (c_{app}), l'OA affecte un seuil maximale à la fréquence $F_c(app)$. En d'autres termes, pendant l'exécution d'une application donnée, la fréquence allouée ne dépassera jamais $F_c(app)$. La figure 5.9 illustre ce fonctionnement.

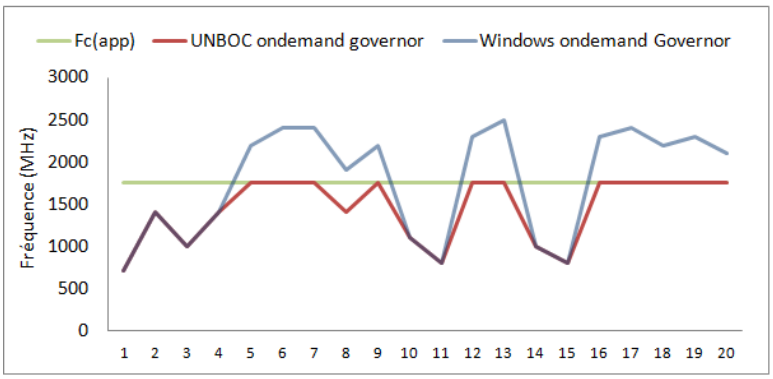

FIGURE 5.9 – Différence entre *ondemand governor* de l'OS avec ANBOC

Nous remarquons dans la figure ci-dessus que la fréquence allouée par AN-

BOC ne dépasse jamais le seuil de $F_c(app)$. Tandis que la fréquence allouée par *Windows* peut atteindre la fréquence maximale. Les deux algorithmes permettent un fonctionnement optimal de l'application, la différence est qu'avec ANBOC, il est possible d'obtenir un gain d'énergie important que nous évaluons dans la section 5.6.

5.5.2 Satisfaction de l'utilisateur

La satisfaction de l'utilisateur est un facteur primordial dans toutes les approches d'optimisations de la consommation d'énergie. Dans cette contribution, nous avons deux mécanismes pour jauger la satisfaction de l'utilisateur, illustrés dans la figure 5.10.

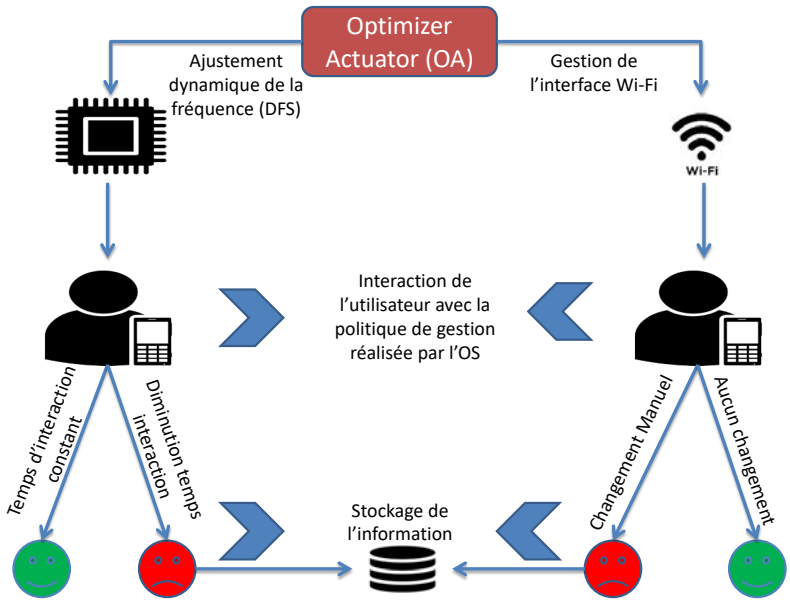

FIGURE 5.10 – La phase d'ajustement réalisée par l'OA

- Pour jauger la satisfaction de l'utilisateur après la configuration de la Wi-Fi, nous considérons toute réactivation manuelle de l'interface comme un signe d'insatisfaction. La configuration choisie par l'utilisateur est sauvegardée et est appliquée pour le prochain scénario similaire. La classification de l'application est également modifiée selon la valeur émise par l'utilisateur.
- Concernant la satisfaction de l'utilisateur en fonction de la fréquence du processeur, nous prenons en compte le temps écoulé entre deux interactions. Si le temps entre deux interactions augmente durant l'utilisation de toute l'application, nous concluons qu'une fréquence plus élevée est demandée par l'utilisateur. Cette information est stockée et est prise en considération pendant la prochaine exécution de l'application dans le même contexte temporel.

Concernant la prédiction, sa correction se fait dans le temps. À chaque fois que l'utilisateur lance une autre application que celle prédite, la séquence est sto-

ckée. Nous fixons un seuil d'erreur pour chaque séquence et un indice de fiabilité. Quand ce seuil d'erreur est atteint, nous relançons le traitement réalisé par GSP. Ce seuil d'erreur a été fixé à 30%. Les séquences pour lesquelles les erreurs de prédiction sont fréquentes, obtiennent un indice de fiabilité bas et seront supprimées des données en entrée de GSP.

5.6 Résultats expérimentaux

Comme mentionné précédemment, le DCM a été distribué aux étudiants de l'université de Valenciennes volontaires pour nous fournir des traces réelles d'utilisation. Cette collecte de données s'est faite durant deux semaines.

Nous avons vu dans l'état de l'art qu'il était toujours difficile d'avoir un comportement fiable de l'utilisateur quand ce dernier est conscient que ses données vont être étudiées. C'est pour cela que nous procédons à une étude préliminaire des *logs* reçus dans le but de choisir ceux qui sont les plus pertinents. Nous sélectionnons 6 utilisateurs réels sur lesquels nous avons réalisé notre compagne de collecte. Nous générons ensuite les séquences d'applications correspondantes via notre l'algorithme GSP. Ces tests expérimentaux nous permettent de valider la faisabilité de notre solution dans un premier temps. Nous présentons dans ce qui suit les scénarios et les résultats de classification.

5.6.1 Résultats de la prédiction et de la classification

La figure 5.11 présente les résultats de la prédiction pour les six utilisateurs.

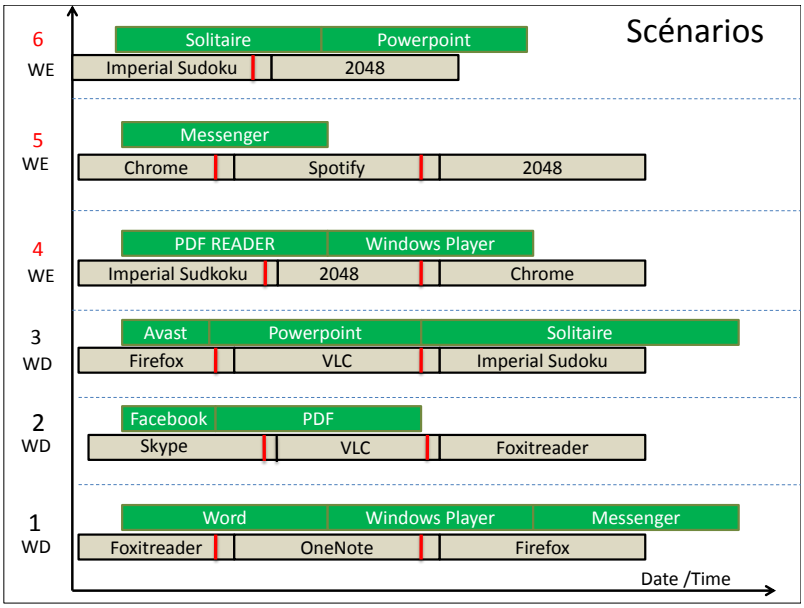

FIGURE 5.11 – Séquences d'applications d'utilisateurs réels

Dans la figure ci-dessus, les utilisateurs 1,2 et 3 utilisent leurs systèmes durant les jours de semaine, tandis que les utilisateurs 4,5 et 6 l'utilisent pendant

le weekend. Les applications en background sont représentées en vert et les applications en premier plan en gris. Les lignes rouges sur les applications représentent le début supposé de l'ajustement des ressources pour l'application prédite. Avant d'évaluer l'efficacité de notre approche, nous classifions les applications selon leurs besoins en puissance de calcul et de connectivité. Le tableau 5.5 représente la classification des applications selon les besoins en connectivité communes à tous les utilisateurs.

Tableau 5.5 – Classification Wi-Fi des applications

Apps	*Wi-Fi Class*
Firefox	On
Skype	On
Chrome	On
Spotify	On
VLC	Off
2048	Off
Sudoku	Off
Foxit	Off
OneNote	Off

La classification *offline* pour le CPU est propre à chaque utilisateur comme le montre le tableau 5.6.

Tableau 5.6 – Résultats de la classification CPU pour les 6 utilisateurs

Apps	Classification CPU pour les 6 utilisateurs					
	Usr 1	Usr 2	Usr 3	Usr 4	Usr 5	Usr6
Firefox	M		F			
Skype		F				
Chrome					M	
Spotify		F			F	
VLC		F	F			
2048				E	M	M
Sudoku			F	F		F
Foxit	F	F				
OneNote	F					

Dans ce tableau $UsrX$ représente l'identifiant de l'utilisateur. La valeur de la classe F représente les besoins CPU les plus faibles, M est la valeur moyenne et E est la classe la plus élevée.

Le tableau 5.6 expose une information importante qui est la différence comportementale entre les différents utilisateurs. Nous pouvons noter qu'il est possible pour une application d'appartenir à deux classes différentes selon l'utilisateur correspondant. Par exemple, pour l'utilisateur 1, *Firefox* a un besoin en CPU moyen, alors que ce besoin est faible avec l'utilisateur 3. Cette situation est due aux différentes interactions de chaque utilisateur et la charge de travail du processeur qui en résulte.

5.6.2 L'évaluation d'ANBOC

La figure 5.12 montre l'impact de notre solution sur la consommation énergétique pour les 6 utilisateurs. Nos résultats expérimentaux montre que la surcouche ANBOC peut réduire la consommation énergétique du système de 33%, en comparaison avec la gestion proposée par le système d'exploitation.

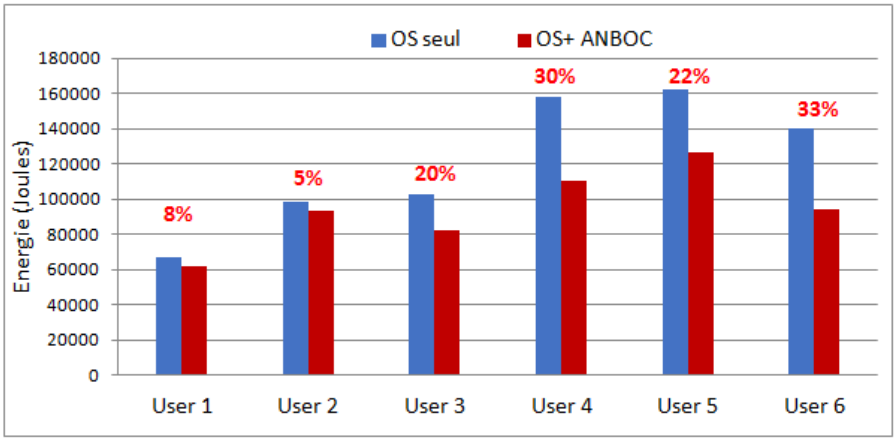

FIGURE 5.12 – Consommation d'énergie pour chaque utilisateur

Les résultats obtenus nous montrent que les gains d'énergie varient largement d'un utilisateur à un autre.

- pour les utilisateurs 1 et 2, les réductions d'énergie consommée obtenues sont en moyenne de 8% et 5%.
- pour les utilisateurs 3 et 4, ce gain énergétique est respectivement de 19% et 30%.
- Pour l'utilisateur 5, la quantité d'énergie économisée est de 22%, tandis que pour l'utilisateur 6, nous obtenons le gain le plus important qui est à hauteur de 33%.

Ces différences de réductions d'énergie sont dues à plusieurs paramètres qui sont principalement l'application exécutée et la manière dont les utilisateurs interagissent avec leurs systèmes. Par exemple les utilisateurs 4 et 5 exécutent tous deux les applications *2048* et *chrome*, mais les gains obtenus pour l'utilisateur 4 sont plus élevés de 8% par rapport à ceux obtenus avec l'utilisateur 5. La raison de cette variation est la différence des classes de ces deux applications.

Ce résultat confirme qu'il existe un impact considérable du type de l'application dans la réduction de puissance électrique. Les applications caractérisées par une charge de travail peu intense, qui ne requièrent aucune connectivité et

une basse interaction avec l'utilisateur sont celles qui offrent les opportunités de réduction d'énergie les moins importantes et vice versa.

Le cas de l'utilisateur 2 illustre très bien cela : en effet cet utilisateur exécute *VLC*, *FoxitReader* et *Spotify*. Ces applications ont des besoins de calcul et de connectivité peu élevés ce qui limite les opportunités de réduction d'énergie.

Dans le but d'expliciter l'impact de chaque application dans la réduction de la consommation d'énergie, nous présentons la figure 5.13 ci-dessous.

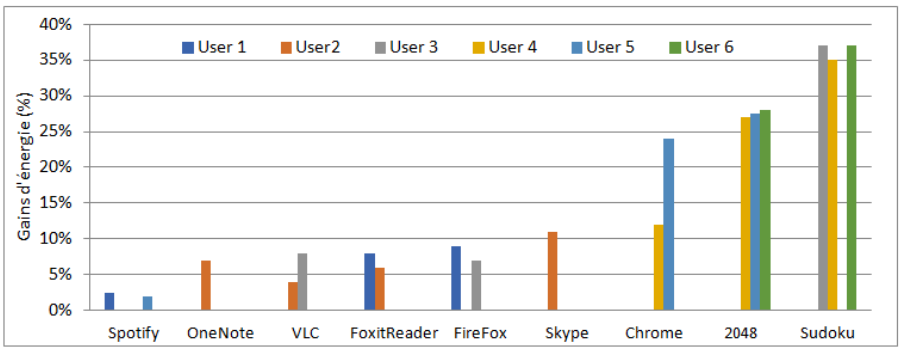

FIGURE 5.13 – Réductions d'énergie obtenues avec chaque application

En analysant la figure, nous notons que toutes les applications impactent la consommation d'énergie de manière différente. Par exemple pour *Spotify* qui est caractérisée par de faibles besoins CPU, la puissance réduite est également faible et s'élève à 2.5%. À contrario, nous avons l'application *OneNote* qui a les mêmes caractéristiques que *Spotify* en besoins de calcul, mais nous offre un gain plus élevé de 7%, car cette dernière ne requiert pas de Wi-Fi.

Nous arrivons à la conclusion que l'OS alloue une fréquence élevée, en comparaison aux besoins réels de l'utilisateur et de l'application. Certaines applications ne sont pas optimisées sur le plan de la consommation énergétique. *Imperial Sudoku*, *2048* en sont de parfaits exemples. L'OS a tendance à leur allouer une fréquence élevée tandis qu'ANBOC les classifie comme des applications avec de faibles besoins.

Nous concluons également qu'il existe une forte corrélation entre la manière dont l'utilisateur interagit avec son système et la consommation de puissance. En effet *Google Chrome* réduit la consommation d'environ 11% quand elle est dans la classe CPU faible tandis qu'elle réduit la consommation de 23% quand elle est dans la classe moyenne.

La figure 5.14 présente la réduction d'énergie obtenue avec notre approche pour nos deux composants hardwares et pour les scénarios d'utilisation.

En analysant la figure, nous notons:

- L'énergie économisée avec la Wi-Fi est relative au temps de l'exécution de l'application. Ce gain est obtenu en mettant l'interface Wi-Fi en OFF et est constant.La consommation d'énergie de l'ensemble du système est réduite de 8%.
- La réduction totale de puissance obtenue avec le DVFS d'ANBOC est relative aux applications exécutées et le comportement de l'utilisateur ainsi que ses interactions.

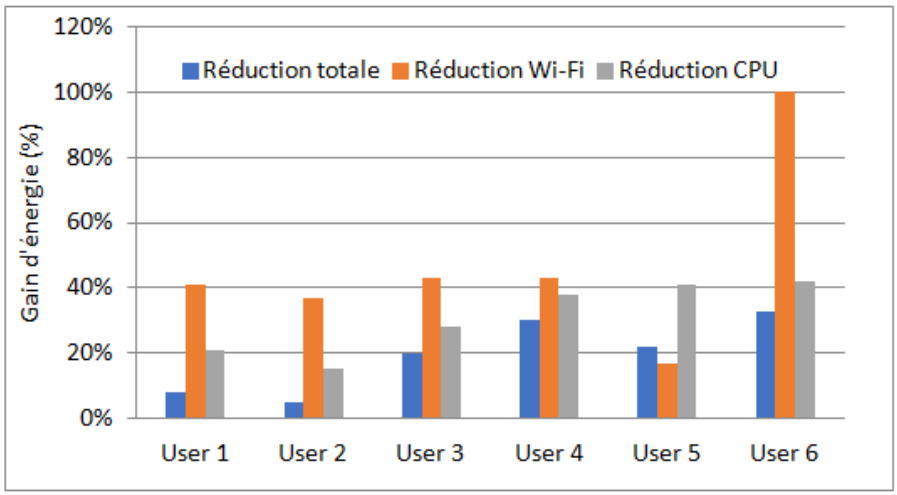

FIGURE 5.14 – Réduction d'énergie normalisée pour chaque composant

Après avoir présenté les différentes réductions d'énergie obtenues avec ANBOC, nous présentons dans ce qui suit le cout de notre solution.

5.6.3 Coût d'ANBOC

Le coût des différentes optimisations réalisées par ANBOC est mesuré selon l'utilisation du processeur, la mémoire vive et la consommation de puissance. Ces mesures sont présentées dans le but de démontrer que notre solution basée sur l'IECSDK n'impacte pas négativement les performances du système ni l'expérience de l'utilisateur. Au contraire, cela offre une réduction de la consommation d'énergie, en comparaison avec la gestion de puissance de l'OS.

Les différents coûts sont présentés dans la table 5.7.

Tableau 5.7 – Coût du ANBOC

Mécanismes	*Puissance (W)*	*CPU (%)*	*Ram (MB)*	*Temps (s)*
Classif. offline	8.7	12%	14.74	12
Prédiction	6	0.3%	3.74	2

Nous avons mesuré le coût des mécanismes de classification ainsi que celui de la prédiction. La classification dure en moyenne 12 secondes. Durant ces 12 secondes, nous avons un surplus de 8.7 W en matière de consommation de puissance, ainsi que 12% d'utilisation CPU et 14.74 MB de mémoire vive.

Pour la prédiction, le durée d'exécution est minime et s'élève à deux secondes, le surplus en consommation de puissance est estimé à 6 W. Les surplus concernant le CPU et la RAM sont relativement faibles et s'élèvent respectivement à 0.3% et 3.74 %. Notre solution requiert quelques secondes pour être chargée au niveau de l'OS et son coût est négligeable. En se basant sur ces résultats, ANBOC peut être facilement embarqué dans les tablettes actuelles ou encore les smartphones de nouvelle génération.

Nous présentons également des résultats additionnels dans le tableau 5.8. Ces tests montrent l'ajustement de la fréquence CPU pour *Youtube* et *Word*. La fréquence est présentée en tant que ratio par rapport à la fréquence maximale supportée par l'*Ultrabook*.

Tableau 5.8 – Résultats Du DFS

Apps	*Temps (S)*	*CPU freq (%)*	*CPU cons (W)*	*Gain (%)*	*Latence*
Youtube	360	100	5.80	14.34	0
Youtube	360	30	4.97	14.34	0
Word	600	100	5.11	7.74	0
Word	600	50	4.72	7.74	0
Word	600	30	4.63	9.64	0

Pour *Youtube*: nous n'avons aucune latence en fixant le seuil de la fréquence maximale à 30%, en comparaison avec un ajustement de la fréquence maximale à 100%. Nous notons aussi une différence de 14.34% concernant la consommation du CPU.

Pour *Word*, réduire la fréquence CPU de 70% ne dégrade pas les performances et un gain d'énergie de 10% est obtenu. La latence qui est nulle dans tous les cas de figure laisse présager un utilisateur satisfait et aucune dégradation des performances du système.

5.7 Conclusion

Dans ce chapitre, nous avons présenté ANBOC, qui est une sur-couche logicielle dont la vocation est d'améliorer la gestion d'énergie proposée par l'OS. Comme toutes nos contributions, cette sur-couche est basée sur le modèle CPA et sur l'IECSDK. ANBOC est basé sur l'analyse des besoins des applications et des interactions de l'utilisateur. Dans cette contribution, nous proposons deux mécanismes qui sont la classification et la prédiction des applications.

La classification supervisée des applications est réalisée selon leurs besoins en CPU et en connectivité Wi-Fi. Cette classification a été implémentée à l'aide d'un classifieur bayésien naïf et ne requiert pas de phase d'apprentissage. Pour la prédiction des applications, nous utilisons l'algorithme de *sequence mining* GSP. Cette prédiction est réalisée en fonction des données temporelles de l'utilisateur ainsi que de la durée d'exécution moyenne des applications. La classification et la prédiction permettent d'ajuster les ressources hardwares à la baisse sans impacter négativement la satisfaction de l'utilisateur.

Nous avons démontré que notre approche alliée à l'utilisation des ILs et des ALs pour la configuration de l'antenne Wi-Fi et l'ajustement dynamique de la fréquence est susceptible de réduire la consommation d'énergie jusqu'à 33%. Ces gains peuvent varier selon les utilisateurs.

À contrario des méthodes d'optimisation de la consommation d'énergie existantes, ANBOC prend en considération un facteur clé qui est l'utilisateur final et sa satisfaction lors des configurations hardwares. Il existe néanmoins quelques

nuances, dans le sens où les gains d'énergie apportés par ANBOC peuvent varier en fonction de l'attitude de l'utilisateur et les applications exécutées par ce dernier. Nous pouvons nous retrouver avec des utilisateurs où les opportunités de réduction d'énergie ne sont pas nombreuses. Cela est dû à la difficulté de tirer un schéma d'exécution discernable chez ces utilisateurs.

Le chapitre suivant présente le framework Enormous. Nous tentons à travers ce framework de répondre aux limitations d'ANBOC en proposant une solution axée sur l'ensemble du contexte de l'utilisateur et pas uniquement le besoin des applications.

Chapitre 6

ENergy Optimization for MObile plateforms using User needS

Nous présentons dans ce chapitre la troisième contribution de cette thèse. Menée à terme, ce travail nous a permis de réaliser le framweork intitulé ENergy Optimization for MObile plateforms using User needS (Enormous). Ce framework s'inscrit naturellement dans la continuité des composants réalisés dans les chapitres 4 et 5 précédents, tout en essayant de combler leurs limitations. Comme les composants SBOC et ANBOC, Enormous a été mis au point dans le but d'améliorer la gestion de la consommation d'énergie du système mobile, en proposant des optimisations qui ne sont pas prises en considération par le système d'exploitation.

Contrairement au chapitre 4, où nous nous sommes focalisés exclusivement sur les données des capteurs embarqués pour la gestion du couple luminosité-volume et au chapitre 5 qui était centré sur la classification et la prédiction des applications, avec Enormous, nous proposons une solution plus complète et englobante. Ce framwork est axé sur toutes les informations qui peuvent influencer de loin ou de prés le comportement de l'utilisateur et par conséquent, sa consommation d'énergie. Les informations considérées dans Enormous proviennent de l'utilisateur, de son système et de l'environnement.

Enormous obéit également au schéma CPA et est responsable de l'identification du contexte de l'utilisateur dans le but de comprendre ses habitudes et ses préférences en matière de ressources hardwares. Comme dans ANBOC, cette identification permet de fournir à l'utilisateur uniquement les ressources nécessaires au fonctionnement optimal de son système. Le framework Enormous implémente des algorithmes d'intelligence artificielle dans le but d'aboutir à un compromis entre les opportunités de réduction de la consommation de puissance et la satisfaction de l'utilisateur. Contrairement à ANBOC où la satisfaction de l'utilisateur était jaugée après chaque optimisation, dans Enormous, les optimisations hardwares sont construites à partir de la satisfaction de l'utilisateur.

Notre framework configure dynamiquement la fréquence du processeur, ajuste le niveau du volume, de la luminosité de l'écran et gère l'état de l'interface Wi-Fi. De façon similaire à nos solutions précédentes, nous pouvons également étendre Enormous au delà ces ressources en gérant le GPS, le GPU, etc. En se basant uniquement sur la gestion des quatre ressources hardware susmentionnées, nous parvenons à réduire la consommation de puissance à hauteur de 35% en comparaison avec la politique de gestion standard réalisée par l'OS. Ce

gain d'énergie est obtenu sans altérer le niveau de satisfaction de l'utilisateur et avec un surcout négligeable.

Ce chapitre est organisé comme suit. La section 6.1 situe le contexte et les motivations de cette contribution. La section 6.2 présente l'architecture fonctionnelle d'Enormous, en détaillant les modules softwares de notre framework. La section 6.3 est dédiée à la classification du contexte de l'utilisateur. Les sections 6.4 et 6.5 présentent l'architecture des réseaux de neurones ainsi que la phase d'apprentissage. Dans la section 6.6, nous présentons les résultats expérimentaux obtenus avec notre approche et les comparons avec la politique de gestion d'énergie implémentée dans le système d'exploitation. Nous terminons avec la conclusion de ce chapitre et les perspectives.

Sommaire

6.1 Contexte et motivation **120**
6.2 Architecture fonctionnelle d'Enormous **124**
6.3 Classifiation des ressources via l'apprentissage supervisé . . **128**
6.3.1 Classification des ressources 130
6.3.2 Création des modèles prédictifs 134
6.4 Architecture de nos réseaux de neurones **135**
6.4.1 Définition de l'architecture de nos MLP 135
6.4.2 Notation utilisée pour la représentation de nos MLP 137
6.5 Algorithme d'apprentissage propagation du gradient **138**
6.5.1 Feed Forward Back-Propagation(FBP) 143
6.5.2 Cascade Back-Propagation (CBP) 143
6.5.3 Paramétrage de nos MLP 143
6.6 Résultats expérimentaux **146**
6.6.1 Outils et environnement expérimental 146
6.6.2 Résultats de la réduction de consommation de puissance avec Enormous 147
6.6.3 Coût d'Enormous 156
6.7 Conclusion **157**

6.1 Contexte et motivation

Cette troisième contribution est la suite logique des deux composants SBOC et ANBOC, avec le même objectif de réduction énergétique. De contribution en contribution, nous avions tenté de proposer des solutions qui soient les plus englobantes possible pour les besoins de l'utilisateur et sa satisfaction, tout en réduisant sa consommation d'énergie. Rappelons qu'avec SBOC, nous mettons en place une approche basée sur les capteurs embarqués présents dans la plateforme mobile pour configurer les ressources dynamiquement, sans l'utilisation de techniques d'intelligence artificielle. Cette approche était une solution soucieuse de l'environnement de l'utilisateur.

Avec ANBOC, nous proposons une solution basée principalement sur les applications avec une classification *offline* de ces dernières, combinée à une prédiction des futures applications qui seront lancées par l'utilisateur.

Dans la contribution de ce chapitre, nous introduisons Enormous qui se veut une solution plus complexe et plus complète que les deux précédentes. En effet avec ce framework, nous ne nous concentrons pas uniquement sur les informations fournies par les capteurs embarqués ou sur les applications, mais nous nous intéressons à tout ce qui englobe le contexte de l'utilisateur et ses habitudes.

Avant de rentrer dans les détails d'Enormous, nous définissons ce qu'est le contexte de l'utilisateur et ce qu'est une solution soucieuse du contexte. La littérature dans le domaine est fournie et il existe un grand nombre de travaux ayant définit la notion de contexte, comme cela a été étudié dans le survey [7]. Parmi les différentes définitions du contexte, celle que nous jugeons la plus complète et la plus en adéquation avec notre travail a été mentionnée dans [1, 31, 32]: « Le contexte est toute information qui peut être utilisée pour caractériser la situation d'une entité. Une entité est une personne, un lieu ou un objet qui est considéré pertinent pour l'interaction entre l'utilisateur et l'application, y compris l'utilisateur et l'application eux mêmes ».

Pour ce qui est des systèmes soucieux du contexte [1], nous reprenons de nouveau la définition donnée par Dey et Abowd [1]: « Un système est sensible au contexte s'il utilise le contexte pour fournir des informations et/ou des services pertinents pour l'utilisateur, où la pertinence dépend de la tâche de l'utilisateur. »

En s'inspirant de ces définitions du contexte, nous présentons le contexte de l'utilisateur comme un ensemble d'informations répondant aux interrogations suivantes:

- Quand est ce que l'utilisateur est actif ?
- Quels sont les besoins de l'utilisateur en matière de configuration hardware ?
- Quelles applications sont souvent utilisées par l'utilisateur ?
- À quelle allure et à quelle vitesse se déplace l'utilisateur ?
- Quel est le mode de vie de l'utilisateur ?
- Quel est le type d'emploi du temps de l'utilisateur ?

Cette liste de questions est loin d'être exhaustive et notre définition du contexte se doit de répondre à un maximum d'interrogations sur l'utilisateur et l'état de son système. Cela doit être réalisé tout en respectant les contraintes de confidentialité et de façon automatique, sans l'intervention de l'utilisateur et sans passer par des questionnaires explicites. En généralisant, le contexte de l'utilisateur doit couvrir un large panel d'informations. Ces informations peuvent être subdivisées en trois catégories distinctes en fonction de leur nature.

- Les données spatio-temporelles.
- Les informations sur les habitudes de l'utilisateur et ses préférences comme les applications lancées et le temps d'activité des applications.
- Les informations sur le système et les ressources hardwares sollicitées par l'utilisateur.

Les informations contenues dans ces trois catégories sont corrélées et exploitées par la suite, comme nous allons le voir dans les sections suivantes. Comme toutes nos approches, Enormous tire profit du flux important d'informations disponbile dans le système. Nous nous appuyons également sur les études réalisées dans [89] qui soulignent le fait que les ressources disponibles dans un sys-

1. En anglais: context aware systems

tème ne sont pas toujours utilisées au maximum de leurs capacités. Par conséquent, un utilisateur ne va pas exploiter toutes les ressources hardwares allouées par son système. Cela nous aide à consolider notre hypothèse qui stipule que la satisfaction de l'utilisateur et son expérience ne sont pas synonymes d'une configuration maximale des ressources hardwares, comme nous avons pu le voir avec ANBOC. Configurer les ressources hardwares à leur maximum implique une consommation d'énergie plus importante et par conséquent, une diminution de la durée de vie de la batterie, ce qui impacte négativement la satisfaction de l'utilisateur.

Partant de ce postulat, nous proposons avec Enormous différentes étapes pour évaluer les besoins de l'utilisateur afin de déterminer les configurations hardwares suffisantes et nécessaires pour sa satisfaction. En parallèle, ces configurations doivent également mener à une réduction de la consommation d'énergie du système. Pour réaliser cela, nous suivons le même schéma procédurale que les approches précédentes basé sur le modèle CPA présenté, dans le chapitre 1. En effet, Enormous exploite les capteurs embarqués et les APIs exposées par le système d'exploitation pour collecter un ensemble d'informations qui définissent le contexte de l'utilisateur. Ce contexte est ensuite classifié via des algorithmes d'intelligence artificielle. La classification est une étape cruciale sur laquelle repose l'aspect novateur de notre framework. Elle permet de corréler les informations du contexte avec les ressources requises par l'utilisateur. Cette classification représente la partie la plus importante de cette contribution et ouvre une nouvelle dimension dans les travaux qui s'intéressent à la consommation énergétique des systèmes mobiles et plus précisément aux travaux se focalisant sur l'utilisateur, comme ceux présentés dans la section 2.5.1. Les résultats de la classification fourniront par la suite toutes les informations nécessaires pour la configuration des ressources hardwares. Ces configurations doivent répondre à trois critères majeurs:

- Réduire la consommation d'énergie globale du système;
- Ne pas altérer la satisfaction de l'utilisateur;
- Ne pas dégrader les performances du système;

Les trois phases principales d'Enormous sont illustrées dans la figure 6.1.

Ces phases peuvent être énumérées comme suit:

1. Nous exploitons les capteurs embarqués physiques présents sur le système mobile, ainsi que les interfaces de programmation APIs exposées par le système d'exploitation pour la collecte de données. Cette collecte vise un large ensemble de données relatives à l'utilisateur, ses applications, sa consommation de ressources hardwares, etc. Toutes ces informations constituent le contexte de l'utilisateur et représentent le point d'entrée d'Enormous. Cette phase de collecte de données est assurée par un module logiciel qui est composé à son tour de trois sondes. Chaque sonde logicielle correspond à un type spécifique d'informations comme cela est présenté dans la section 6.2. Toutes ces sondes sont implémentées en background et s'exécutent en mode *offline* pendant une période de temps prédéfinie. Les informations collectées sont stockées dans un fichier XML unifié pour être traitées en aval. Cette phase correspond à la première étape du modèle CPA.

2. Dans cette seconde phase, nous proposons une nouvelle classification du

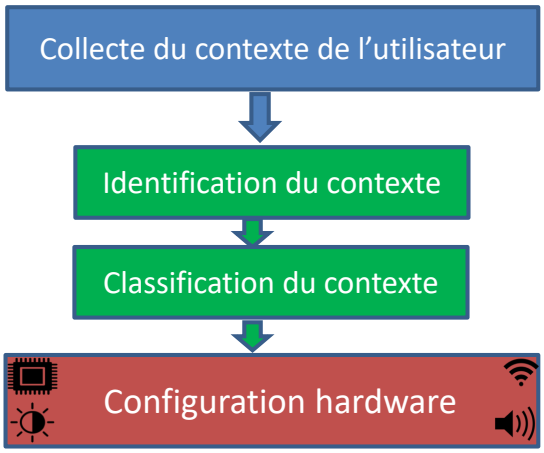

FIGURE 6.1 – Abstraction de l'architecture globale d' Enormous basée sur le modèle CPA

contexte de l'utilisateur, collecté précédemment. Les informations collectées sont traitées via des algorithmes d'intelligence artificielle et plus précisément des techniques d'apprentissage supervisé. Ces techniques d'apprentissage ont pour but de repérer toutes les régularités et les patterns dans le comportement de l'utilisateur et ses interactions avec le système. Pour réaliser cette tâche, des réseaux de neurones avec différentes architectures ont été implémentés et sont comparés dans le but d'adapter leurs architectures aux différents utilisateurs. Ces réseaux de neurones sont exécutés dynamiquement et permettent de fournir les configurations hardwares nécessaires à chaque utilisateur. La classification sert à identifier les opportunités de réduction d'énergie et permet de les exploiter par la suite. Évidemment, cela doit se faire sans détériorer la satisfaction de l'utilisateur. Cette phase correspond à la seconde étape du modèle CPA.

3. Dans la troisième phase, les résultats de la classification sont exploités pour implémenter des politiques de gestion de puissance. Les techniques telles que l'allocation dynamique de fréquence et du voltage du processeur DVFS, la gestion de l'interface de connexion sans fil Wi-Fi, l'ajustement de la luminosité et du volume peuvent être implémentées efficacement et tirer pleinement profit des résultats de la classification. Cette dernière phase correspond à l'étape finale du modèle CPA.

L'objectif est toujours d'offrir aux utilisateurs aux profils hétérogènes et aux différents besoins une gestion personnalisée de la consommation d'énergie. Cette gestion dépendra de leurs habitudes, besoins, style de vie, profession, etc. Toutes les actions de réduction de la consommation d'énergie sont réalisées en *runtime* et sont transparentes à l'utilisateur. Le but étant de minimiser sa participation.

6.2 Architecture fonctionnelle d'Enormous

Dans le but de mettre en place les trois phases mentionnées dans la section 6.1, plusieurs modules logiciels ont été développés. Ces modules communiquent entre eux pour aboutir à notre objectif final de réduction de la consommation d'énergie. Les modules principaux d'Enormous sont présentés dans la figure 6.2 ci-dessous.

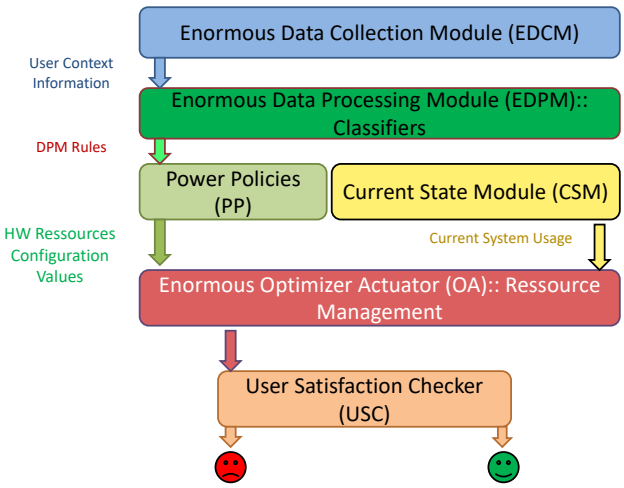

FIGURE 6.2 – Modules logiciels d'Enormous

Nous avons:

1. Enormous Data Collection Module (EDCM): ce module est une extension du DCM implémenté dans ANBOC. L'EDCM englobe trois sondes logicielles responsables de la collecte du contexte de l'utilisateur comme le montre la figure 6.3.

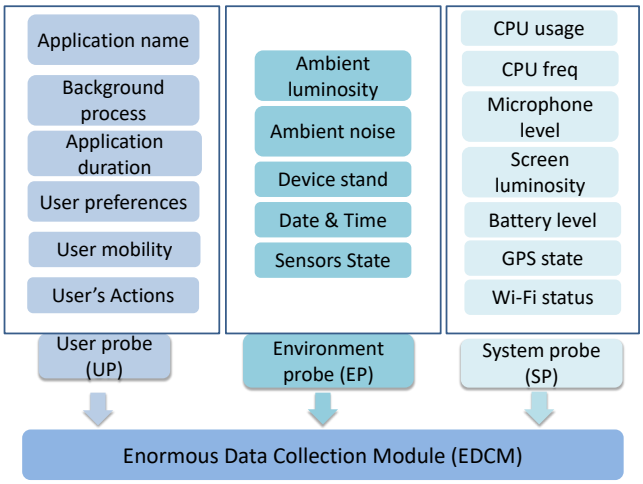

FIGURE 6.3 – Contexte de l'utilisateur collecté par le DCM

Le fonctionnement de ce module de collecte est similaire à ce qui a été présenté dans la section 5.2. Les trois sondes de l'EDCM sont:

(a) User Probe (UP): cette sonde est liée à l'utilisateur, ses préférences hardwares et aux applications exécutées pendant une période de temps définie. L'UP est déployée durant l'utilisation du système mobile et son déploiement est invisible à l'utilisateur. Cette sonde indique les applications qui sont exécutées en premier plan, les applications en arrière plan et le temps d'exécution moyen de chaque application. Elle indique également le degrés de mobilité de l'utilisateur et son type d'activité comme par exemple, immobile, entrain de marcher ou encore utilisateur entrain de courir. Cette sonde nous permet également de consolider notre supposition sur les différences entre les utilisateurs. En effet, nous supposons que les utilisateurs ont des comportements différents entre les jours de semaine et les weekends et également entre les différents moments d'une même journée. Ces différences résident principalement dans le fait que les utilisateurs n'utilisent pas les mêmes applications et n'ont pas les mêmes préférences hardwares. Ces préférences peuvent varier également en fonction du moment de la journée et le jour de semaine. Pour un salarié d'entreprise par exemple, les applications ludiques ne sont pas supposées être lancées pendant les horaires de travail. L'utilisation des applications, leur nature et leur temps d'utilisation est porteur d'informations sur l'utilisateur, son comportement, son style de vie, etc. Dans un premier temps, nous avons fixé la période de collecte à deux semaines comme dans ANBOC. Néanmoins cette période est modifiable et ajustable selon nos besoins.

(b) Environment Probe (EP): cette sonde est responsable de capturer les informations liées à l'environnement de l'utilisateur. Elle permet de savoir dans quelles conditions l'utilisateur se trouve à un moment T. Parmi les informations contenues dans cette sonde, nous retrouvons: le degré de lumière ambiante, le niveau du bruit ambiant, la position du système, la date et l'heure. Pour la collecte de ces informations, nous utilisons certains des capteurs embarqués illustrés dans SBOC comme le microphone et l'ALS. Cette sonde appuie également notre hypothèse sur l'impact de l'environnement de l'utilisateur sur sa consommation d'énergie. Des variations dans ces paramètres auront un impact direct sur l'utilisation du système mobile.

(c) System Probe (SP): cette dernière sonde indique les ressources consommées par le système. Nous y retrouvons des informations comme: la moyenne de la charge du processeur (%), la fréquence du processeur allouée par le système d'exploitation (MHz), le volume des hauts parleurs (%), le degré de luminosité de l'écran (%), l'état de la Wi-Fi, le niveau de la batterie ou encore l'activité des capteurs embarqués, tels que le GPS. L'évolution des ressources consommées dans le temps est porteuse d'informations sur l'intensité de l'utilisation du système. Ces informations sont également susceptibles d'apporter des éclaircissements sur les habitudes de l'utilisateur et ses besoin hardwares. Par exemple, la charge d'utilisation moyenne du CPU varie d'un utilisateur à un autre en fonction des habitudes. Une personne travaillant

dans un bureau sollicite probablement plus de ressources de calcul qu'une personne qui travaille dans un chantier.

Ces trois sondes ne sont pas exploitées séparément mais sont combinées. Les informations collectées sont stockées comme suit:

- l'évolution de toutes les informations collectées est corrélée aux données temporelles spécifiées dans l'EP;
- pour chaque application exécutée en premier plan, nous stockons toutes les informations disponibles dans les sondes;
- Lorsque l'utilisateur change d'application, le processus est réitéré des informations est réitéré;
- À la fin de la campagne de collecte de données, nous aurons pour chaque application, l'évolution du contexte de l'utilisateur correspondant comme le montre la figure 6.4.

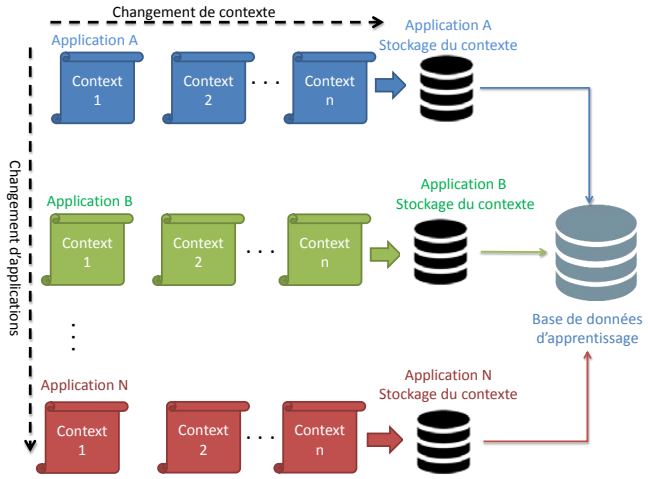

FIGURE 6.4 – Changement du contexte de l'utilisateur en fonction de l'application en premier plan

L'EDCM est le premier module exécuté par Enormous. Ce module est exécuté en background et se lance à la mise sous tension du système mobile. Ces données sont collectées pendant deux semaines consécutives pour créer une base de données d'apprentissage (Learning Database) comme montré dans la figure 6.4. Le contenu de cette base sera consommé par le prochain module Enormous Data Processing Module (EDPM). Toutes ces informations sont collectées via l'interfaçage des sondes susmentionnées avec les capteurs embarqués et notamment les APIs exposées par l'OS telles que *GetFouregroundWindow ()*. Cette API permet d'avoir un point d'entrée pour la collecte du nom des applications en premier plan. *GetBatteryLife ()* est une autre API qui permet d'avoir le niveau de batterie courant du système. Le format de stockage choisi est le format XML. Nous avons choisi ce format pour sa modularité et les facilités d'extension qu'il permet. De plus, L'IECSDK a été optimisé pour tirer profit des données XML via un parseur qui lui est dédié.

Comme cela va être développé dans la section 6.6, le surcout engendré par l'EDCM est relativement faible. La collecte de données est réalisée en

mode *offline* et est exécutée une seule fois, excepté dans le cas où l'utilisateur n'est pas satisfait des politiques d'optimisation réalisées. Lorsque les optimisations impactent négativement la satisfaction de l'utilisateur, la phase de collecte est réitérée pour enrichir la base d'apprentissage. Ce procédé est similaire à celui d'ANBOC.

2. EDPM: le second module d'Enormous représente la contribution majeure de ce framework et assure la phase de traitement dans le modèle CPA. L'EDPM contient un nombre de classifieurs *ad-hoc* implémentés en techniques d'apprentissage supervisé. Chaque classifieur est dédié à un composant hardware. Ces algorithmes d'apprentissage sont responsables du traitement des informations du contexte de l'utilisateur fournis par l'EDCM, pour générer les politiques d'optimisations. Ce module sera présenté en détail dans la section 6.3.
3. Power Policies (PP): les sorties des classifieurs de l'EDPM sont générées sous formes de règles de configuration hardware. Ces règles sont appliquées aux différents composants ciblés. Par exemple, pour la gestion de la luminosité de l'écran, la règle est sous forme d'un intervalle contenant les degrés de luminosité les plus appropriés au contexte courant de l'utilisateur. Imaginons un utilisateur se trouvant dans un environnement sombre et utilisant une application qui ne requiert pas une forte luminosité, la règle peut être sous la forme de l'intervalle [0%-25%]. Cet intervalle indique qu'il n'est pas nécessaire d'aller au delà des 25% de luminosité pour satisfaire l'utilisateur et que la luminosité optimale pour ce contexte se trouve entre 0 et 25%.
4. Current State Module (CSM): ce module fournit en temps réel des informations sur l'utilisation du système à un instant T. Cette action est réalisée avant la configuration hardware. Cela sert à abroger tout effet négatif sur la satisfaction de l'utilisateur en désactivant ou en baissant des ressources qui seraient utilisées à cet instant. Le CSM nous permet à grande échelle d'améliorer la précision des optimisations. Les informations fournies par le CSM sont principalement les applications exécutées et les ressources utilisées par le système comme la connectivité Wi-Fi, l'intensité de la luminosité ou encore la mobilité de l'utilisateur. Dans le cas où l'utilisateur est entrain de charger un fichier sur le cloud par exemple et que la sortie du classifieur indique que le contexte de l'utilisateur n'a pas besoin de Wi-Fi, Le CSM nous permet de ne pas désactiver la connectivité et d'éviter un désagrément de l'utilisateur.
5. Enormous Optimizer Actuator (EOA): le principe de ce module est le même que celui de l'OA présenté dans la section 5.5. Ce module est responsable de l'ajustement des ressources en manageant les configurations des sous-composants matériels. L'objectif d'Enormous étant de gérer toutes les ressources matériels susceptibles d'engendrer une économie de l'énergie consommée. Dans cette contribution, nous nous sommes intéressés principalement à la luminosité de l'écran, la fréquence du processeur, le volume des hauts parleurs et l'état de l'interface Wi-Fi. Ces quatre ressources représentent plus de 60% de la consommation totale du système. Pour réaliser cette action, l'EOA prend en compte les politiques d'optimisations PP et les informations du système en temps réel fournis par le CSM.

6. User Satisfaction Check (USC): une action cruciale dans cette contribution est la vérification de la satisfaction de l'utilisateur. Après l'ajustement des différentes ressources et la configuration du hardware, la satisfaction de l'utilisateur est vérifiée en analysant son comportement avec deux mécanismes complémentaires.
   (a) Si l'utilisateur désactive la gestion proposée, l'EOA est notifié pour modifier la politique d'optimisation responsable de cette insatisfaction.
   (b) Nous avons également mis en place des *hooks* liés au clavier et à l'écran tactile. Durant l'utilisation du système mobile, ces *hooks* sont responsables de collecter la vitesse d'interaction entre l'utilisateur et son système, ainsi que le nombre de cliques et de *touchscreen*. Lorsque ce nombre varie anormalement après un ajustement de ressource, nous en concluons un changement dans le comportement de l'utilisateur synonyme d'insatisfaction.
   (c) Un exemple de non satisfaction de l'utilisateur:
      i. si un nouveau degré de luminosité est alloué par l'EOA, l'USC vérifie que cela a été accepté par l'utilisateur.
      ii. si l'utilisateur augmente la luminosité après cette configuration automatique, le contexte courant de l'utilisateur est stocké avec le niveau de luminosité sélectionné par l'utilisateur. Cette valeur sera prise en compte lors du prochain contexte similaire.

   L'étude de la satisfaction de l'utilisateur est principalement basée sur les actions de ce dernier. Son comportement à la suite d'une optimisation est considéré comme une métrique de satisfaction. Enormous a été conçu pour être une solution entièrement automatisée qui ne nécessite pas l'intervention manuelle de l'utilisateur. La vérification de la satisfaction de l'utilisateur nous permet également d'améliorer la précision des sorties de nos classifieurs en ajustant les valeurs en fonction des *feedbacks* de l'utilisateur. Ces deux mécanismes sont similaires à ceux d'ANBOC.

Nous rappelons que tous les modules présentés communiquent entre eux pour aboutir aux motivations de cette contribution. Dans la section 6.3 suivante, nous présentons le module EDPM et l'architecture des classifieurs qu'il encapsule.

6.3 Classifiation des ressources via l'apprentissage supervisé

Le contexte de l'utilisateur est en changement perpétuel. Ces changements sont dûs à l'utilisateur lui même, son style de vie, les variations dans son environnement, l'heure, la date, etc. En fonction du contexte, l'utilisateur et les applications peuvent nécessiter moins de ressources que celles allouées par le système d'exploitation, comme présenté dans le chapitre dédié à ANBOC. Prenons un exemple minimaliste d'un utilisateur qui finit le travail à 18H tous les jours, au travail il est connecté sur le réseau Wi-Fi de son entreprise, en sortant du travail, il n'a pas le réflexe de désactiver son interface Wi-Fi. L'antenne Wi-Fi est activée, elle consomme de l'énergie mais ne sert strictement à rien. D'autre

part, les utilisateurs se comporteront différemment en fonction de ces changements de contexte. Pour le même contexte, deux utilisateurs distincts utilisant les mêmes applications, peuvent avoir besoin de niveaux de ressources différents.

Le but du module EDPM est de comprendre et assimiler ces changements dans les besoins de l'utilisateur. Cette compréhension permet de dégager les instants et les cas de figures où il est possible de configurer à la baisse certaines ressources, sans altérer la satisfaction de l'utilisateur. Tout ce processus de compréhension est réalisé via la classification du contexte de l'utilisateur, classification dont le comportement de l'utilisateur est l'épicentre. Comme susmentionné, les classifieurs sont implémentés dans l'EDPM et ont pour objectif:

- la détection des variations dans le contexte de l'utilisateur.
- l'exploitation de ces variations pour la calibration des exigences de l'utilisateur final en matière de configuration hardware.
- La génération des politiques adaptatives de configuration pour chaque composant matériel.

La classification du contexte de l'utilisateur se fait en fonction des quatre ressources ciblées. Nous aurons donc quatre classifieurs, chacun d'eux est dédié à une ressource hardware spécifique.

- Un classifieur pour les besoins en puissance de calcul (CPU).
- Un classifieur dédié aux besoins du volume des hauts-parleurs
- Un classifieur dédié à la luminosité de l'écran.
- Un classifieur ciblant les besoins en connectivité via l'interface Wi-Fi.

Parmi les différentes méthodes de classification existantes [69], notre choix s'est porté sur les réseaux de neurones artificiels (Artificial Neural Networks (ANNs)). Ce choix a été motivé principalement par le type de données qui forment le contexte de l'utilisateur et par le caractère incertain de l'utilisateur [33]. En effet, cerner les besoins de l'utilisateur, déduire du sens de son comportement et pouvoir corréler ce comportement à ses besoins en ressources hardwares, s'avère être une tâche difficile. Pour répondre à cela, les réseaux de neurones sont caractérisés par leur capacité à déduire du sens et de la signification à partir de données complexes. Cette caractéristique des ANNs peut être exploitée pour l'extraction de patterns d'utilisation et la détection de tendances qui sont trop complexes à notifier par les humains ou d'autres techniques d'apprentissage automatique.

L'utilisation d'un ANN est réalisée en deux phases essentielles, la première est la phase d'apprentissage, qui permet la corrélation des données en entrée avec données en sortie, comme cela va être détaillé dans les sections suivantes. Après la phase d'apprentissage, un réseau de neurones peut être considéré comme un expert dans la catégorie d'informations qui analysées. Dans notre cas, ces informations représentent la globalité du contexte de l'utilisateur. Cet expert est utilisé pour fournir des projections sur des nouveaux contextes qui n'ont pas été étudiés, ce qui nous permet de répondre aux questions suivantes:

- Que faire si l'utilisateur se trouve dans un nouveau contexte ?
- Quelle est la meilleure configuration hardware pour l'utilisateur à un moment donné ?

Nous justifions également le choix des réseaux de neurones par le fait qu'ils offrent un apprentissage adaptatif, des opérations en temps réel et une auto-organisation. Ajoutons à cela que les réseaux de neurones peuvent être implé-

mentés en parallèles ou dans le cloud. Nous avons également vu émergé ces dernières années de nouveaux composants hardwares embarqués dans les systèmes mobiles qui sont spécialement dédiés à l'exécution des réseaux de neurones, comme avec le smartphone *Huawei Mate 10*. Dans la section 6.3.1, nous présentons les différentes entrées et sorties de chaque classifieur.

6.3.1 Classification des ressources

La classification des ressources doit nous fournir pour chaque contexte de l'utilisateur une configuration hardware appropriée pour la ressource ciblée. Toutes les informations fournies par l'EDCM constituent les point d'entrée de nos classifeurs.

- Données de l'utilisateur: les préférences des utilisateurs en matière de ressources, les catégories des applications lancées, le nombre des applications lancées et la mobilité de l'utilisateur.
- Données de l'environnement: la lumière ambiante et le bruit ambiant.
- Données du système: la fréquence du CPU allouée, le ratio par rapport à la fréquence maximale du CPU et les débits entrant/sortant de la Wi-Fi.

Nous considérons ces ressources comme étant indépendantes les unes les autres dans la classification. La figure 6.5 illustre nos quatre classifieurs.

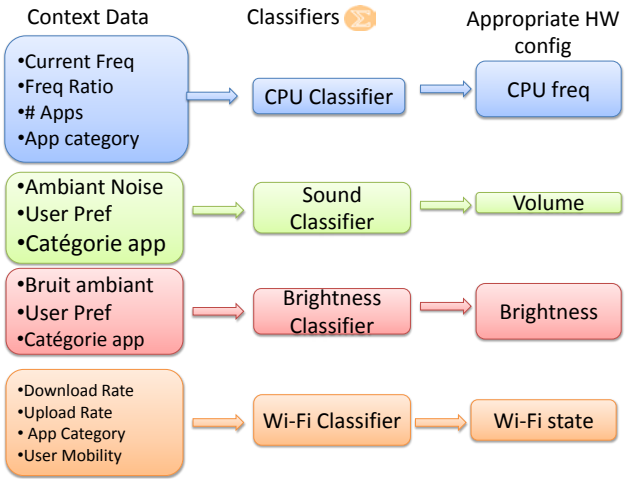

FIGURE 6.5 – Classifieurs Ad-Hoc

Nous commençons par présenter le classifieur dédié aux besoins de calcul.

6.3.1.1 Classification du contexte selon les besoins de calcul

Le classifieur CPU prend en entrée les besoins de l'application et les préférences de l'utilisateur en matière de puissance de calcul. Ces données sont:

- *Current CPU freq*: cette entrée est triviale et représente la fréquence du processeur allouée par le système d'exploitation pendant un contexte donné.
- *Freq Ratio*: ce paramètre représente le ratio de la fréquence allouée par l'OS par rapport à la fréquence maximale du processeur. Quand ce ra-

tio est élevé, cela indique que l'utilisateur a tendance à nécessiter une fréquence CPU élevée et vice versa.
- # *App*: ce paramètre désigne le nombre d'application exécutée en premier plan ainsi que les applications en arrière plan. Cette entrée a été choisie pour trouver une corrélation entre le nombre d'applications exécutées et la fréquence CPU la plus appropriée au contexte de l'utilisateur.
- *App Category*: cette entrée représente la catégorie de l'application exécutée en premier plan, nous avons réalisé une classification spécifique des applications en trois catégories distinctes, en se basant sur leurs besoins comme cela va être expliqué dans le tableau 6.1. La catégorie de l'application est utilisée pour déterminer les besoins de l'application en puissance de calcul.

Les sorties de ce classifieur représentent les bornes supérieures de la fréquence CPU la plus appropriée, de manière similaire à ANBOC. Ces sorties sont:
- Classe cpu_1: cette classe désigne les contextes nécessitant une fabile fréquence CPU (en dessous de 800 MHz).
- Classe cpu_2: pour les contextes qui requièrent une fréquence CPU moyenne qui se situe entre 800 MHz et 1.25GHz.
- Classe cpu_3: les contextes qui requièrent une forte puissance de calcul sont affectés à cette classe. La fréquence y est comprise entre 1.25 GHz et 1.75 GHz.
- Classe cpu_4: cette classe désigne les contextes dont le besoin en fréquence CPU est supérieur à 1.75 GHz.

6.3.1.2 Classification du contexte de l'utilisateur en fonction du besoin en volume sonore

Pour la classification du contexte en fonction du volume, les entrées sont:
- *Ambient Noise*: ce paramètre indique le niveau du bruit ambiant de l'environnement et est obtenu par le capteur Ambient Noise Sensor (ANS) présenté dans le chapitre dédié à SBOC. Le volume des hauts parleurs est inversement proportionnel à celui du bruit ambiant.
- *User Pref*: cette entrée représente la moyenne des différents volumes des hauts parleurs qui ont été sélectionnés par l'utilisateur. Cette moyenne est corrélée à l'heure de la journée et le jour de la semaine.
- *App Category*: la catégorie de l'application dans ce classifeur sert à déterminer les besoins du volume des applications. Par exemple, le volume requis par *Spotify* est très différent de celui de *Tumblr*.

La sortie de ce classifieur est le volume le plus approprié parmi les classes suivantes:
- Classe $sound_1$: pour les contextes nécessitant un volume très bas variant de 0% à 25%.
- Classe $sound_2$: pour les contextes où le volume des hauts parleurs est situé entre 25% et 50%.
- Classe $sound_3$: cette classe est destinée aux contextes qui requièrent un volume élevé se situant entre 50% et 75%.
- Classe $sound_4$: cette dernière classe comprend les contextes utilisateurs dont le besoin en volume est très élevé et est compris entre 75% et 100%.

Étant donné l'absence de références à cette problématique, le choix de ces classes a été réalisé de façon intuitive et peut être raffiné par la suite.

6.3.1.3 Classification du contexte selon les besoins en luminosité

Les entrées de ce classifieur sont:

- *Ambient Lux*: cette entrée représente la lumière ambiante et est présente dans toutes les méthodes de gestion de la luminosité. Par exemple, dans le Samsung Galaxy S7, l'ajustement de la luminosité est basé uniquement sur la lumière ambiante de l'environnement [103]. Cette donnée est capturée via le microphone, présenté précédemment dans SBOC.
- *User Pref*: ce paramètre est basé sur le même principe que celui qui a été présenté pour le volume. Cette entrée peut donner également des indications sur la vue de l'utilisateur.
- *App Category*: tout comme pour le volume, ce paramètre nous sert à faire le lien entre les applications exécutées et le niveau de luminosité qu'elles requièrent.

La sortie de ce classifieur est la luminosité de l'écran la plus appropriée parmi les quatre classes suivantes:

- Classe $bright_1$: pour les contextes nécessitant une luminosité très basse dont la valeur varie entre 0% et 25%.
- Classe $bright_2$: pour les contextes dont la luminosité est entre 25% et 50%.
- Classe $bright_3$: cette classe est destinée aux contextes qui requièrent une luminosité élevée se situant entre 50% et 75%.
- Classe $bright_4$: cette dernière classe comprend les contextes utilisateurs dont le besoin en luminosité est très élevé et est compris entre 75% et 100%.

6.3.1.4 Classification du contexte selon les besoins en Wi-Fi

Pour ce classifieur nous avons 4 entrées:

- *Download Rate*: cette entrée représente le débit de téléchargement, plus précisément, la quantité des données téléchargées via la Wi-Fi en ko/s.
- *Upload Rate*: cette entrée représente les débit d'envoi, comme l'entrée précédente, c'est la taille des données envoyées en ko/s qui nous intéresse. Ces deux paramètres ont été vus dans la classification d'ANBOC.
- *App Category*: ce paramètre représente la classification *offline* des applications en fonction de leurs besoins en connectivité, comme cela va être illustré dans les sections suivantes.
- *Mobility*: ce paramètre indique la mobilité de l'utilisateur et nous informe de la distance de l'utilisateur à un hotspot Wi-Fi. Ce paramètre nous est retourné par le gyroscope et la boussole.

Ce classifieur nous donne en sortie l'état de la Wi-Fi correspondant au contexte, parmi les trois classes suivantes:

- Connected: cette classe désigne les contextes qui requièrent une connexion Wi-Fi.
- Disconnected: nous retrouvons dans cette classe les contextes d'utilisateur qui n'ont pas besoin de Wi-Fi, mais susceptible d'en avoir besoin, comme par exemple, un contexte où l'utilisateur est mobile et se rapproche d'un hotspot Wi-Fi.
- Off: cette dernière classe est destinée aux contextes qui ne nécessitent aucune connectivité Wi-Fi. Cela peut se traduire par les besoins de l'application ou encore la mobilité de l'utilisateur.

Comme mentionné dans les sections précédentes, toutes les entrées de nos réseaux de neurones sont capturées pendant la même période et sont stockées dans une base de données d'apprentissage par l'EDCM. Le temps de collecte est ajustable et dépend du changement dans l'application en premier plan et les données temporelles comme nous l'avons illustré dans la figure 6.4.

6.3.1.5 Catégorie des applications

Tous les classifieurs présentés ont comme paramètre d'entrée une information relative au besoin de l'application en premier plan. Ce paramètre a été introduit pour améliorer la précision des classifieurs. Cette catégorisation des applications en fonction des quatre ressources a été réalisée de manière intuitive en se basant sur le fonctionnement des applications comme cela est présenté dans le tableau 6.1.

- Pour le CPU, nous avons trois valeurs qui sont F pour *faible*, M pour *moyen* et E pour *élevé*.
- Pour la Wi-Fi, nous avons trois valeurs. 1 désigne les applications qui nécessitent une connectivité internet et 0 pour les applications qui n'en ont pas besoin. 1/0 pour les applications qui fonctionnent en mode hors ligne et en ligne comme *Google Maps* et *Spotify*.
- Pour la luminosité et le volume, nous avons également trois valeurs *F, M* et *E*.

Tableau 6.1 – Catégories des applications

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

Nos quatre classifieurs passent en premier lieu par une étape d'apprentissage pour construire un modèle prédictif. L'apprentissage utilisé dans Enormous est de type supervisé dans lequel, nous utilisons un ensemble d'instances de contextes d'utilisateurs que nous avons collecté. Nous faisons correspondre à ces contextes, des sorties désirées (souhaitées) appelées également des cibles. Le modèle résultant de cette étape d'apprentissage est utilisé pour classifier de nouvelles instances de contextes qui ne font pas parti de la base de données dédiée à l'apprentissage. La section 6.3.2 suivante détaille ce mécanisme de construction des modèles prédictifs pour nos quatre classifieurs.

6.3.2 Création des modèles prédictifs

Après avoir déterminé les paramètres en entrée et les classes en sortie de chaque classifieur, dans la section 6.3.1. L'étape qui suit est de construire la base d'apprentissage. Pour chaque classifieur de ressources, nous sélectionnons un ensemble de valeurs en entrée, ces valeurs représentent le contexte de l'utilisateur. Par expérimentation, nous sélectionnons ensuite les sorties désirées correspondantes. Cette construction est schématisée dans la figure 6.6.

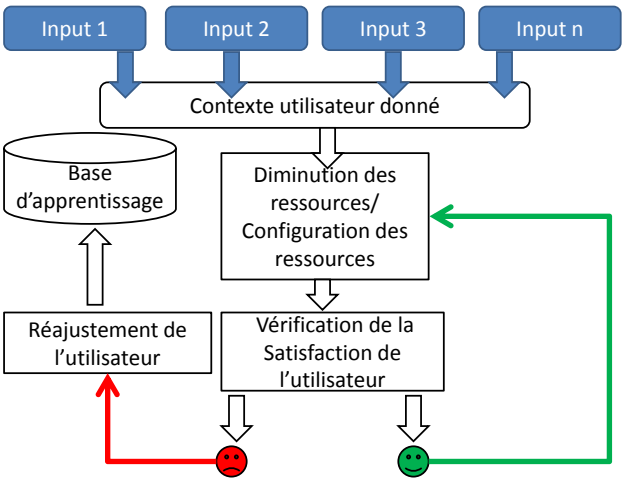

FIGURE 6.6 – Processus de sélection des sorties souhaitées des classifieurs

- Pour la luminosité et le volume, nous avons en entrée des valeurs données pour les paramètres sélectionnés précécédemment dans la section 6.3.1.3. Nous procédons ensuite à la détermination des classes en sortie via un mécanisme appelé *Change Blindness* [121] également utilisé dans le fonctionnement d'ANBOC. Nous rappelons que ce principe a été révélé par des chercheurs en psychologie humaine. Il est basé sur l'incapacité des humains de détecter des changements importants dans leur environnement, si ces derniers se produisent de manière progressive et graduelle. Avec Enormous, nous exploitons le mécanisme du *Change Blindness* en diminuant graduellement la luminosité de l'écran et le volume des hauts parleurs, jusqu'à ce que l'utilisateur s'aperçoive de ce changement. Lorsque la luminosité de l'écran ou le volume des hauts parleurs sont diminués automatiquement et ne sont pas modifiés manuellement par l'utilisateur, nous concluons que la satisfaction de l'utilisateur n'a pas été impactée. Nous sauvegardons ensuite les niveaux de la luminosité et du volume adéquats correspondant au contexte de l'utilisateur en entrée. Si pendant la phase de diminution automatique, l'utilisateur intervient manuellement en augmentant la luminosité de l'écran ou le volume des hauts parleurs, nous concluons que l'utilisateur n'est pas satisfait de cette configuration. Nous sauvegardons dans ce cas comme sortie souhaitée les derniers niveaux de luminosité et de volume qui n'ont pas altéré la satisfaction de l'utilisateur.
- Pour le CPU, nous sauvegardons les valeurs des paramètres en entrée

puis nous baissons également la fréquence jusqu'à la détection d'un comportement anormal de l'utilisateur. Le comportement anormal peut être défini comme l'arrêt manuel de l'application en premier plan, des appuis incessants sur l'écran, redémarrage de l'application, actualisation de l'application ou de la page web consultée. Tout ce comportement est détecté via le module USC. Nous stockons ensuite la dernière valeur de la fréquence CPU qui a satisfait l'utilisateur.

- Concernant l'élaboration des sorties souhaitées pour la classification Wi-Fi, nous sauvegardons les entrées du classifieur pour lesquelles nous souhaitons assigner les sorties correspondantes. Ensuite par expérimentation, nous activons et désactivons la connectivité en analysant les retours de l'utilisateur. Par exemple, à 18H en jour de semaine, nous désactivons la connectivité Wi-Fi en supposant que l'utilisateur est en mobilité et n'est pas à proximité d'un hotspot Wi-Fi. Si l'utilisateur réagit en réactivant la connectivité, nous affectons aux entrées enregistrées, la classe *connected*. Ce processus représente l'unique phase où l'utilisateur est appelé implicitement à participer à l'élaboration de la base d'apprentissage.

Après avoir explicité la manière avec laquelle nous construisons notre base d'apprentissage avec le processus de sélection des sorties cibles. Nous présentons dans la partie 6.4 suivante l'architecture, la structure et les configurations de nos réseaux de neurones.

6.4 Architecture de nos réseaux de neurones

Pour réaliser notre classification, nous avons utilisé le perceptron multicouche - MLP [97]. Un réseau MLP est basé sur l'apprentissage supervisé qui a fait ses preuves en matière d'efficacité. Ce réseau de neurones appartient à la famille des réseaux à propagation avant où l'information se propage dans un sens unique des entrées vers les sorties. Les réseaux MLP se composent de trois couches qui sont: une couche d'entrées, une ou plusieurs couches intermédiaires de traitement et une couche pour les sorties souhaitées. Chaque couche contient un nombre de neurones artificiels. Ces trois couches sont connectées entre elles via leurs neurones respectifs. La construction d'un réseau MLP se fait en deux étapes principales:

1. Définition de l'architecture du réseau de neurones.
2. Apprentissage du réseau de neurones - *Learning*.

6.4.1 Définition de l'architecture de nos MLP

Nous utilisons les symboles présentés dans le tableau 6.2 pour exprimer les différents paramètres utilisés dans nos MLP.

Tableau 6.2 – Formalisme utilisé pour nos MLP

Symbole	Description
$a_{n,m}$	La nième entrée du neurone m
o_m	La sortie du neurone m
x_m	L'agrégation du neurone m
$w_{n,m}$	Le poids de la liaison entre deux neurones n et m
f	La fonction d'activation des neurones
e_k	L'erreur de la K^{me} sortie du réseau
σ_m	Le signal d'erreur du neurone m
θ_m	Le biais du neurone m
I	Le nombre de neurones dans la couche des entrées
J	Le nombre de neurones dans la couche des sorties
K	Le nombre de neurones dans la couche des sorties

Définir l'architecture d'un réseau de neurones MLP consiste à déterminer la structure des différentes couches, le nombre de neurones contenus dans chaque couche, les fonctions d'activation des neurones et le format des données en entrée et en sortie. Les réseaux de neurones d'Enormous sont formés par:

- La taille de la couches des entrées: nous avons une couche en entrée dont le nombre de neurones représente le nombre de paramètres du contexte de l'utilisateur pour chaque ressource. Nous en dénombrons quatre pour les réseaux dédiés au CPU et à la Wi-Fi et trois pour les réseaux de la luminosité et du volume. Ces entrées ont été présentées dans la section 6.3.1.
- Le nombre et la taille des couches de traitement: le choix du nombre et de la taille des couches intermédiares est une tâche qui impacte fortement les performances du MLP. À ce jour, il n'y a pas de méthodes exactes qui permettant de choisir un nombre optimal pour ces couches intermédiaires [16]. Pour statuer sur le nombre de neurones contenus dans chaque couche intermédiaire, la solution la plus préconisée est empirique. Cette solution nécessite la réalisation d'expérimentations et de comparaisons entre les différents taux d'erreurs obtenus à la fin de la phase d'apprentissage.
- La taille de la couches en sortie: le nombre de neurones contenus dans la couche de sortie représente les différentes classes de nos classifieurs. Nous avons quatre neurones pour les réseaux dédiés au CPU, volume et luminosité. Pour la classification de la Wi-Fi, nous avons une couche en sortie composée de 3 neurones.
- Initialisation des poids entre les différents neurones: les poids des liaisons entre les différents neurones représentent un paramètre qui influe sur la précision du résultat final de la classification. Afin de maximiser les performances de nos réseaux de neurones, nous initialisons les différents poids par une valeur comprise entre -1 et 1. Les poids reliant les entrées à la couche intermédiaires sont initialisés à 1.

- Fonction d'activation des neurones: il existe de nombreuses fonctions d'activation [62]. Ces fonctions déterminent la sortie de chaque neurone en se basant sur ses entrées. En suivant le modèle d'un réseau de neurones MLP, nous utilisons la fonction *sigmoid* [63].

Nous utilisons un codage binaire pour les résultats de la classification. Les neurones de la couche de sortie sont soit à 0, soit à 1. Le nombre de neurones N est déterminé par logarithme binaire du nombre des classes possible selon l'équation suivante.

$$N = \begin{cases} N & SiY = 2^n \\ log_2(n) + 1 & Sinon. \end{cases}$$

Nous utilisons la notation $I - -J - -K$ pour désigner un réseau MLP avec I neurones en entrée, J neurones dans la couche de traitement et K neurones pour la couche en sortie.

6.4.2 Notation utilisée pour la représentation de nos MLP

Dans nos réseaux MLP, chaque neurone est considéré comme une structure composée d'une agrégation (x) et une valeur représentant sa sortie (o), noté neurone $= (x, o)$. La sortie du neurone est exprimée via la fonction d'activation $o = f(x)$. Une couche est représentée avec:

1. Un ensemble de neurones.
2. Un vecteur comportant les entrées.
3. Un vecteur de biais émanant des neurones des couches précédentes.
4. Une matrice des poids de liaison avec la couche précédente (w).
5. Un vecteur de sortie.

La matrice des poids est de taille: $m * n$, avec m, le nombre de neurones et n la taille du vecteur en entrée. La case (i, j) de cette matrice contient le poids de la liaison entre le neurone j de la couche précédente et le neurone i de la couche courante. Dans la figure 6.7, nous avons un réseau MLP 3 – 2 – 1, le biais et la sortie de chaque neurone n sont respectivement θ_n et O_n.

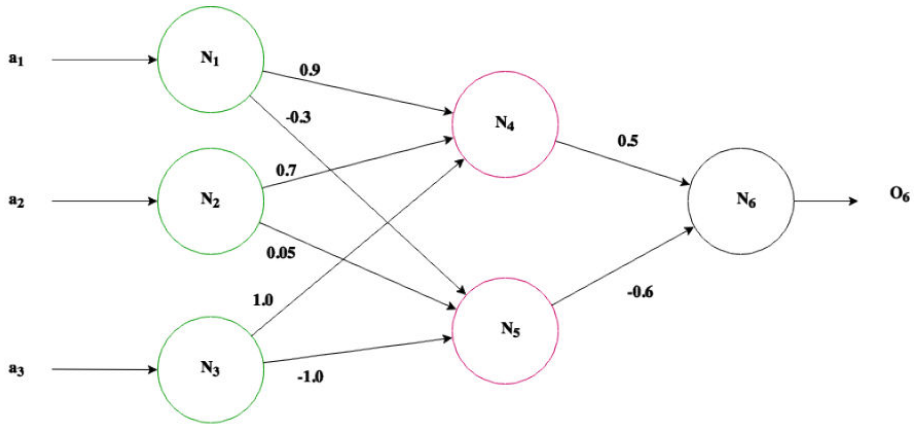

FIGURE 6.7 – MLP 3 – 2 – 1

Le tableau 6.3 représente le codage du réseau MLP précédent. Les entrées de la couche de traitement représentent les sorties de la couche des entrées, de même, les sorties de la couche de traitement représentent les entrées de la couche des sorties.

Tableau 6.3 – Codage du réseau MLP 3 – 2 – 1

Neurones	Entrées	Biais	W	Sorties
Couche d'entrées				
$\{N_1, N_2, N_3\}$	$\{a_1, a_2, a_3\}$	$\{\theta_1, \theta_2, \theta_3\}$	1 0 0 0 1 0 0 0 1	$\{O_1, O_2, O_3\}$
Couche de traitement				
$\{N_4, N_5\}$	$\{O_1, O_2, O_3\}$	$\{\theta_4, \theta_5\}$	0.9 0.7 1.0 -0.3 0.05 -1	$\{O_4, O_5\}$
Couche de sorties				
$\{N_6\}$	$\{O_4, O_5\}$	$\{\theta_6\}$	0.5 -0.6	$\{O_6\}$

Après avoir défini l'encodage de nos MLP, nous présentons dans la section 6.5 notre algorithme d'apprentissage.

6.5 Algorithme d'apprentissage propagation du gradient

Tous les MLP implémentés dans Enormous sont basés sur l'algorithme de propagation du gradient (Back Propagation Learning Algorithm (BPLA)) [25]. L'algorithme BPLA est parmi les algorithmes qui ont été les plus étudiés et les plus utilisés pour l'apprentissage des réseaux de neurones artificiels.

Ces algorithmes ont été introduits par Rumelhart et al. Dans le processus d'apprentissage, le BPLA utilise l'algorithme du gradient pour ajuster le poids des connexions de nos réseaux de neurones. Le BPLA agit en deux phases, la propagation de l'information et la rétropropagation de l'erreur, comme l'indique les deux figures 6.8 et 6.9.

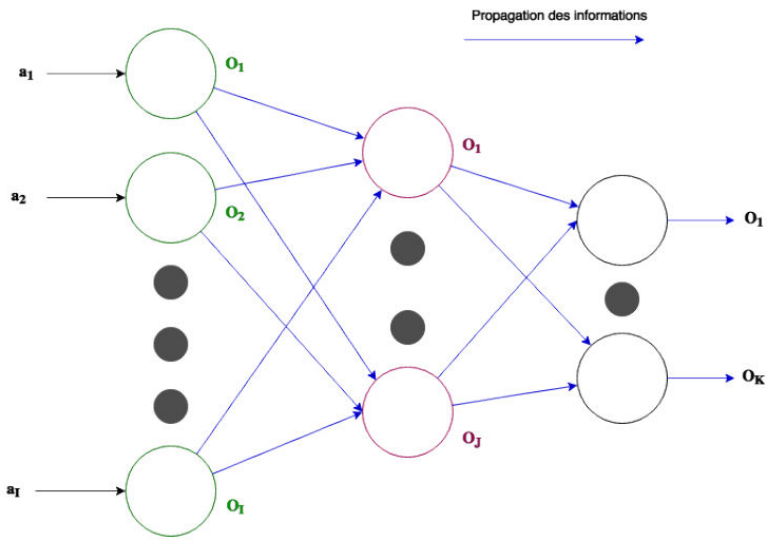

FIGURE 6.8 – Propagation de l'information

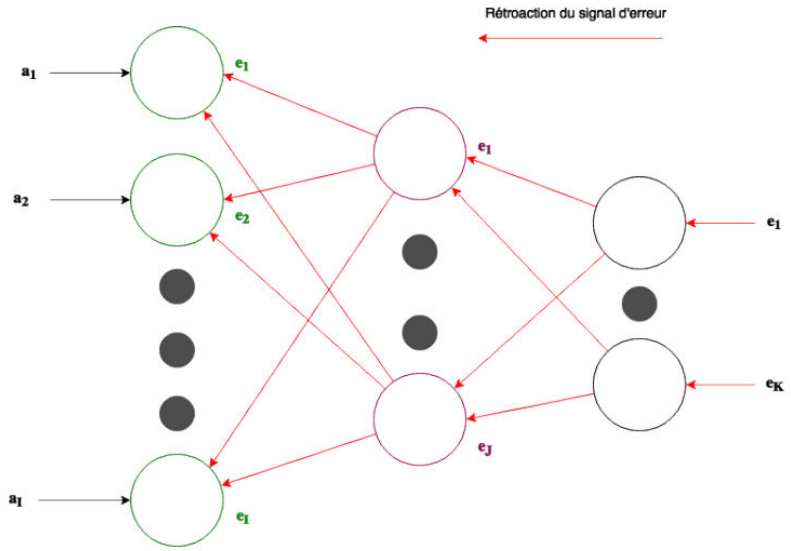

FIGURE 6.9 – Retropropagation de l'erreur

La sortie de chaque neurone représente le produit de l'agrégation des neurones précédents et les poids correspondants. Les valeurs des entrées sont converties en signaux de sorties, en calculant les fonctions d'activation. Les différences entre ces sorties et les sorties désirées (obtenues via le processus présenté dans 6.3.2) forment des erreurs qui sont corrigées via la rétro-propagation. Les poids du réseau sont alors changés en fonction des erreurs.

En réitérant cette étape plusieurs fois, l'erreur tend à diminuer et le réseau offre une meilleure classification. Les combinaisons des poids qui minimisent la

fonction d'erreur sont considérées comme la solution de notre problème d'apprentissage. En d'autres termes, le problème d'apprentissage consiste à trouver une combinaison de poids optimale dans le but de trouver une approximation aux contextes de l'utilisateur qui n'ont pas été classifiés.

Avant de commencer l'apprentissage, nous réalisons trois étapes de prétraitement:

- **Étape 1 - Normalisation des données**: les données en entrée qui forment le contexte de l'utilisateur sont normalisées dans un intervalle [0,1].
- **Étape 2 - Paramétrage du réseau**: nous choisissons le nombre des couches intermédiaires et la fonction d'activation par expérimentation. Comme mentionné dans les sections précédentes, nous utilisons la fonction *Sigmoid*:

$$f(x) = \frac{1}{(1 + e^{-x})}$$

  qui est la fonction de transfert d'erreur et dont la valeur se situe dans l'intervalle [0,1].
- **Étape 3**: charger les données d'apprentissage avec les entrées et les sorties désirées.

Après avoir chargé les données d'apprentissages avec les entrées $a_{1,\ldots n}$ et les sorties désirées $d_{1,\ldots n}$, les informations sont propagées entre les couches, en commençant par la couche des entrées. Chaque neurone calcule son agrégation selon la relation 6.1, avec l représentant le nombre de neurones de la couche précédente.

$$x_m = \sum_{l=0}^{L} w_{l,m} * inputs[l] + \theta_m. \tag{6.1}$$

Les neurones de la même couche sont activés au même instant. Chaque neurone m calcule sa sortie $o_m = f(x_m)$, cette sortie est ensuite propagée vers la couche suivante jusqu'à atteindre la dernière couche. Après la fin de la propagation des informations, les erreurs des sorties sont calculées au niveau de la couche des sorties. Ces erreurs sont ensuite retropropagées vers les couches précédentes. nous avons E l'erreur totale du réseau définie comme suit:

$$E = \frac{1}{2} \sum_{k}^{K} e_k^2 avec : e_k = O_k - d_k \tag{6.2}$$

Avec d_k représentant la sortie désirée. L'objectif du BPLA étant de minimiser l'erreur totale E. Les poids entre les neurones changent après chaque itération t. Le taux de changement est déterminé par le taux d'apprentissage η et la dérivée de l'erreur totale par rapport au poids du neurone courant $\frac{\partial E}{\partial w}$.

$$w(t+1) = w(t) + \Delta w(t) avec : \Delta w(t) = -\eta \frac{\partial E}{\partial w} \tag{6.3}$$

Le calcul des différentes dérivées est différent d'une couche à l'autre. Le BPLA commence par les couches de sorties, ensuite les couches traitement.

- Pour la couche des sorties, nous avons:

$$\frac{\partial E}{\partial w_{j,k}} = \frac{\partial \frac{1}{2}\sum_{k'}^{K} e_{k'}^2}{\partial w_{j,k}} = \frac{1}{2}\sum_{k'}^{K} \frac{\partial e_{k'}^2}{\partial w_{j,k}} \tag{6.4}$$

En remplaçant $e_{k'}$ dans l'expression:

$$\frac{\partial e_{k'}^2}{\partial w_{j,k}} = \frac{\partial (o_{k'} - d_{k'})^2}{\partial w_{j,k}} = \frac{\partial (o_{k'} - d_{k'})^2}{\partial o_{k'}} * \frac{\partial o_{k'}}{\partial x_{k'}} * \frac{\partial x_{k'}}{\partial w_{j,k}} \tag{6.5}$$

En développant la relation:

$$\frac{\partial e_{k'}^2}{\partial w_{j,k}} = \begin{cases} 2 * e_k * f'(x_k) * o_j & Sik' = k \\ 0 & Sinon. \end{cases} \tag{6.6}$$

En remplaçant dans l'équation 6.5:

$$\frac{\partial E}{\partial w_{j,k}} = e_k * f'(x_k) * o_j; \tag{6.7}$$

Nous définissons le signal d'erreur de la couche des sorties comme suit:

$$\delta_k = e_k * f'(x_k); \tag{6.8}$$

- Pour la couche de traitement, nous avons:

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial \frac{1}{2}\sum_{k}^{K} e_k^2}{\partial w_{i,j}} = \frac{1}{2}\sum_{k}^{K} \frac{\partial e_k^2}{\partial w_{i,j}} \tag{6.9}$$

En remplaçant e_k dans l'expression :

$$\frac{\partial e_{k'}^2}{\partial w_{i,j}} = \frac{\partial (o_k - d_k)^2}{\partial w_{i,j}} = \frac{\partial (o_k - d_k)^2}{\partial o_k} * \frac{\partial o_k}{\partial x_k} * \frac{\partial x_k}{\partial o_j} * \frac{\partial o_j}{\partial x_j} * \frac{\partial x_j}{\partial w_{i,j}} \tag{6.10}$$

En développant l'équation:

$$\frac{\partial e_{k'}^2}{\partial w_{i,j}} = 2 * e_k * f'(x_k) * w_{j,k} * f'(x_j) * o_i \tag{6.11}$$

En remplaçant dans l'équation 6.9:

$$\frac{\partial E}{\partial w_{i,j}} = f'(x_j) * o_i * \sum_{k}^{K} e_k * f'(x_k) * w_{j,k} = f'(x_j) * o_i * \sum_{k}^{K} \delta_k * w_{j,k} \tag{6.12}$$

Nous définissons le signal d'erreur de la couche de traitement de la manière suivante:

$$\delta_j = f'(x_j) * \sum_{k}^{K} \delta_k * w_{j,k} \tag{6.13}$$

La valeur de changement de poids entre deux neurones m et n est définie comme suit $\Delta w_{m,n} = -\eta * \delta_m * o_n$.

Pour le biais, l'entrée est toujours égale à 1, donc: $\Delta\theta_m = -\eta * \delta_m$

En se basant sur ces différentes équations, nous présentons l'algorithme d'apprentissage utilisé. Nous utilisons un critère d'arrêt composé de deux conditions:

- Atteindre un seuil maximal de l'erreur total.
- Atteindre le nombre maximal d'itérations.

Algorithme 4 : Algorithme de BPLA

1 **Début**
2 **tant que** *Critère d'arrêt* **faire**
3 **Pour chaque** *Chaque élément de la base d'apprentissage* **faire**
4 /* Phase de propagation de l'information */
5 - Récupérer les entrées de l'élément;
6 - Propager l'information dans le MLP dans les couches I – J – K;
7 - Calculer l'erreur $e_k = o_k - d_k$
8 /* Rétropropagation de l'erreur */
9 Pour les neurones de la couche de sortie
10 - $\sigma_k = e_k * f'(x_k)$;
11 - $\partial w_{j,k} = -\eta * \sigma_k * o_j$;
12 - $\theta_k = \eta * \sigma_k$;
13 Pour les nœuds de la couche de traitement
14 - $\sigma_j = f'(x_j) * \sum k\sigma_k * w_{j,k}$;
15 - $\partial w_{i,j} = -\eta * \sigma_j * o_i$;
16 - $\theta_j = -\eta * \sigma_j$;
17 La mise à jour des poids entre les différents neurones
18 - Entre les nœuds de la couche des entrées et de traitement
19 - $w_{i,j} = w_{i,j} + \partial w_{i,j}$;
20 - $\theta_j = \theta_{i,j} + \partial\theta_{i,j}$;
21 - Entre les nœuds de la couche de traitement et la couche de sortie
22 - $w_{j,k} == w_{j,k} + \partial w_{j,k}$;
23 - $\theta_k = \theta_{j,k} + \partial\theta_{j,k}$;
24 Evaluation du critère d'arrêt

Avec Enormous, nous avons testé et comparé deux types de réseaux de neurones basés sur l'algorithme BPLA qui sont le Feed Forward Back-Propagation (FBP) et le Cascade Back-Propagation (CBP) [33] .

6.5.1 Feed Forward Back-Propagation(FBP)

Le réseau FBP est un réseau de neurones artificiels acyclique. Dans ce réseau, l'information se déplace que dans un sens unique vers l'avant. Cela se fait en allant des couches d'entrée vers les couches de sorties en passant par les couches intermédiaires. Il n'y a pas de boucle ou de cycle dans ce type de réseau de neurones. La figure 6.10 montre le réseau de neurones FBP responsable de la classification du contexte en matière de volume. Nous avons trois neurones dans la couche d'entrée, 10 neurones formant la couche intermédiaire et une couche de sortie avec quatre neurones. La couche de sortie correspond aux différentes classes du classifieur du volume.

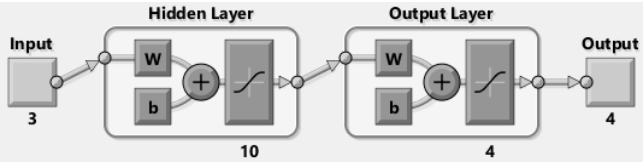

FIGURE 6.10 – Feed Forward Back-Propagation dédié au volume

6.5.2 Cascade Back-Propagation (CBP)

Le réseau CBP [44] est un réseau de neurones artificiels cyclique, contrairement au FBP. Les réseaux CBP incluent une connexion entre les entrées vers toutes les couches et de chaque couche à sa couche successive. Prenons l'exemple d'un réseau composé de trois couches. Nous aurons ainsi une connexion entre la couche 1 vers la couche 2, de la couche 2 vers la couche 3 et de la couche 1 vers la couche 3. Ce réseau aura également une connexion allant des entrées vers les trois couches. La figure 6.12 illustre ce type de réseau pour la classification du contexte selon les besoins en luminosité.

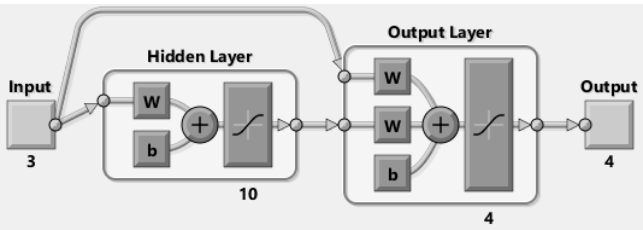

FIGURE 6.11 – Cascade Back Propagation dédié à la luminosité

6.5.3 Paramétrage de nos MLP

La phase d'apprentissage de nos réseaux de neurones a été réalisée avec 2000 échantillons de contextes utilisateurs pour chaque ressource et chaque utilisateur. Le nombre maximal d'itérations a été fixé à 2000. Le nombre de couche de traitement est fixé par défaut à deux. Pour chaque ressource et chaque utilisateur, nous choisissons le type de MLP et l'architecture la plus adéquate.

- Augmenter le nombre des couches intermédiaires ou le nombre de leurs neurones n'est pas synonyme de précision. Au contraire, cela pénalise l'approche en augmentant la durée d'apprentissage.
- Augmenter l'échantillonage destiné à l'apprentissage peut provoquer du sur-apprentissage - *overlearning* [131]. Dans ce cas là, le réseau devient incapable de classifier les contextes qui n'appartiennent pas à la base d'apprentissage.

Après avoir réalisé la phase d'apprentissage, les poids entre les neurones sont fixés. À la fin de cette étape nous nous retrouvons avec quatre MLP experts. Chacun d'eux est dédié à une ressource spécifique et à un utilisateur donné. Ces quatre MLP sont testés ensuite avec des contextes utilisateurs non classifiés. Cette phase est appelée la phase de simulation. Nous présentons dans la figure 6.12 une comparaison entre les deux architectures CBP et FBP pour quatre classifeurs dédiés au même utilisateur. Les paramètres qui déterminent le choix de nos architectures sont l'erreur quadratique moyenne - Mean Square Error (MSE) [142] et la précision du MLP.

- L'erreur quadratique moyenne (MSE) indique la précision de notre réseau de neurones. Elle est exprimée sous la forme d'une distance *Euclidienne* entre les sorties souhaitées et les sorties réelles obtenues. Plus cette valeur est petite, plus notre réseau de neurone est précis. Elle peut être considérée comme une erreur globale que l'apprentissage tend à minimiser.
- La précision montre la relation entre les sorties du réseau de neurones et les sorties souhaitées durant la phase d'apprentissage.

Ressources	ANNs	Structure	MSE	Précision
CPU	CBP	5 -- 12 -- 4	0.234	73%
		5 -- 10 -- 4	0.123	89%
		5 -- 4 -- 4	0.325	69%
		5 -- 3 -- 4	0.453	59%
	FBP	5 -- 12 -- 4	0.211	78%
		5 -- 10 -- 4	0.312	75%
		5 -- 4 -- 4	0.411	67%
		5 -- 3 -- 4	0.190	84%

Ressources	ANNs	Structure	MSE	Précision
Bright	CBP	3-- 12 -- 4	0.404	62%
		3 -- 10 -- 4	0.147	82%
		3 -- 5 -- 4	0.126	87%
		3 -- 3 -- 4	0.248	69%
	FBP	3 -- 12 -- 4	0.231	73%
		3-- 17 -- 4	0.342	67%
		3 -- 4 -- 4	0.132	86%
		3 -- 3 -- 4	0.151	78%

Ressources	ANNs	Structure	MSE	Précision
Volume	CBP	3 -- 12 -- 4	0.227	66%
		3 -- 10 -- 4	0.197	72%
		3 -- 8 -- 4	0.253	64%
		3 -- 4 -- 4	0.267	59%
	FBP	3 -- 12 -- 4	0.173	74%
		3-- 17 -- 4	0.312	55%
		3 -- 4 -- 4	0.143	79%
		3 -- 20-- 4	0.209	68%

Ressources	ANNs	Structure	MSE	Précision
Wi-Fi	CBP	4-- 15 -- 3	0.147	83%
		4 -- 12 -- 3	0.258	76%
		4 -- 8 -- 3	0.580	65%
		4 -- 6 -- 3	0.124	91%
	FBP	4 -- 12 -- 3	0.118	92%
		4 -- 5 -- 3	0.290	74%
		4 -- 4 -- 3	0.420	69%
		4 -- 2 -- 3	0.312	71%

FIGURE 6.12 – Comparaison des architectures des ANN's pour l'utilisateur 1

Les résultats reportés dans la figure 6.12 indiquent:

- L'architecture CBP fournit de meilleurs résultats en comparaison avec le FBP pour la classification du contexte en fonction des besoins en CPU et luminosité.
- L'architecture FBP est plus précise pour les classifications du volume et la Wi-Fi.

- Le choix des architectures et leurs précision peuvent différer d'un utilisateur à un autre à cause des différentes manières d'interaction qu'ont les utilisateurs et le contexte lui même. Comme mentionné précédemment Il n'existe pas de règles fixe pour choisir l'architecture adéquate, le seul moyen réside dans l'expérimentation et l'analyse des résultats obtenus. Le tableau 6.4 nous montre les types et les architectures choisis pour nos six utilisateurs.

Tableau 6.4 – Architecture des réseaux de neurones pour tous les utilisateurs

Users	Ressources	Architecture	Structure	MSE	Précision
User 1	CPU	CBP	5 – 10 – 4	0.123	89%
	Volume	FBP	3 – 4 – 4	0.143	79%
	Brightness	FBP	3 – 5 – 4	0.126	87%
	Wi-Fi	CBP	4 – 12 – 3	0.121	92%
User 2	CPU	FBP	5 – 12 – 4	0.127	85%
	Volume	FBP	3 – 4 – 4	0.111	92%
	Brightness	FBP	3 – 5 – 4	0.128	84%
	Wi-Fi	CBP	4 – 12 – 3	0.127	85%
User 3	CPU	CBP	5 – 13– 4	0.122	90%
	Volume	CBP	3 – 10– 4	0.134	83%
	Brightness	FBP	3 – 8 – 4	0.129	85%
	Wi-Fi	CBP	4 – 1– 3	0.119	92%
User 4	CPU	CBP	5 – 9 – 4	0.123	89%
	Volume	FBP	3 – 4 – 4	0.143	88%
	Brightness	CBP	3 – 2 – 4	0.120	87%
	Wi-Fi	CBP	4 – 3 – 3	0.115	91%
User 5	CPU	CBP	5 – 11 – 4	0.211	81%
	Volume	FBP	3 – 7– 4	0.129	87%
	Brightness	FBP	3 – 8 – 4	0.143	78%
	Wi-Fi	CBP	4 – 4 – 3	0.137	82%
User 6	CPU	FBP	5 – 12 – 4	0.119	91%
	Volume	FBP	3 – 4 – 4	0.139	79%
	Brightness	FBP	3 – 3 – 4	0.132	82%
	Wi-Fi	CBP	4 – 4 – 3	0.119	92%

Après avoir montré l'élaboration et la mise en place de nos réseaux de neurones, ces derniers vont être utilisés pour générer les politiques de gestion de puissance pour la réduction d'énergie.

6.6 Résultats expérimentaux

Cette section présente les résultats expérimentaux d'Enormous. L'objectif de ces expériences est de valider l'architecture de notre framework, évaluer les réductions de consommation de puissance et d'énergie obtenues et le coût de notre solution. Cette section est subdivisée en trois parties. Nous présentons en premier lieu un rappel sur les outils utilisés durant l'implémentation et les expériences. Dans la section 6.6.2, nous présentons les résultats expérimentaux en matière de consommation de puissance et d'énergie. La dernière section présente le coût de notre solution.

6.6.1 Outils et environnement expérimental

Comme les composants précédents SBOC et ANBOC, Enormous est également basé sur l'IECSDK. Les différents modules ont été implémentés sous forme d'ILs et ALs comme suit:

- EDCM: Le module de collecte de données a été implémenté sous forme de 3 ILs. Chaque IL correspond à une sonde.
- EDPM: Le module englobant les MLP est implémenté sous forme de quatre ALs. Chaque AL correspond à un classifieur.
- CSM: Le module responsable de la récupération de données dynamiquement avant l'application des politiques d'optimisation est implémenté sous forme d'IL.
- Finalement, nous avons quatre ALs pour la configuration hardware, chaque AL est destinée à une ressource hardware donnée.

La figure 6.13 présente l'architecture basée sur l'IECSDK.

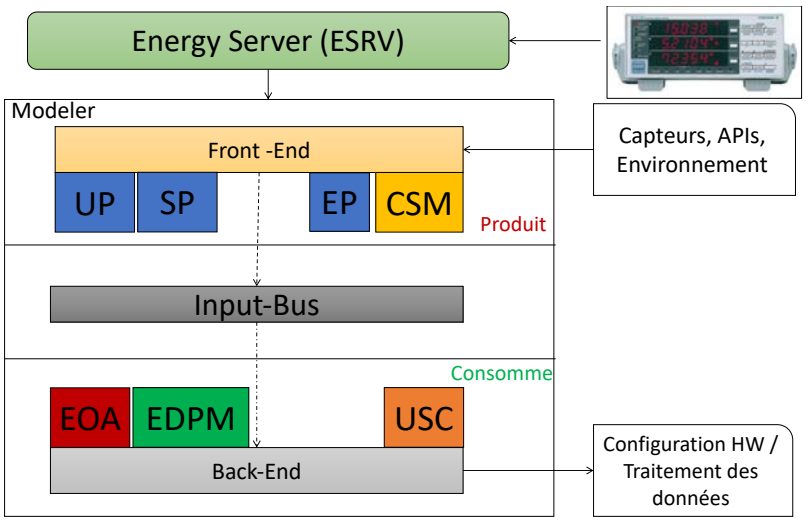

FIGURE 6.13 – Architecture d'Enormous basée sur l'IECSDK

Pour les expérimentations et le test des réseaux de neurones, nous avons utilisé la bibliothèque *Neural Network Toolbox* fournie par Matlab. Cette *tooblox* nous permet de paramétrer les différentes architectures. Après la simulation de nos réseaux de neurones, nous implémentons l'architecture correspondante

via un *Wrapper* en langage C. Ce wrapper est par la suite intégré dans l'AL qui correspond à l'EDPM. Nous présentons dans ce qui suit, les résultats de consommation de puissance obtenus avec Enormous, concernant des utilisateurs réels et des utilisateurs synthétiques. Les expérimentations ont été réalisées sous l'*Ultrabook Intel* présenté dans la section 3.

6.6.2 Résultats de la réduction de consommation de puissance avec Enormous

Dans le but de montrer l'impact d'Enormous sur la consommation de puissance, nous avons réalisé dans la section 6.6.2.1 des tests dans différents scénarios avec plusieurs contextes d'utilisateurs réels. Nous rappelons que ces contextes ont été collectés pendant deux semaines pour être classifiés. Cette classification génère les politiques d'optimisations de puissance à appliquer. Dans la section 6.6.2.2, nous avons utilisé des traces d'utilisateurs synthétiques pour montrer l'impact d'Enormous sur toute une séquence d'applications. Rappelons que les sorties de nos classifieurs correspondent à une borne supérieure de fréquence d'horloge, un intervalle de valeurs pour le volume et la luminosité, et un état pour l'interface Wi-Fi. L'EOA agit comme suit:

- L'EOA réduit graduellement la luminosité de l'écran et le volume des hauts parleurs par Δ unités chaque λ secondes jusqu'à atteindre la configuration minimale satisfaisante pour l'utilisateur. Pour ne pas impacter la satisfaction de l'utilisateur et converger le plus rapidement possible vers la configuration optimale, Δ a été fixée à 3 unités (%) et λ à 4 secondes.
- Pour la gestion de la fréquence du processeur, l'EOA diminue la fréquence via les APIs exposées par l'OS de manière similaire à ANBOC.
- Finalement pour la Wi-Fi, l'interface est directement configurée via l'API *WLAN* en fonction du besoin du contexte ciblé. Ces configurations occurrent à chaque nouveau contexte. Dans la version actuelle d'Enormous, nous n'utilisons pas la prédiction d'ANBOC.

6.6.2.1 Gestion de la consommation de puissance pour un contexte donnée

Les tableaux 6.5 et 6.6 représentent respectivement six contextes utilisateurs et les six résultats de la classification correspondants. Les utilisateurs ont été sélectionnés depuis notre entourage via la campagne de collecte réalisée au préalable. Nous retrouvons parmi ces six utilisateurs, deux doctorants, deux sportifs et deux étudiants. Pour chaque réseau de neurones, nous avons sélectionné 2000 échantillons, pour chaque jour de semaine et chaque intervalle de temps, de manière similaire à ANBOC.

Tableau 6.5 – Contextes d'utilisateurs pour six utilisateurs réels

Context data	User 1	User 2	User 3	User 4	User 5	User6
CPU Freq	1750	1250	1250	1750	800	800
# Apps	5	3	2	4	1	2
App CPU CAT	M	L	L	L	L	L
Ambient Noise	35	57	76	15	42	0
Sound user pref	50	60	80	40	53	90
App Sound Need	L	M	H	L	M	H
Ambient Luminosity	25	58	85	35	65	47
Bright user pref	48	47	70	41	52	65
App Bright Need	M	M	M	H	M	H
Download (KB)	17	27	1.7	4.2	7.7	78
Upload (KB)	18	0.0	4.3	3.2	19.2	2.4
Wi-Fi App Cat	OFF	ON	OFF	OFF	ON	ON
Mobility	Low	High	Medium	High	Low	Low
Application	2048	Facebook	VLC	Foxit Reader	OneNote	Skype

Tableau 6.6 – Résultats de la classification

Classes	User 1	User 2	User 3	User 4	User 5	User 6
CPU freq (MHz)	1250	800	800	1250	800	800
Sound level (%)	[0,25]	[25,50]	[50,75]	[0,25]	[25,50]	[75,100]
Brightness level (%)	[0,25]	[25,50]	[50,75]	[25,50]	[25,50]	[50,75]
Wi-Fi state	Discon	Discon	OFF	OFF	Discon	Connec

Nous notons des deux tableaux précédents:

- La fréquence du processeur allouée par Enormous est toujours inférieure ou égale à celle allouée par le système d'exploitation.
- Nous pouvons obtenir des résultats de classification similaires pour des contextes utilisateurs différents. Les utilisateurs 4 et 5, par exemple ont les mêmes configurations concernant le volume [25% - 50%] mais les informations de leurs contextes respectifs sont différentes. La même information est notée pour la classification de la luminosité concernant les utilisateurs 2 et 5.
- La classification Wi-Fi est fortement corrélée avec la catégorie de l'application en matière de connectivité et la mobilité de l'utilisateur. Par exemple, l'utilisateur 2 exécute *Facebook*, alors que les résultats de la classification Wi-Fi nous donne l'état *Disconnected*. Cela est dû à la mobilité de l'utilisateur et son éloignement d'un hotspot Wi-Fi.

Enormous peut déterminer une différente configuration que celle présentée dans le tableau 6.5. Si la mobilité de l'utilisateur change par exemple, la configuration Wi-Fi sera différente que celle du tableau 6.6. Les figures 6.14, 6.15 et 6.16 représentent respectivement les configurations hardwares de l'OS Vs les configurations d'Enormous pour la fréquence CPU, le volume et la luminosité.

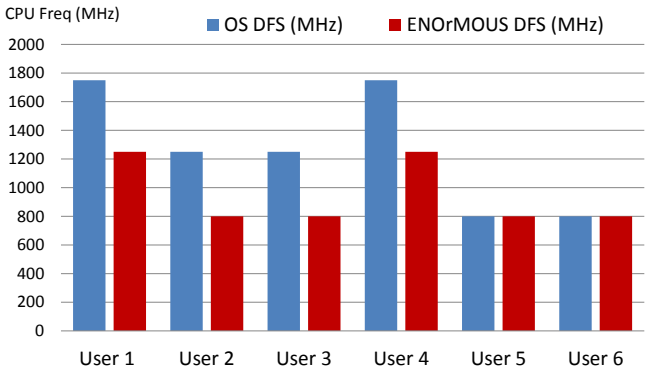

FIGURE 6.14 – Gestion de la fréquence CPU de l'OS vs Enormous

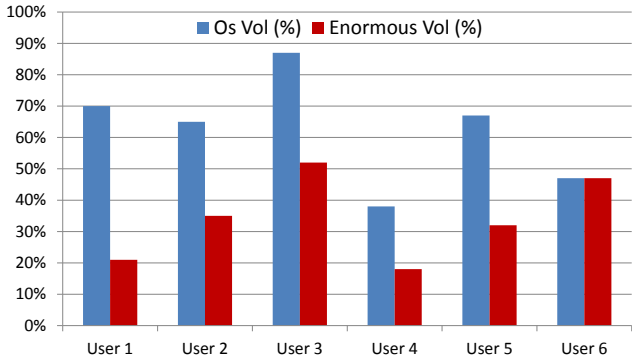

FIGURE 6.15 – Gestion du volume de l'OS Vs Enormous

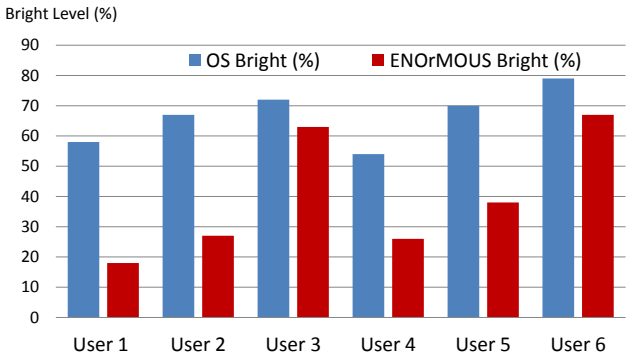

FIGURE 6.16 – Gestion de la luminosité de l'OS Vs Enormous

Le tableau 6.7 représente la gestion Wi-Fi d'Enormous comparée à celle du système d'exploitation.

Tableau 6.7 – Comparaison de la gestion de la Wi-Fi

Users	OS Wi-Fi Management	Enormous Wi-Fi Management
User 1	Connected	Disconnected
User 2	Connected	Connected
User 3	OFF	OFF
User 4	Connected	OFF
User 5	Connected	Disconnected
User 6	Connected	ON

La figure 6.17 présente les différents gains de puissance obtenus pour nos 6 utilisateurs. Les mesures ont été réalisées avec notre Yokogawa MT210 et ont été réitéré 10 fois. Une moyenne a ensuite été calculée pour être représentée. Les gains présentés varient d'un utilisateur à un autre. Le gain maximal a été obtenu avec l'utilisateur 1 et le gain minimal avec l'utilisateur 6.

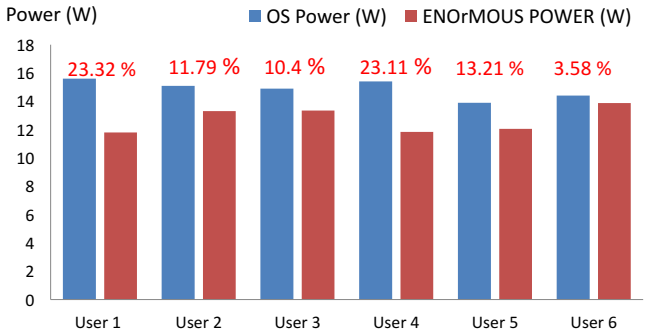

FIGURE 6.17 – Résultats de la consommation de puissance avec le déploiement d'Enormous

En analysant les résultats de la figure ci-dessus, nous notons:

- La réduction de puissance totale obtenue en manageant la fréquence du processeur est relative aux besoins de l'application, le comportement de l'utilisateur et la manière de l'utilisateur à interagir avec sa plateforme mobile.
- Le gain obtenu en baissant le volume des hauts parleurs est relativement bas et correspond à 4% de la consommation de puissance totale du système pour 100 unités de volume. Cette information a également été constatée avec SBOC.
- Le gain de puissance obtenu avec la gestion de la luminosité de l'écran est linéairement corrélé avec le niveau de luminosité sélectionné. 3% de niveau de luminosité représentent approximativement 0.155 W.
- Le gain de puissance qui résulte de la désactivation de l'interface Wi-Fi est également constant et est évalué à 1.6 W, ce qui correspond à un gain de 8%. La déconnexion du réseau sans la désactivation de l'interface génère un gain de 3.3% sur la consommation totale de la plateforme mobile.

Dans le but de détailler les réductions de la consommation énergétique obtenues avec Enormous, nous avons réalisé un nombre de tests supplémentaires.

Pour ces tests, nous avons eu recours à des utilisateurs synthétiques basés sur nos six utilisateurs réels.

6.6.2.2 Réduction de puissance avec les séquences d'applications

Dans ces tests additionnels, nous avons mis en place des scénarios d'utilisation illustrés dans la figure 6.18. Pour chaque utilisateur, nous donnons des séquences d'applications et un contexte défini. Comme mentionné précédemment, ces utilisateurs sont synthétiques et sont basés sur nos six utilisateurs réels présentés dans la section 6.6.2.1. Ces utilisateurs ont été simulés en combinant les informations collectés précédemment. L'objectif de ces résultats est de montrer le fonctionnement d'Enormous durant une utilisation normale du système, lorsque l'utilisateur est entrain d'exécuter une séquence d'applications. Ces résultats sont plus représentatifs d'une utilisation réelle en comparaison avec les résultats présentés précédemment où nous montrions les actions d'Enormous pour un contexte donné à un instant T.

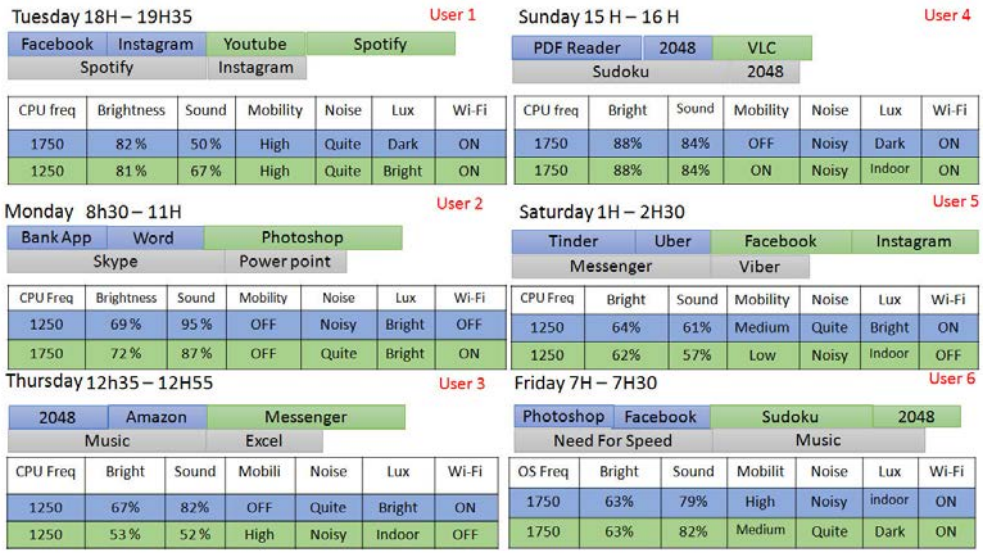

FIGURE 6.18 – Scénarios d'utilisation pour les utilisateurs synthétiques

Les informations présentées dans la figure 6.18 ont été obtenues comme suit:

- En premier lieu, nous choisissons arbitrairement six jours et six intervalles de temps pour chaque jour, ces informations représentent les données temporelles.
- Pour chaque jour, les séquences d'applications ont été obtenues en interviewant nos six utilisateurs réels sur leurs applications préférées. Les séquences d'applications peuvent être composées d'une ou deux applications. La première séquence est représentée en bleu et la seconde séquence en vert. Nous donnons également les applications en background qui sont représentées en gris.
- La fréquence du processeur allouée par le système d'exploitation, le volume des hauts parleurs, la luminosité de l'écran, et l'état de la Wi-Fi ont

été capturés dynamiquement via le CSM pour chaque séquence.

- Les préférences des utilisateurs en matière de volume et de luminosité de l'écran correspondent à celles de nos utilisateurs réels. Nous rappelons que ces préférences peuvent varier en fonction du contexte.
- Les informations de l'environnement telles que la lumière ambiante, le bruit ambiant et et les intervalles de temps sont également collectées par le CSM durant les expériences.

Les résultats de la classifications pour les utilisateurs 1, 2 et 3 sont illustrés dans le tableau 6.8.

Tableau 6.8 – Classification results for users 1,2 and 3

Users	Sequences	ENO CPU freq	ENO Vol	ENO Bright	ENO Wi-Fi
User 1	1st	1250	66 %	52 %	Discon
	2nd	/	/	/	OFF
User 2	1st	/	65 %	60 %	/
	2nd	/	/	/	OFF
User 3	1st	/	52 %	53 %	/
	2nd	/	/	43%	/

La figure 6.19 présente l'évolution de la consommation de puissance pour les utilisateurs 1,2 et 3 durant leurs séquences respectives.

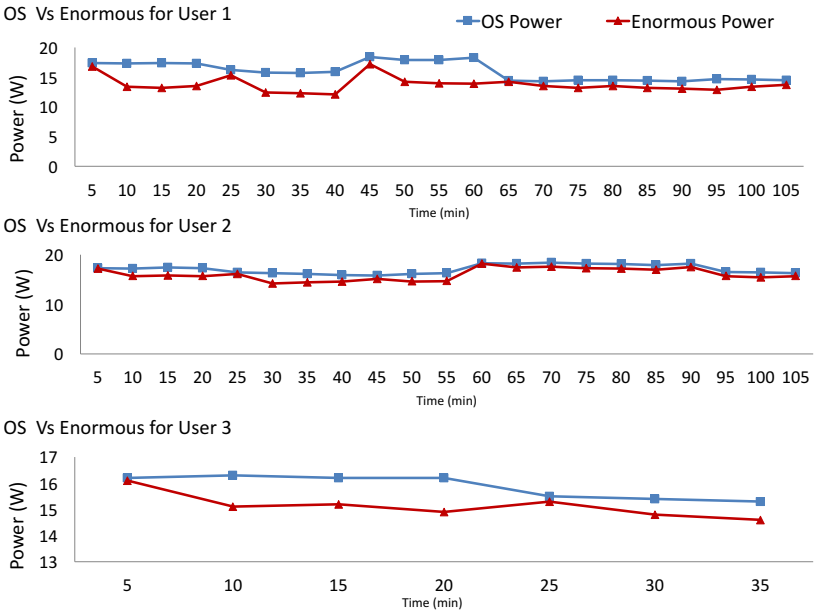

FIGURE 6.19 – Consommation de puissance des utilisateurs 1,2 et 3

Nous notons de ces résultats pour les utilisateurs 1,2 et 3 les informations suivantes:

1. L'utilisateur 1: pour la première séquence composée de *Facebook* et *Instagram*, la fréquence du processeur a été baissée de 1750 MHz à 1250 MHz. La luminosité de l'écran et le volume ont été respectivement baissés à 66% et 52%. L'interface Wi-Fi quant à elle a été déconnectée. Lorsque l'utilisateur lance *Youtube* et *Spotify*, Enormous gère uniquement la connectivité en désactivant l'interface. La désactivation de l'interface Wi-Fi malgré le type d'applications lancées s'explique par le fait que l'utilisateur a été simulé en mouvement et loin d'un hotsport Wi-Fi. La configuration du reste des composants hardwares n'a pas été changée sur la base des résultats de la classification. En analysant les résultats de la figure 6.19, nous notons que la différence entre l'OS et Enormous est minime après la 65 ème minute, ce qui correspond au lancement de *Spotify*. Nous concluons que nous avons plus d'opportunités d'économie de puissance lorsque l'utilisateur lance plusieurs applications et requiert des ressources hardwares élevées. Cette information a également été constatée durant les tests d'ANBOC.
2. L'utilisateur 2: l'impact d'Enormous sur la consommation de puissance est relativement faible. Cependant, la consommation de puissance avec le déploiement d'Enormous ne dépasse jamais la consommation de puissance du système d'exploitation seul. Pour la première séquence avec *Bank App*, *Word* et *Skype* en background, Enormous diminue le volume à 65% et la luminosité de l'écran à 60%. Lorsque l'utilisateur exécute *Photoshop* avec *PowerPoint* en background, Enormous désactive l'interface Wi-Fi car l'application ne requiert aucune connectivité.
3. L'utilisateur 3: cet utilisateur est le seul dont la consommation de puissance ne dépasse pas les 17 W. Lorsque l'utilisateur 3 lance *2048*, *Amazon* et le lecteur de musique en background, la fréquence CPU et l'état de l'interface Wi-Fi n'ont pas été changés. Enormous diminue le volume et la luminosité de l'écran respectivement à 52% et 53%. Pour la seconde séquence, Enormous réduit la luminosité de 10% supplémentaire. Le volume quant à lui, n'a pas été modifié car aucune application n'utilise cette ressource.

Les résultats de la classifications pour les utilisateurs 4,5 et 6 sont illustrés dans le tableau 6.9.

Tableau 6.9 – Classification results for users 4, 5 and 6

Users	Sequences	ENO CPU freq	ENO Vol	ENO Bright	ENO Wi-Fi
User 4	1st	800	/	38 %	OFF
	2nd	/	/	/	OFF
User 5	1st	/	/	/	Discon
	2nd	/	/	43 %	/
User 6	1st	/	/	/	Discon
	2nd	800	38 %	37 %	————

La figure 6.20 représente la consommation d'énergie pour les utilisateurs 4,5 et 6.

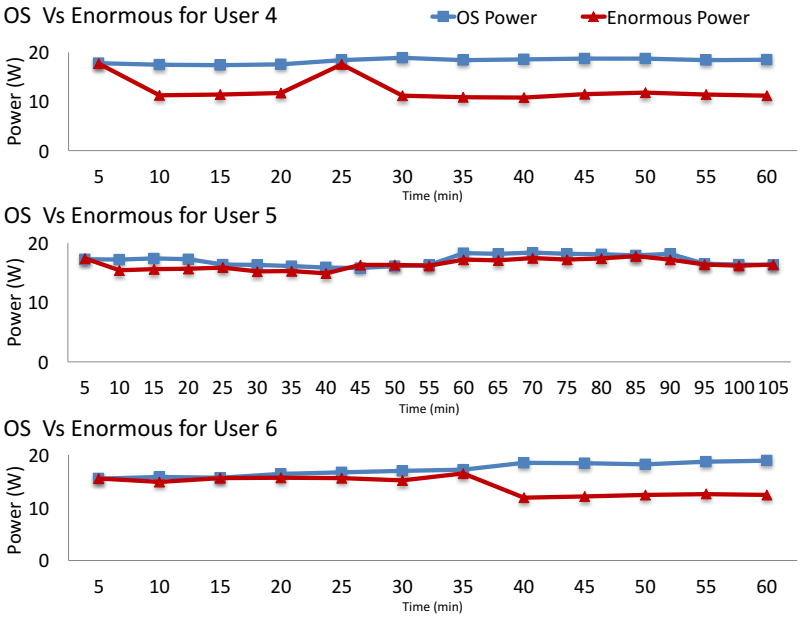

FIGURE 6.20 – Consommation de puissance des utilisateurs 4,5 et 6

Pour ces trois utilisateurs, nous notons:

1. L'utilisateur 4: cet utilisateur est celui qui présente le plus d'opportunités de réduction de puissance. Pour la première séquence, la fréquence du processeur a été baissée à 800 MHz, la connectivité Wi-Fi a été désactivée et la luminosité de l'écran a été réduite à 38 %. Pour la seconde séquence, Enormous garde les mêmes configurations en baissant d'avantage le volume.
2. L'utilisateur 5: la réduction de puissance obtenue avec Enormous est relativement faible. C'est dû principalement au contexte de l'utilisateur et les besoins de applications qui n'offrent pas des opportunités de réduction de puissance. Pour la première séquence, la plateforme est déconnectée du réseau sans être complètement désactivée à cause de la mobilité de l'utilisateur. Pour la seconde séquence, Enormous baisse la luminosité de 9%.
3. L'utilisateur 6: Pour ce utilisateur, Enormous a été très efficace comme nous pouvons le voir dans la figure 6.20. Pour la première séquence, le système mobile est déconnecté sans aucune autre configuration. Cependant, pour la seconde séquence, l'interface Wi-Fi est désactivée, le volume a été baissée à 38% et la luminosité de l'écran à 37%. La borne de la fréquence du processeur a également été baissée à sa valeur minimale qui est de 800 MHz.

Les informations communes à tous les utilisateurs qui ressortent des figure 6.19 et 6.20 sont principalement:

1. La consommation de puissance avec le déploiement d'Enormous est toujours inférieur à celle du système d'exploitation seul. Cette information

confirme notre objectif premier qui était de fournir dans le pire des cas, la même gestion d'énergie que celle réalisée par le système d'exploitation.

2. Tout changement dans le contexte de l'utilisateur impacte la consommation de puissance de la plateforme mobile. En effet, nous notons par exemple qu'à chaque fois que l'utilisateur change d'application, il en résulte soit une augmentation, soit une réduction dans la consommation de puissance avec ou sans Enormous.
3. Les consommation de puissance avec et sans Enormous illustrées respectivement en rouge et en bleu suivent le même schéma d'évolution. En effet, lorsque que nous avons une augmentation ou une réduction dans la consommation de puissance lorsque Enormous n'est pas déployé, le même changement opère dans la consommation de puissance avec Enormous.

La figure 6.21 illustre les gains obtenus avec Enormous pour les six utilisateurs. Pour chaque utilisateur, nous donnons les résultats de la réduction énergétique en (%) pour les deux séquences, ainsi que pour tout le scénario.

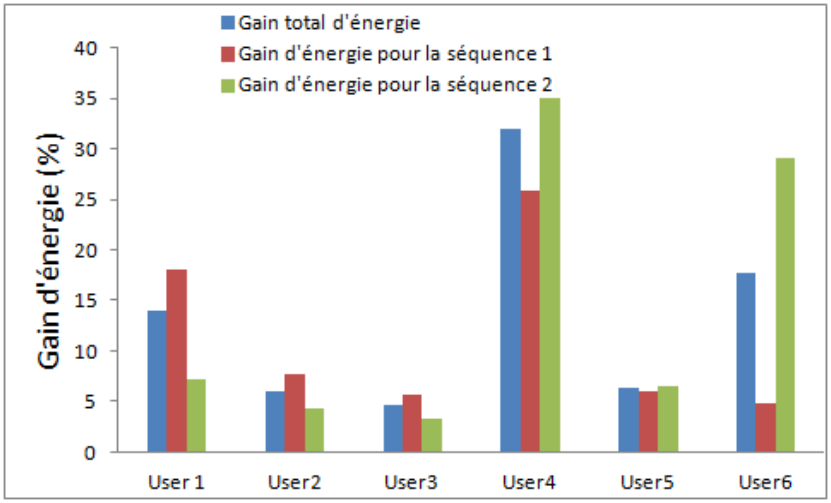

FIGURE 6.21 – Synthèse des gains énergétique sur les 6 scénarios

La réduction énergétique la plus importante obtenue est celle de l'utilisateur 4, pour qui la consommation d'énergie a été réduite de 25.9% pour la première séquence et 35.10 % pour la seconde séquence. La réduction totale pour l'ensemble du scénario est de 32 %.

Pour l'utilisateur 6, les configurations hardwares proposées par Enormous offrent une réduction d'énergie relativement élevée de 17.68 % pour tout le scénario. Nous remarquons avec cet utilisateur que la gain d'énergie obtenu sur la première séquence est très faible et est évalué à 4.75%. Pour la seconde séquence, Enormous nous offre un gain à hauteur de 29.1%. Ceci confirme notre hypothèse de départ qui mettait le contexte de l'utilisateur au cœur des opportunités de réduction de la consommation d'énergie. En effet, pour le même utilisateur, nous avons différentes consommations en fonction de son contexte.

Pour l'utilisateur 1, Enormous réduit la consommation d'énergie de 13.29% sur tout le scénario d'utilisation. Pour les utilisateurs 2,3 et 5, la consomma-

tion d'énergie est réduite respectivement de 6%, 4.60% et 6.28%. Avec ces trois utilisateurs, les opportunités de réduction d'énergie n'étaient pas nombreuses, étant donné les applications exécutées et les ressources sollicitées.

6.6.3 Coût d'Enormous

Dans cette partie, nous présentons le coût total d'Enormous en matière de consommation de puissance, de ressources et de latence. Nous présentons dans cette partie le coût des différents modules d'Enormous.

6.6.3.1 Collecte de données

La collecte des données dure deux semaines consécutives pour chaque utilisateur. La figure 6.22 montre la consommation de puissance du module EDCM.

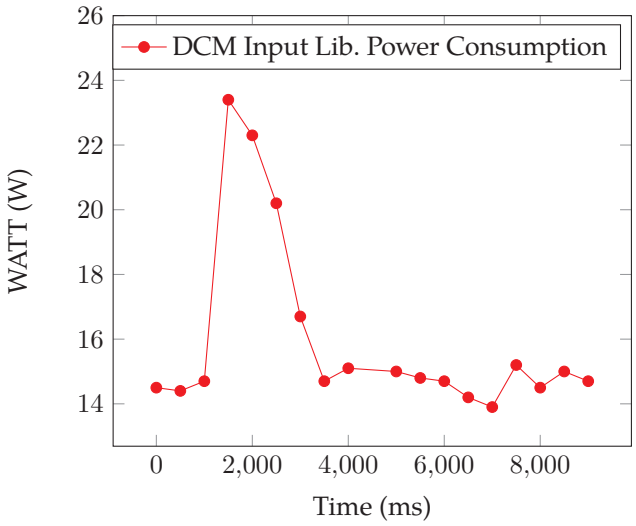

FIGURE 6.22 – Consommation de puissance de l'EDCM

Nous notons que le déploiement de l'EDCM cause un pic de puissance de 10 W pendant 3 secondes. Ces trois secondes correspondent à la durée d'interfaçage du module avec les différents capteurs. Après ces trois secondes, la consommation de puissance se stabilise et le surcout de puissance devient négligeable. Ce surcout se manifeste uniquement lors du lancement de la collecte et est commun à tous les utilisateurs.

L'espace nécessaire au stockage des données collectées varie d'un utilisateur à un autre en fonction de son comportement et son usage de la plateforme. En moyenne, la taille varie entre 22 MB et 34 MB. Comme mentionné précédemment, la collecte de donnée est réalisée qu'une seule fois et est enrichie en fonction de la précision des politiques d'optimisation. Nous présentons dans ce qui suit le cout généré par l'EDPM.

6.6.3.2 Cout de l'apprentissage

Nous nous sommes intéressés à la mesure du cout de l'apprentissage des réseaux de neurones car cela représente la phase la plus énergivore. La fréquence de l'apprentissage dépend de la phase de collecte de données. En effet, le traitement est réalisé pour la première fois après la fin des deux semaines de collecte. Ensuite, l'apprentissage est réitéré à chaque fois que la base d'apprentissage est enrichie. Les surcouts sont mesurés en fonction du CPU, de la RAM et de la consommation de puissance. Le tableau 6.10 nous montre le coût moyen de l'apprentissage des classifieurs pour nos six utilisateurs réels, avec les 2000 échantillons correspondants à chaque ressource.

Pour chaque utilisateur, nous calculons le temps moyen de l'apprentissage, l'utilisation du CPU, la consommation de la mémoire vive et le pic de puissance. Nous notons qu'il n'y a aucune phase d'apprentissage qui dure plus de 10 secondes.

Tableau 6.10 – Mesure de la phase d'apprentissage de nos classifieurs

Users	Tps Apprentissage (Sec)	Moy CPU util (%)	Moy RAM surcout (MB)	Moy Puissance surcout (W)
User 1	4.2 sec	45 %	120 MB	11 W
User 2	8 sec	39 %	109 MB	10.8 W
User 3	7 sec	43 %	112 MB	11.3 W
User 4	3.7 sec	37 %	104 MB	9.9 W
User 5	6.2 sec	44 %	117 MB	11.7 W
User 6	9.7 sec	49 %	131 MB	10.4 W

Le coût d'Enormous en matière de collecte de données et d'apprentissage permet d'embarquer la solution dans le système mobile sans passer par un serveur. Néanmoins, lorsque la taille du contexte devient plus importante, cette opération ne sera plus envisageable à cause des limitations des ressources du système mobile. Pour cette raison, nous explorons actuellement l'utilisation du *cloud* pour l'apprentissage et la simulation de nos réseaux de neurones. Cela permettra de trouver un compromis entre les réductions de consommation de puissance, la latence et les surcouts générés.

6.7 Conclusion

Dans ce chapitre, nous avons présenté une nouvelle technique de réduction de consommation de puissance par le biais d'Enormous. Avec ce framework, nous mettons l'utilisateur et son contexte au cœur des optimisations de la consommation énergétique. Comme toutes les contributions de cette thèse, Enormous obéit au même schéma CPA. Dans la phase de traitement, nous avons implémenté des réseaux de neurones *ad-hoc* pour classifier chaque contexte en fonction d'une ressource matérielle spécifique.

Nous avons également étendu l'utilisation de l'Intel Energy Checker SDK qui permet la définition de nouvelles ILs pour la collecte de données et ALs pour le contrôle des composants hardwares. Contrairement aux méthodes d'optimisation de consommation de puissance existantes, où les besoins de l'utilisa-

teur final et son comportement sont rarement pris en compte, dans cette contribution, nous exploitons un ensemble de données tirées de l'environnement, du système et de l'utilisateur lui même pour constituer le contexte de l'utilisateur. Nous nous sommes intéressés principalement à la gestion de la fréquence du processeur, la luminosité de l'écran, le volume des hauts parleurs et la connectivité Wi-Fi. Néanmoins notre approche demeure générique et applicable à tous les composants matériels comme le GPS, le GPU, ... etc.

En comparaison avec l'optimisation de la consommation énergétique proposée par le système d'exploitation Windows, Enormous offre un gain d'énergie qui atteint les 35%. Cependant, ce gain peut varier en fonction de l'utilisateur, sa manière d'interagir avec les applications, les applications exécutées et les données spatiotemporelles, de manière similaire à ce qui a été vu avec ANBOC.

À ce stade du travail, nous projetons de réduire les coûts de notre framework en utilisant des serveurs distants et en améliorant le temps de calcul de nos réseaux de neurones. Une des futures étapes de ce travail est la comparaison d'un plus grand nombre d'algorithmes d'apprentissage tout en enrichissant les informations émanant du contexte de l'utilisateur. Nous envisageons également le déploiement d'Enormous dans des différents types de plateformes pour rallonger la durée d'apprentissage et corriger les différentes configurations.

Chapitre 7

User Requests Based Optimization Component

Dans ce chapitre, nous présentons la dernière contribution réalisée dans cette thèse à travers le composant User Request Based Optimization Component (URBOC). Ce composant est basé sur Enormous et en illustre une autre utilisation. URBOC exploite également le mécanisme de prédiction des applications utilisé dans ANBOC. Le composant de ce chapitre montre que les contributions de cette thèse peuvent être combinées et sont parfaitement compatibles. Dans les contributions présentées dans les chapitres 5 et 6, nous souhaitions réaliser une approche automatisée sans l'intervention explicite de l'utilisateur.

Les composants SBOC, ANBOC et Enormous obéissent au modèle CPA. Leur principal objectif était d'aboutir à une réduction de la consommation d'énergie du système mobile, en proposant une meilleure gestion des ressources HW que celle proposée par l'OS.

À contrario, le composant présenté dans ce chapitre ne fonctionne pas exactement de la même manière et ne suit pas le modèle CPA. En effet, avec URBOC, nous essayons de faire intervenir l'utilisateur dans les optimisations. URBOC ne prend pas l'initiative de configurer une ressource HW sans être sollicité par l'utilisateur. La figure 7.1 présente une vue abstraite d'URBOC.

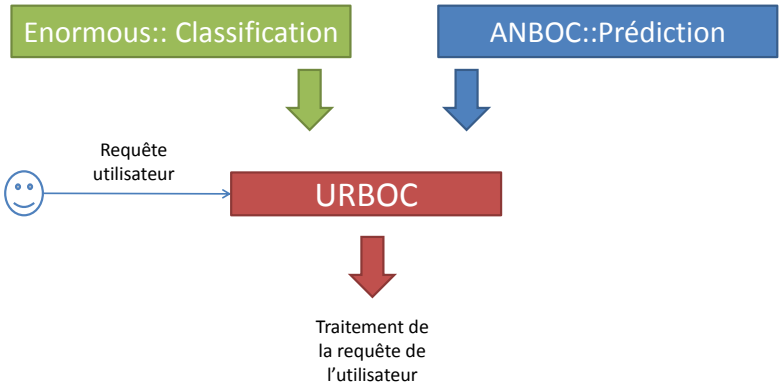

Figure 7.1 – Architecture fonctionnelle d'URBOC

Pour contextualiser l'utilité d'URBOC et son coté novateur, considérons comme point de départ un utilisateur lambda avec des contraintes critiques, l'empêchant de recharger son système mobile. Avec URBOC, cet utilisateur pourra envoyer une requête comportant jusqu'à quelle heure, il souhaite avoir la disponibilité de son système. URBOC analyse la faisabilité de cette requête en se basant sur les habitudes de l'utilisateur et les informations émanant d'Enormous.

URBOC se base également sur une base de connaissances mise à sa disposition et exploite la prédiction des applications d'ANBOC. En addition aux optimisations d'Enormous, URBOC applique différents niveaux d'optimisations hardwares dites extrêmes qui n'ont pas été utilisées dans les contributions précédentes.

Ce chapitre est organisé comme suit, nous commençons par donner l'architecture fonctionnelle et les différents modules logiciels qui interviennent dans la section 7.1. Dans la section 7.2, nous détaillons les différents traitements pour la réalisation de la requête de l'utilisateur. La section 7.3, nous présente un nombre de scénarios et d'exemples pour illustrer l'utilisation d'URBOC, suivi de la conclusion.

Sommaire

7.1 Mode de fonctionnement et modules logiciels **160**
7.2 Fonctionnement de l'UOA **163**
7.2.1 Application des HOP sans prédiction 164
7.2.2 Le rôle de la prédiction dans URBOC 165
7.3 Exemples illustratifs du fonctionnement URBOC **166**
7.3.1 Application des HOP sans l'utilisation de la prédiction 167
7.3.2 Scénario pour l'utilisation de la prédiction dans URBOC 171
7.4 Conclusion **173**

7.1 Mode de fonctionnement et modules logiciels

Avant d'introduire les différents modules logiciels d'URBOC, nous commençons par présenter son mode de fonctionnement global.

1. Le point de départ est un utilisateur ayant une contrainte spécifique qui l'empêche de mettre en charge son système mobile, sachant que ce dernier lui est indispensable.
2. L'utilisateur envoie alors une requête à URBOC avec l'heure jusqu'à laquelle il souhaite utiliser son système. Par exemple, à 14 H et avec un niveau de batterie noté $Niv_{batterie}$, l'utilisateur envoie une requête dans le but de maintenir son système sous tension jusqu'à 22 H.
3. Pour chaque contexte, URBOC configure les ressources hardwares selon la classification d'Enormous présentée dans la section 6.4.
4. URBOC vérifie ensuite la durée de vie restante de la batterie et en informe l'utilisateur.
5. Pendant l'exécution d'URBOC, nous avons deux cas de figure possibles qui dépendent de la comparaison du $Niv_{batterie}$ avec un seuil prédéterminé du niveau de batterie noté $Bat_{Threshold}$. Nous pouvons positionner par exemple $Bat_{Threshold}$ à 50%. Ces deux scénarios sont:
   (a) $Niv_{batterie} > Bat_{Threshold}$: dans ce cas de figure, nous prenons en considération uniquement le contexte courant de l'utilisateur pour les différentes optimisations HW. Cette partie est développée dans 7.2.1.

(b) $Niv_{batterie} < Bat_{Threshold}$: dans ce cas de figure, la prédiction des futures applications est utilisée. Elle est combinée aux politiques d'optimisations extrêmes . Nous utilisons la prédiction à partir de ce seuil car il devient plus difficile de satisfaire la requête de l'utilisateur au fur et à mesure que le niveau de batterie diminue. Cette partie est développée dans 7.2.2.

Dans ce qui suit, nous présentons les différents modules qui composent URBOC, comme illustré dans la figure 7.2.

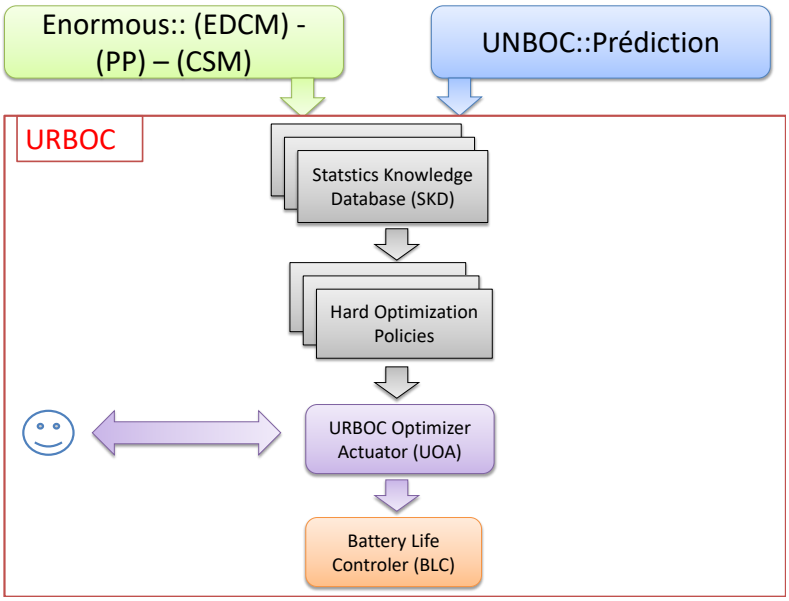

FIGURE 7.2 – Architecture modulaire d'URBOC

Les modules logiciels présentés dans la figure ci-dessus sont:

- Le mécanisme de prédiction d'ANBOC.
- EDCM, EDPM et CSM: ces modules d'Enormous ont été vus dans la section 6.2. Ils correspondent respectivement aux modules de collecte, de traitement et de récupération de l'usage du système en temps réel.
- Statistics Knowledge Database (SKD): cette base de connaissances est spécifique à URBOC et contient des informations additionnelles sur l'utilisateur et le système. La figure 7.3 résume les informations disponibles dans cette base de connaissance.

Statistics Knowledge Base (SKB)
Classement des applications selon la consommation de batterie (%)
Durée d'exécution moyenne des applications
Correspondance entre le niveau de batterie et l'heure de la journée

FIGURE 7.3 – Base de connaissances SKD

Les informations ci-dessus sont obtenues par calcul statistique:

1. Durée d'exécution moyenne des applications selon leur ordre dans une séquence d'applications donnée comme réalisé pour ANBOC.
2. Classement des applications selon leur consommation de batterie: cette information représente la quantité de batterie consommée par les applications selon leurs durées d'exécution moyenne. Par exemple: *Facebook* pour une durée de 20 minutes consomme 5% de batterie. Cette information est obtenue par mesure durant l'exécution de l'application. Après avoir réalisé ces mesures pour toutes les applications, ces dernières sont classées selon leur consommation de batterie. Ce classement, ainsi que ces différentes mesures sont susceptibles de varier d'un utilisateur à un autre.
3. Correspondance entre le niveau de la batterie et l'heure de disponibilité: cette information fournit pour un niveau de batterie $Niv_{batterie}$ l'heure jusqu'à laquelle l'utilisateur peut encore exploiter son système. Cette information a également été obtenue par calcul statistique. Pour chaque niveau de batterie de 0% à 100%, nous faisons correspondre l'heure jusqu'à laquelle le système mobile peut être disponible. Par exemple, à 50% de batterie, le système peut être utilisé jusqu'à 17H30.

- HOP: Contrairement aux PP d'Enormous, les HOP sont considérées comme des mesures d'optimisations drastiques. L'objectif est de satisfaire la contrainte temporelle de l'utilisateur en arrêtant par exemple les applications en background et en désactivant les ressources inutilisées. Nous avons trois niveaux d'optimisations illustrés dans la figure 7.4.
- URBOC Optimizer Actuator (UOA): ce module est responsable d'appliquer les optimisations HW d'Enormous et les HOP. En comparaison avec l'EOA d'Enormous, ce module offre la possibilité de désactiver les applications en arrière plan et les ressources HW qui ne sont pas utilisées. L'UOA prend également en considération la prédiction des futures applications dans le but d'anticiper la faisabilité de la requête de l'utilisateur, lorsque le niveau de batterie est inférieur au seuil prédéfini.
- Battery Life Checker (BLC): ce module est responsable de l'estimation de la durée de vie restante de la batterie en prenant en entrée l'utilisation du système en temps réel. Cette estimation se fait via une API proposée par

l'OS *GetBatteryLife ()*. Dans URBOC, la satisfaction de l'utilisateur n'est pas prise en considération de façon explicite. La priorité est de maintenir la disponibilité du système et répondre à la requête.

Hardware Optimization Policies (HOP)
1er niv: Désactivation des ressources HW non utilisées
2ème niv: Désactivation des applications en background
3ème niv: Configuration minimal de l'ensemble des ressources

FIGURE 7.4 – Politiques d'optimisations extrêmes HOP

La section 7.2 présente le fonctionnement des optimisations réalisées par l'UOA.

7.2 Fonctionnement de l'UOA

Nous présentons dans ce qui suit le fonctionnement de l'UOA basé uniquement sur Enormous. Nous détaillons ensuite comment les politiques d'optimisations HOP et le mécanisme de prédiction sont exploités pour mener à terme les actions proposées dans URBOC. La figure 7.5 ci-dessous illustre le premier traitement réalisé par l'UOA.

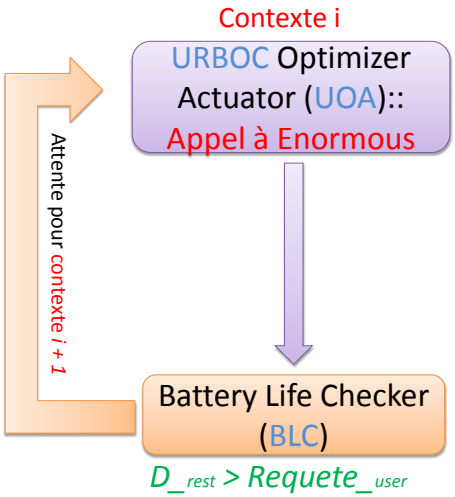

FIGURE 7.5 – Retour aux optimisation d'Enormous à chaque changement de contexte

Les étapes illustrées dans la figure ci-dessus sont les suivantes:

1. Lorsque l'utilisateur envoie une requête, l'UOA applique les politiques d'optimisations d'Enormous pour la configuration de la fréquence du processeur, le niveau du volume des hauts-parleurs, l'intensité de la luminosité de l'écran et la connectivité Wi-Fi, sur la base des résultats de la classification.

2. Le BLC vérifie ensuite la durée de vie restante de la batterie notée D_{res}. Cette vérification se fait via l'API *GetBatteryLife ()*. L'estimation du BLC est réalisée uniquement sur la base du contexte courant. Tout changement dans les applications lancées ou les ressources hardwares sollicitées affectera cette information. C'est pour cette raison que le BLC est appelé à chaque changement de contexte.
3. Si D_{res} satisfait la requête de l'utilisateur, l'UOA est mis en attente du prochain changement de contexte pour exploiter la classification d'Enormous et faire appel au BLC.

7.2.1 Application des HOP sans prédiction

Dans le cas où D_{res} ne répond pas à la requête de l'utilisateur après les configurations d'Enormous et que $Niv_{batterie} > Bat_{Threshold}$, l'UOA applique les optimisations extrêmes HOP, comme illustré dans la figure 7.6. Les HOP comportent trois niveaux d'optimisations allant du plus souple au plus extrême comme présenté dans la figure 7.4.

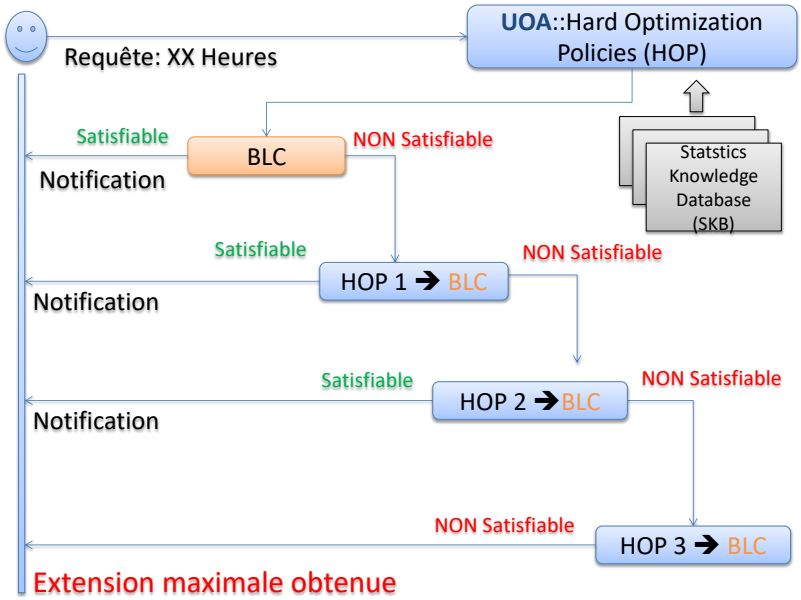

FIGURE 7.6 – Application des HOP par l'UOA

L'application des HOP obéit à l'algorithme suivant:

1. Application du 1er niveau des HOP: pour ce niveau, l'UOA commence par désactiver les ressources HW non utilisées. Le BLC est ensuite exécuté pour vérifier la durée de vie restante de la batterie.
2. Application du 2me niveau des HOP: si l'étape précédente ne répond pas à la requête de l'utilisateur, l'UOA applique le second niveau des HOP en désactivant les applications en arrière plan. Cette désactivation se base sur le classement des applications selon leur consommation de batterie. Pour réaliser cela, l'UOA se fie aux classement des applications les plus

énergivores fourni par la SKD. Le BLC est ensuite appelé pour vérifier D_{res}.

3. Application du 3me niveau des HOP: si le second niveau des HOP ne permet pas de satisfaire la requête de l'utilisateur, l'UOA applique le dernier niveau des optimisations extrêmes. L'UOA affecte aux ressources utilisées une configuration minimale.

Après chaque configuration HW, l'utilisateur est averti de la durée de vie résultante. Lorsque nous avons un changement de contexte, URBOC réitère le processus en faisant appel à la classification d'Enormous. Ce retour en arrière opère quelque soit le niveau d'optimisation atteint. Il permet également de ne pas priver l'utilisateur d'une utilisation normale de son système, lorsque cela n'impacte pas la réalisation de sa requête temporelle.

Dans le cas ou $Niv_{batterie} < Bat_{Threshold}$, l'UOA fait appel à la prédiction d'Enormous pour anticiper les actions de l'utilisateur. L'exploitation de la prédiction rend l'utilisation du système plus restrictive comme le montre la section suivante.

7.2.2 Le rôle de la prédiction dans URBOC

Avant de présenter la façon avec laquelle la prédiction est prise en considération dans URBOC, nous rappelons que nous utilisons le mécanisme de prédiction basée sur l'algorithme GSP développé pour ANBOC dans le chapitre 5. En fonction de l'application courante, nous sommes en mesure de prédire les applications qui suivront selon le nombre d'occurrence de la séquence. L'ensemble du mécanisme est détaillé dans la section 5.4.

La prédiction dans URBOC sert à surveiller le comportement de l'utilisateur. Cette surveillance se fait sur la base des applications exécutées, à partir du moment où le niveau de batterie est relativement faible. La figure 7.7 schématise le rôle de la prédiction et la communication avec les différents modules.

Contrairement aux critères de sélection du facteur K (Section 5.4.1) dont le rôle était de prédire uniquement la prochaine application qui suivra celle en cours d'exécution, dans URBOC, la longueur de séquence choisie est composée de 4 applications. Néanmoins si cette longueur n'est pas disponible, nous nous contentons de prendre la longueur maximale qui sera de $K = 2$ ou $K = 3$. Nous avons augmenté le facteur K pour pouvoir anticiper au maximum l'évolution de la durée de vie de la batterie et pouvoir prendre les dispositions nécessaires concernant les configurations HW. En effet, l'extension de la durée de vie de la batterie atteignable est proportionnelle à la longueur des séquences d'applications prédites.

Le procédé est le suivant:

1. Le mécanisme de prédiction fournit à l'UOA la séquence d'applications qui suivra l'application en cours d'exécution.
2. En se basant sur la durée d'exécution des applications prédites et leurs consommation de batterie (%), l'UOA estime le niveau de batterie à la fin de la séquence d'applications prédite.
3. L'UOA consulte ensuite la table de correspondance *Niveau de batterie(%) - Temps)* fournie par la SKD.

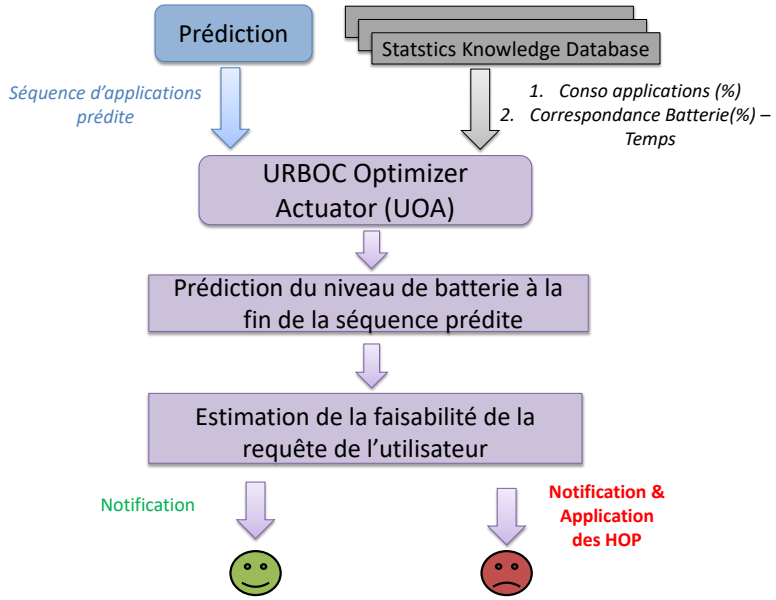

FIGURE 7.7 – Le rôle de la prédiction dans URBOC

- Si le niveau restant de la batterie prédit permet de satisfaire la requête temporelle de l'utilisateur, ce dernier est avertit des résultats de la prédiction et les estimations réalisées.
- Si le niveau de batterie prédit ne permet pas de satisfaire la requête de l'utilisateur, l'UOA applique les différentes optimisations en commençant par celles d'Enormous, en suivant le même schéma présenté dans la figure 7.6.

La prédiction sert à rallonger la durée de vie de la batterie à long terme pour les contextes à venir. Lorsque le niveau de batterie estimé à la fin des applications prédites ne satisfait pas la contrainte temporelle de l'utilisateur, nous passons au niveau d'optimisation $(Niveau_{OPT} + 1)$ en commençant par les optimisations d'Enormous jusqu'au dernier niveau des HOP. Ces optimisations opèrent avant le début présumée de la séquence d'application prédite. Ce cas est détaillé par le scénario présenté dans la section 7.3.2.

Si le 3me niveau des HOP ne satisfait pas la requête de l'utilisateur, ce dernier est informé du risque de la non faisabilité de sa requête, étant donnée son utilisation courante du système. Nous présentons dans la section suivante des scénarios pour illustrer d'avantage le traitement réalisé par URBOC.

7.3 Exemples illustratifs du fonctionnement URBOC

Par manque de temps, nous n'avons pas pu tester les fonctionnalités d'URBOC dans un environnement réel mais nous présentons dans cette section un ensemble de scénarios synthétiques pour illustrer l'utilisation d'URBOC. Ces scénarios ont été réalisés sur notre plateforme de test avec deux cas de figures. Le premier cas de figure est réalisé sans l'utilisation de la prédiction alors que le second utilise la prédiction des prochaines applications.

7.3.1 Application des HOP sans l'utilisation de la prédiction

Le tableau ci-dessous présente l'état du système mobile comportant des informations sur la batterie, l'état des ressources HW et les applications exécutées. En partant de cet état, nous montrons différents scénarios concernant les actions réalisées par URBOC. Chaque scénario correspond à une heure souhaitée par l'utilisateur.

Tableau 7.1 – Informations sur l'état actuel du système mobile

Informations sur la Batterie	Configuration HW	Application exécuté	Applications en Background
Batterie: 64% Heures restantes: 6 H	$Freq_{cpu}Max$: 1750 MHz Vol: 60% Wi-Fi: ON GPS: ON Mobile Data: ON Luminosité: 97%	Messenger	Music Player Facebook Gmail PDF Reader

Le tableau 7.2 présente les quatre scénarios basés uniquement sur l'utilisation des HOP sans la prédiction.

Tableau 7.2 – Scénarios utilisant uniquement les HOP

Scénario	Requête user	Optimisations
7.3.1.1	18H30	Enormous
7.3.1.2	20H00	Enormous et HOP niveau 1
7.3.1.3	21H30	Enormous , HOP niveau 1 et HOP niveau 2
7.3.1.4	22H30	Enormous , HOP niveau 1, HOP niveau 2 et HOP niveau 3

7.3.1.1 Application des optimisations d'Enormous

Dans le premier scénario, nous avons un utilisateur qui envoie sa requête à 11 H et souhaite que son système mobile soit utilisable jusqu'à 18H30. La durée de vie restante de la batterie est de 6 H comme indiqué dans le tableau ci-dessus. Cette valeur est obtenue via l'API *GetBatteryLife* et dépend de l'utilisation courante de la plateforme. Les six heures restantes ne permettent pas de satisfaire la requête avec l'utilisation courante. La figure 7.8 montre comment URBOC traite cette requête. Dans ce premier cas de figure, URBOC fait appel à la classification d'Enormous pour réaliser les configurations suivantes:

- Réduction de $Freq_{cpu}Max$ à 1250 MHz.
- Réduction de la luminosité de l'écran à 63%.
- L'état de l'interface Wi-Fi n'a pas été configuré car Enormous indique la nécessité de la garder active.
- Le volume a été baissé à 43%.

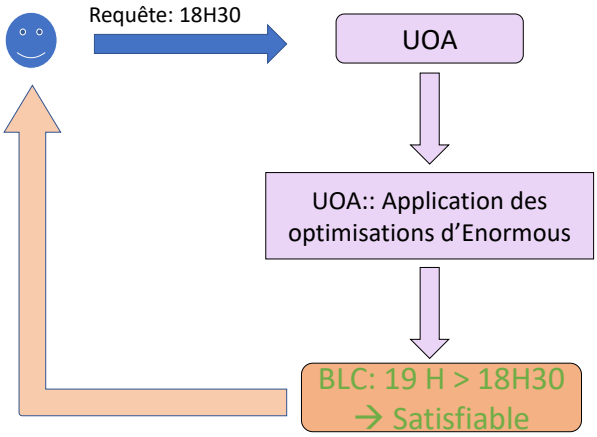

FIGURE 7.8 – Scénario 1: Applications des *Power Policies* d'Enormous sans l'utilisation des HOP

Après les configurations d'Enormous, le BLC renvoi que la disponibilité du système peut être maintenue jusqu'à 19 H, satisfaisant ainsi la requête de l'utilisateur. Pour ce contexte, nous ne faisons pas appel aux HOP.

7.3.1.2 Application du 1er niveau des HOP

Partant des mêmes informations présentées dans le tableau 7.1, le second scénario est lorsque l'utilisateur désire une disponibilité allant jusqu'à 20 H. La figure 7.9 schématise le traitement de cette requête.

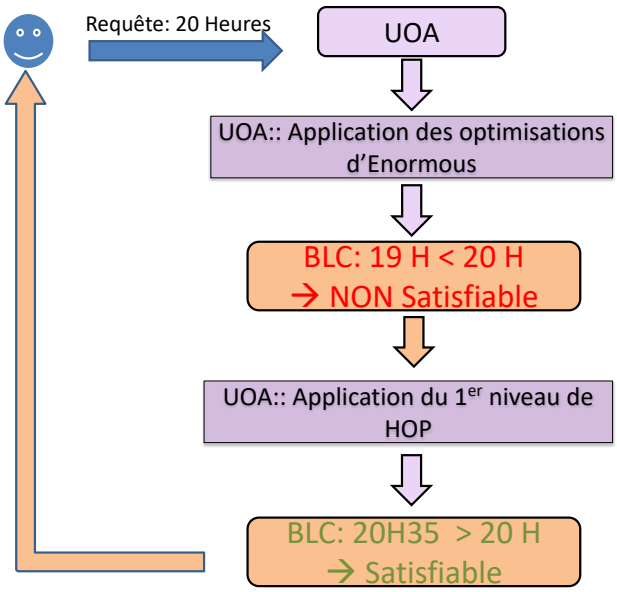

FIGURE 7.9 – Scénario 2: Utilisation du premier niveau d'optimisation de HOP

Sachant que les optimisations réalisées sur la base de la classification d'Enormous ont permis d'avoir une disponibilité jusqu'à 19H, il est nécessaire d'appliquer les HOP pour maintenir le système jusqu'à 20 H. Pour traiter cette requête le procédé est le suivant:

1. L'application des PP d'Enormous et l'appel au BLC.
2. Après l'application des PP, le niveau de batterie restant ne permet toujours pas de satisfaire les besoins de l'utilisateur.
3. L'UOA applique alors le premier niveau d'optimisations des HOP en désactivant les ressources non-utilisées. Dans ce cas de figure, le GPS est désactivé ainsi que les données mobiles car le système est connecté à la Wi-Fi.
4. Le BLC vérifie ensuite la durée de vie restante de la batterie qui satisfait la demande de l'utilisateur et lui permet d'utiliser son système jusqu'à 20 H 35.

7.3.1.3 Application du 2me niveau des HOP

Dans le scénario précédent la durée de vie de la batterie a été rallongée pour une utilisation allant jusqu'à 20 H 35. Dans ce scénario l'utilisateur souhaite pouvoir utiliser son système mobile jusqu'à 21 H 30, toujours en partant de l'état du système présenté dans le tableau précédent. Le traitement réalisé par URBOC pour satisfaire la requête de l'utilisateur est illustré dans la figure 7.10.

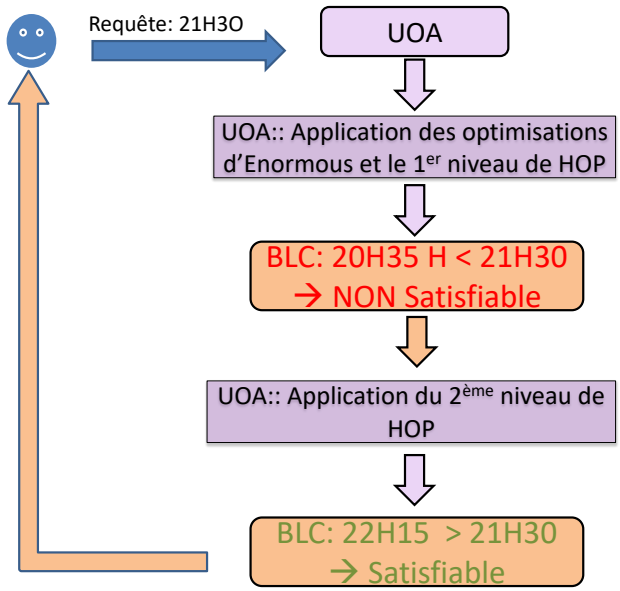

FIGURE 7.10 – Scénario 3: Application du second niveau d'optimisation HOP

L'application des politiques d'optimisations d'Enormous et le premier niveau des HOP ne permettent pas de maintenir la disponibilité du système jusqu'à 21 H 30. Dans ce cas, URBOC commence à appliquer le second niveau des HOP en désactivant toutes les applications en background selon les préférences

de l'utilisateur et le classement des applications, en fonction de leur consommation de batterie. Comme mentionné précédemment, ces données proviennent de la base SKD. Dans ce scénario, L'UOA désactive les applications *Gmail*, *PDF Reader*, ainsi que *Facebook*. Le BLC est ensuite appelé pour vérifier la durée restante de la batterie, indiquant une utilisation possible allant jusqu'à 22 H 15.

7.3.1.4 Application du 3me niveau des HOP

Dans ce dernier scénario, l'utilisateur souhaite avoir une disponibilité allant jusqu'à 22 H 30. Pour répondre à la requête de l'utilisateur, l'UOA fait appel une deuxième fois au second niveau des HOP pour désactiver les applications en background sachant que dans le scénario précédent, l'application *Music Player* n'a pas été fermée. Après cette désactivation, le BLC est appelé pour vérifier la faisabilité de la requête de l'utilisateur. Nous constatons que la fermeture de l'application ne suffit pas pour atteindre l'heure désirée par l'utilisateur. Dans ce cas là, le troisième niveau des HOP est utilisé. Ce niveau d'optimisation est le plus extrême et consiste en la configuration minimale de toutes les ressources matérielles comme suit:

- Réduction de la fréquence CPU à 800 MHz.
- Réduction du volume à 10%.
- Réduction de la luminosité à 20%.

La figure 7.11 schématise ce traitement réalisé par URBOC.

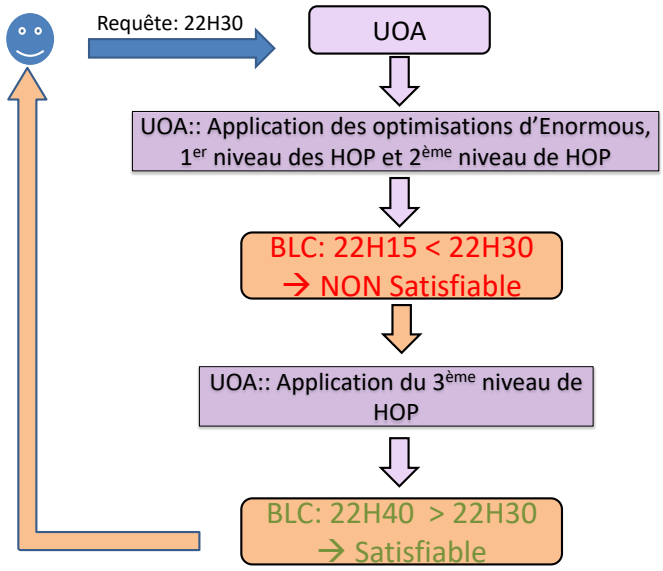

FIGURE 7.11 – Scénario 4: Application du troisième niveau d'optimisation HOP

Après ces configurations, le niveau de la batterie restant permet l'utilisation du système jusqu'à 22 H 40, ce qui satisfait la requête de l'utilisateur. Ces différents scénarios nous montrent le rôle des HOP dans le traitement et la satisfaction de la requête de l'utilisateur. Dans le cas où les configurations d'Enormous alliées à celles des HOP ne suffisent pas à maintenir l'autonomie du système jusqu'à l'heure souhaitée par l'utilisateur, ce dernier est notifié.

7.3.2 Scénario pour l'utilisation de la prédiction dans URBOC

Dans cette section, nous présentons un scénario où l'envoi de la requête se fait lorsque le niveau de batterie est inférieur à 50%. Le tableau 7.3 présente les informations sur l'état du système au moment de l'envoie de la requête. Nous indiquons l'heure d'envoi, le niveau de batterie, les ressources HW utilisées, l'application en cours d'exécution, les applications en arrière plan et l'heure jusqu'à laquelle l'utilisateur souhaite utiliser son système sans le recharger.

Tableau 7.3 – Informations sur l'état actuel du système

Heure	Batt (%)	Ressources HW	Application exécuté	Apps en Background	Requête User
18 H	49%	CPU freq: 1250 MHz Vol: 60% Wi-Fi: ON Bluetooth; ON Mobile Data: ON Luminosité: 84 %	PDF Reader	Instagram Facebook Hotmail	23 H

Le tableau 7.4 présente les résultats de la prédiction pour $K = 2$, les applications qui suivront *PDF Reader* sont *2048* et *VLC*.

Tableau 7.4 – Prédiction des applications

Application en cours (i)	Application (i +1)	Application (i + 2)
PDF Reader	*2048*	*VLC*

Le tableau 7.5 présente les informations fournies par la SKD concernant la durée d'exécution des applications prédites, ainsi qu'une estimation de leur consommation de batterie.

Tableau 7.5 – Informations de la SKD

Applications	Durée d'exécution (mins)	Batterie consommée (%)
2048	20 minutes	5 %
VLC	45 minutes	9%

Le tableau 7.6 suivant résume les scénarios de cette partie.

Tableau 7.6 – Scénarios utilisant la prédiction

Scénarios	Requête user	Optimisations
7.3.2.1	22H15	Prediction et Enormous
7.3.2.2	23H	Prédiction, Enormous, HOP Niveau 1 et HOP niveau 2

7.3.2.1 Scénario 1 utilisant la prédiction

Dans le premier scénario, avec un niveau de batterie de 49%, l'utilisateur demande une disponibilité allant jusqu'à 22 H 15 . Pour satisfaire cette requête, URBOC applique les traitements illustrés dans la figure 7.12.

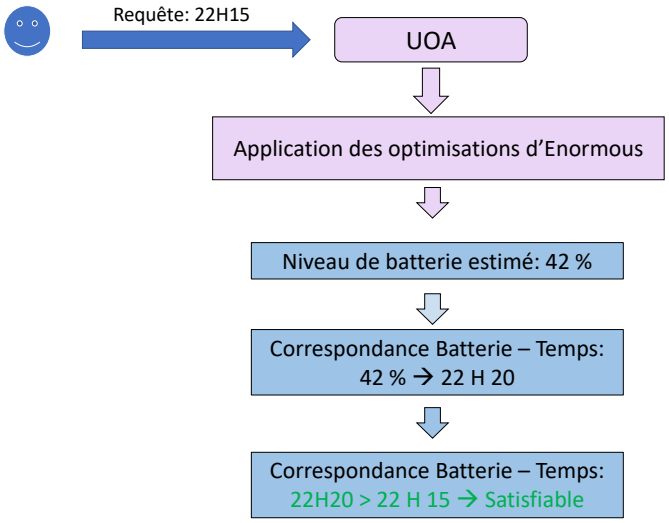

FIGURE 7.12 – Scénario 4: Application du troisième niveau d'optimisation HOP

Ces traitements sont:

1. L'UOA applique les optimisations d'Enormous et estime le niveau de batterie restant à 42% après la fin d'exécution de *2048* et *VLC*. Sans Enormous, le niveau de batterie restant après l'exécution des applications prédites aurait été de 35%.
2. Les 42% de batterie restantes permettent une utilisation allant jusqu'à 22 H 20 selon la table de correspondance Batterie - Temps, permettant ainsi de répondre à la requête de l'utilisateur.

Dans ce premier scénario, nous avons utilisé la prédiction et les optimisations d'Enormous sans recourir aux optimisations HOP.

7.3.2.2 Scénario 2 utilisant la prédiction

Le second scénario, l'utilisateur demande une autonomie allant jusqu'à 23 H. Le traitement réalisé par URBOC est illustré dans la figure ci-dessous.

Les optimisations d'Enormous ont permis d'avoir une disponibilité allant jusqu'à 22 H 20, ce qui ne satisfait pas la requête de l'utilisateur. Dans ce cas, l'UOA fait appel au premier niveau des HOP en désactivant le bluetooth et les données mobiles. La désactivation de ces ressources augmente l'estimation du niveau de batterie restant à 45%, correspondant à une utilisation allant jusqu'à 22 H 55. Ce niveau d'optimisations ne permet toujours pas de répondre aux attentes de l'utilisateur.

Nous procédons alors au second niveau d'optimisations des HOP en désactivant les applications en background qui sont *Instagram Hotmail* et *Facebook*.

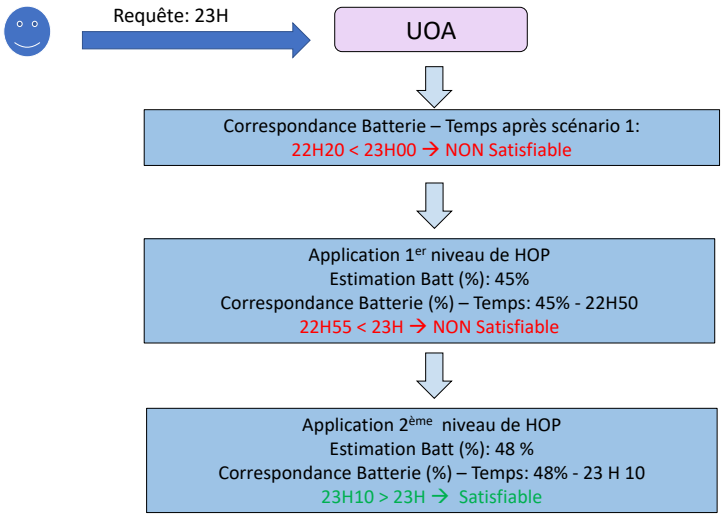

FIGURE 7.13 – Scénario 4: Application du troisième niveau d'optimisation HOP

Avec ce second niveau de HOP, le niveau de batterie restant après l'exécution des applications prédites est estimé à 48%. Selon les données de la SKD, cela permet une utilisation du système allant jusqu'à 23 H 10 satisfaisant ainsi la requête de l'utilisateur.

7.4 Conclusion

Nous avons présenté dans ce chapitre la dernière contribution de cette thèse à travers la proposition du composant URBOC. Ce composant logiciel est basé sur Enormous et sur le mécanisme de prédiction d'ANBOC. D'autres modules spécifiques à URBOC ont également été proposés. Avec URBOC, nous avons proposé une nouvelle fonctionnalité qui n'existe dans aucune plateforme mobile actuelle. Cette fonctionnalité permet à un utilisateur ayant des contraintes temporelles l'empêchant de recharger son système mobile, de rallonger l'autonomie du système jusqu'à l'heure désirée. Pour traiter la requête de l'utilisateur, URBOC utilise:

- Les informations sur les besoins de l'utilisateur et ses habitudes émanant d'Enormous.
- Le mécanisme de prédiction réalisé dans ANBOC.
- Les informations sur la durée de vie restante de la batterie.
- Les données statistiques de la SKD.

URBOC réalise également des configurations extrêmes (HOP) pour aboutir à la satisfaction de la requête de l'utilisateur. Contrairement aux approches précédentes, nous ne prenons pas en compte le comportement de l'utilisateur lors de l'application de ces politiques d'optimisations extrêmes. Nous partons du principe que la priorité est de maintenir la disponibilité du système jusqu'à l'heure souhaitée par l'utilisateur.

Nous avons également présenté un nombre de scénarios pour expliciter les

différentes phases qui permettent d'aboutir à la satisfaction de la contrainte temporelle de l'utilisateur. Avec URBOC, nous montrons que toutes les solutions proposées sont modulaires et peuvent être combinées pour permettre une meilleure gestion de la consommation d'énergie que celle proposée par les OS.

Par manque de temps, à ce stade de développement, nous n'avons pas déployé les fonctionnalités d'URBOC dans un environnement réel mais les scénarios présentés montrent qu'il est possible d'ajouter ce composant dans les OS actuels, pour fournir cette fonctionnalité à l'utilisateur.

Troisième partie

Conclusion et perspectives

Chapitre 8

Conclusion générale et perspectives

L'optimisation de la consommation d'énergie dans les systèmes mobiles est un objectif important qui concerne différentes entités, allant des grandes firmes de l'informatique mobile jusqu'aux utilisateurs finaux, en passant par les acteurs de la recherche académique et industrielle. Une gestion intelligente d'énergie permet d'étendre la durée de vie de la batterie, impacte la fiabilité du système et présente d'importantes retombées économiques pour les constructeurs. Les travaux de l'état de l'art proposent divers méthodes pour pouvoir optimiser et réduire l'énergie consommée par nos plateformes mobiles. Cependant, rares sont les travaux qui centrent leurs solutions sur les besoins de l'utilisateur final et son contexte. Or c'est le comportement de cet utilisateur qui détermine la consommation d'énergie de la plateforme mobile.

Dans cette thèse, nous avons proposé un ensemble de contributions qui visent à réduire la consommation d'énergie des systèmes mobiles par l'analyse des besoins et des habitudes de l'utilisateur. Les solutions proposées prennent en compte l'important flux de donnée exposé par le système d'exploitation et les capteurs embarqués disponibles. Nous avons proposé le modèle CPA pour collecter et traiter des données émanant de plusieurs sources. Le traitement résultant sert à configurer les différentes ressources matérielles ciblées. Ces différentes configurations HW implémentées permettent de réduire la consommation d'énergie du système, tout en respectant la satisfaction de l'utilisateur.

Cette thèse a été réalisée en collaboration avec *Intel Corp* dans le but d'étendre le framework IECSDK. Ce chapitre rappelle l'ensemble de nos contributions, ainsi que les perspectives des travaux futurs de cette thèse.

Sommaire

8.1 Conclusion Générale **178**
- 8.1.1 Sensor Based Optimization Component SBOC 178
- 8.1.2 Application Needs Based Optimization Component ANBOC 178
- 8.1.3 Enormous 178
- 8.1.4 User Request Based Optimization Component (URBOC) 179

8.2 Perspectives **179**
- 8.2.1 Perspectives générales 180
- 8.2.2 Améliorations d'ANBOC 180
- 8.2.3 Améliorations d'Enormous 182

8.1 Conclusion Générale

Ce travail a donné lieu à différentes contributions basées sur le modèle CPA qui sont:

8.1.1 Sensor Based Optimization Component SBOC

Ce composant représente la première contribution de cette thèse. SBOC est est une sur-couche logicielle qui s'ajoute au système d'exploitation, dans le but d'améliorer la consommation de puissance de la plateforme mobile. Notre composant est basé sur les données envoyées par les différents capteurs embarqués disponibles dans le système mobile. Cette première version de SBOC agit principalement sur la luminosité de l'écran et le volume des hauts parleurs mais peut être élargie à d'autres ressources hardwares. SBOC agit de façon dynamique sans phase d'apprentissage et configure les ressources HW à la volée, en fonction des données des capteurs. SBOC réduit dans certains cas de figure, la consommation de puissance à hauteur de 26%, avec un léger surcout. SBOC participe également à l'extension de l'Intel Energy Checker SDK, en lui ajoutant de nouvelles fonctionnalités par l'implémentation des *Inputs Libraries* et des *Actuator Librairies*.

8.1.2 Application Needs Based Optimization Component ANBOC

A la différence de son prédécesseur, ANBOC est axé sur les applications exécutées et leurs besoins en ressources hardwares. Cette sur-couche logicielle a été mise au point dans le but d'analyser l'interaction entre l'utilisateur et les applications que ce dernier utilise. Cette analyse nous sert par la suite à assimiler comment les différents sous-composants hardwares sont exploités par les applications et l'utilisateur. Cette compréhension de l'interaction entre l'utilisateur et l'application permet de cerner leurs besoins. Pour ce faire, nous exploitons des algorithmes d'intelligence artificielle et de fouille de données. Ces algorithmes d'intelligence artificielle servent dans un premier temps à classifier les applications selon leurs besoins en ressources HW. Ces algorithmes nous servent également prédire les futures applications qui seront lancées. La classification combinée à la prédiction permet d'anticiper les futures actions de l'utilisateur, tout en lui offrant la configuration hardware nécessaire. Cette configuration doit répondre à deux critères majeurs: une réduction de la consommation d'énergie, ainsi qu'un impact négligeable sur la qualité de service, les performances du système et la satisfaction de l'utilisateur. ANBOC nous permet d'avoir un gain d'énergie considérable dépassant les 30%, en comparaison avec la gestion de l'OS.

8.1.3 Enormous

Enormous obéit également au schéma CPA et est considéré comme la solution la plus englobante et complète de cette thèse. Contrairement à ANBOC qui était une solution axée sur les applications, Enormous est responsable de l'identification de l'ensemble du contexte de l'utilisateur. Ce contexte inclut les données sur l'utilisateur, le système et l'environnement. L'analyse du contexte

nous guide d'avantage dans la compréhension des habitudes de l'utilisateur et ses préférences. Cette identification permet de prédire et de cadrer les futures besoins de l'utilisateur en matière de configuration hardware. Le framework Enormous implémente un ensemble de réseaux de neurones artificiels, dans le but d'aboutir à un compromis entre les opportunités de réduction de la consommation de puissance et la satisfaction de l'utilisateur.

Notre framework configure dynamiquement la fréquence du processeur, ajuste le volume, la luminosité de l'écran et gère la l'état de l'interface Wi-Fi. À l'instar des composants précédents, Enormous est une solution générique et peut être étendu en gérant d'avantage de composants HW comme le GPU, le GPU, etc. En se basant sur la gestion des quatre ressources ressources hardwares, nous sommes parvenus à réduire la consommation d'énergie à hauteur de 35% en comparaison avec la politique de gestion standard, offerte par l'OS. Ce gain d'énergie est obtenu avec un surcout négligeable.

8.1.4 User Request Based Optimization Component (URBOC)

La dernière contribution de cette thèse est la mise en place du composant URBOC. Avec cette solution, nous essayons de faire intervenir l'utilisateur dans les optimisations, en lui donnant la possibilité d'envoyer des requêtes pour étendre la durée de vie de sa batterie. Contrairement aux solutions précédentes, URBOC ne prend pas l'initiative de configurer une ressource sans être sollicité.

Pour contextualiser l'utilité d'URBOC et son coté novateur, notre point de départ est un utilisateur lambda avec des contraintes critiques comme par exemple un rendez vous important ou un voyage d'affaires, sans qu'il ait la possibilité de recharger son système mobile. Avec URBOC, l'utilisateur aura la possibilité d'envoyer une requête comportant l'heure souhaitée de disponibilité de son système. URBOC analyse la faisabilité de la requête de l'utilisateur en se basant sur Enormous et cela pour chaque changement de contexte. URBOC utilise également une base de connaissances mise à sa disposition. Ce dernier composant exploite également la prédiction des applications développée pour ANBOC. URBOC applique différents niveaux d'optimisations hardwares dites extrêmes pour aboutir à la satisfaction de la requête de l'utilisateur. À ce stade de développement, nous n'avons pas encore déployé URBOC pour des tests dans un environnement réel, mais cette tâche est en cours d'élaboration.

8.2 Perspectives

Notre travail s'est attaqué à la problématique de la consommation d'énergie des systèmes mobiles, en se focalisant sur les besoins de l'utilisateur. Nous envisageons de profiter des connaissances acquises lors de la réalisation de cette thèse afin de traiter les limites d'ANBOC et d'Enormous. Cette partie résume les perspectives de cette thèse. Nous discutons dans ce qui suit des perspectives communes à toutes nos approches, puis nous développons les améliorations possibles pour nos deux composants ANBOC et d'Enormous.

8.2.1 Perspectives générales

Les contributions réalisées dans cette thèse ont été testées uniquement sur des plateformes *Windows*. Cette limitation est principalement due à l'actuelle version de l'IECSDK qui est uniquement compatible avec l'OS de *Microsft*. Néanmoins, en parallèle avec cette thèse, nous avons travaillé sur le portage de l'ESRV et le *Modeler* pour cibler des plateformes s'exécutant sous *Linux*. Cela permettrai l'implémentation de l'ensemble de nos composants pour des systèmes mobiles *Android*. À long terme, nous aurons la possibilité d'adapter les fonctionnalités de nos solutions selon chaque plateforme ciblée.

Une des difficultés rencontrées durant cette thèse est l'obtention des données des utilisateurs. En effet, nos solutions considèrent chaque utilisateur comme un cas à part, ce qui rend indispensable la collecte de données pour chaque utilisateur ciblé. Cependant, les utilisateurs sont souvent réticents à l'idée d'exposer leurs données personnelles à des fins d'analyse et d'études. C'est pour cette raison, que nous sommes actuellement entrain de réfléchir à un processus d'anonymisation des données collectées. Ce processus se fera en deux phases:

1. La suppression des identifiants des bases de données. Ces identifiants peuvent être le nom, prénom, age, adresse, etc.
2. L'application de filtres et transformations cryptographiques aux bases de données, comme par exemple, le chiffrement et le hachage de données par des algorithmes dédiés, comme l'algorithme Secure Hash Algorithm (SHA) [41].

Dans les versions actuelles de nos composants et notre framework, il n'était pas primordial de réaliser ce cryptage des données collectées car tout les traitements étaient réalisés dans le système mobile, en local et sans connexion aux serveurs distants.

Cette anonymisation servira également à collecter d'autres données sensibles, comme l'emplacement de l'utilisateur, son age, son métier, etc. Ces données nous offriront davantage de possibilités d'optimisations. Cela permettrai également d'enrichir nos mécanismes de classification et de prédiction.

Une autre piste prometteuse serait la décentralisation des différentes phases de traitement en utilisant le *cloud computing* comme nous l'exposons dans la section 8.2.3. Enfin, une des dernière amélioration commune serait la disponibilité de nos différentes solutions sous la forme d'applications dans les différents stores existants. Cela augmenterai la quantité des données collectées et permettrai également d'avoir les retours des utilisateurs, dans le but d'analyser leur satisfaction.

8.2.2 Améliorations d'ANBOC

Pour ANBOC, les améliorations que nous proposons peuvent être scindées en deux groupes:

- Améliorations de la classification des applications:
  1. Classification dynamique des applications en *run-time*: dans la version actuelle d'ANBOC, la classification est statique, nous envisageons dorénavant de réaliser cette classification dynamiquement. Avant le lancement de chaque application, le mécanisme de classification

sera lancé pour s'adapter aux besoins de l'utilisateur. Cette classification peut différer pour la même application, augmentant ainsi la prédiction de notre mécanisme.

2. Adaptation des algorithmes de classification selon chaque utilisateur: la précision de la classification varie en fonction des entrées dont elle dispose. Ces entrées sont corrélées au comportement de l'utilisateur, c'est pour cette raison que nous prévoyons de comparer plusieurs techniques de classification pour les utiliser au moment opportun et avec l'utilisateur adéquat. Pour ce faire, nous sommes actuellement entrain d'étudier d'autres algorithmes de classification tels que les arbres de décision, les machines à vecteurs de support (SVM) et les forêts aléatoires.
3. Choix automatique et adaptatif des classes de chaque classifieur: les classes de nos classifieurs sont fixées arbitrairement via la classification supervisée. Nous pensons à utiliser un apprentissage non supervisé pour que le nombre et le type des différentes classes en sortie soient sélectionnés automatiquement.

- Améliorations de la prédiction des applications:
  1. Ajout de contraintes pour la prédiction: dans la version actuelle de la prédiction, nous nous basons sur l'heure et le jour de la semaine. Nous proposons d'ajouter des contraintes d'ordre spatial comme l'emplacement de l'utilisateur et l'allure à laquelle il marche. En prenant en compte d'avantage de contraintes pour la prédiction, nous augmentons sa précision. Par exemple, au lieu de prédire les applications qui suivront pendant un Lundi de 8h à 10h, nous réaliserons la prédiction pour un Lundi de 8h à 10h lorsque l'utilisateur se trouve à son lieu de travail. Cela réduit le nombre d'applications susceptibles d'être prédites.
  2. Implémentation de techniques de *prefetching* des applications pour tirer profit de la prédiction: la version actuelle d'ANBOC ne tire pas pleinement profit de la prédiction. Cette dernière est principalement utilisée pour la classification et pour l'ajustement graduel dont le but de préserver la satisfaction de l'utilisateur. Nous proposons d'exploiter la prédiction pour réaliser le *prefetching* des données des applications.

     Prenons l'exemple minimaliste de deux applications qui se suivent: *Facebook* et *Huffington Post*. Nous pouvons précharger la page d'accueil ainsi que ses articles du *Huffington Post* pendant l'exécution de *Facebook*. Cela aura pour but de tirer profit de la connectivité utilisée pour *Facebook* et configurer la connectivité en OFF pendant le lancement de l'application *Huffington Post*, générant ainsi des économies d'énergie.

Comme mentionné précédemment, nous réfléchissons également à exécuter la classification et la prédiction sur des serveurs distants. Une telle décentralisation demande une étude approfondie du cout généré. En effet, comme nous avons pu le voir dans la section dédiée aux travaux antérieurs, l'*offloading computing* peut être fastidieux dans certains cas.

8.2.3 Améliorations d'Enormous

Nous étudions actuellement la possibilité de décentraliser Enormous dans le *cloud*, pour réaliser la phase d'apprentissage et la génération des politiques d'optimisations comme le montre la figure 8.1.

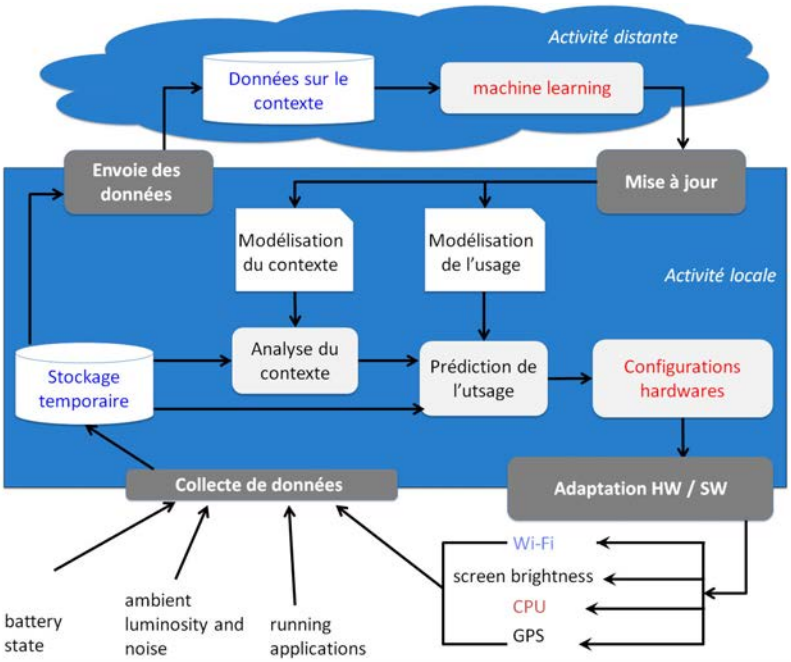

FIGURE 8.1 – Future architecture d'Enormous

La nouvelle architecture d'Enormous sera composée de deux parties.

1. Une partie embarquée dans le système mobile, appelée activité locale.
2. Une partie qui s'exécute à distance dans le *cloud*, appelée activité distante.

Cette architecture réduirait la mémoire utilisée, ainsi que les surcouts en temps de traitement et en consommation de puissance. Le mode de fonctionnement de cette architecture est comme suit:

1. Nous collecterons localement les différentes données, émanant de sources multiples.
2. Les données collectées sont stockées dans une base de données temporaire et seront envoyées à la partie distante. Cet envoi est réalisé lorsque le système sera en charge et connecté à un réseau Wi-Fi.
3. Une fois sur la partie distante, ces données vont être analysées via des algorithmes de *machine learning*. Nous utiliserons pour cette analyse différentes techniques d'intelligence artificielle comme le *deep learning* et les réseaux de neurones récurrents.
4. Ces techniques d'apprentissage seront adaptées à chaque utilisateur, selon la précision obtenue.

5. Les résultats de traitement seront utilisés pour prédire l'usage du système et la caractérisation du contexte de l'utilisateur.
6. La prédiction et la caractérisation seront utilisées pour configurer les différentes ressources HW disponibles.

Une telle architecture permettrai également de procéder à la classification de différents utilisateurs selon leur comportement et leurs besoins en configurations HW. Ce regroupement des différents utilisateurs servira à générer des politiques d'optimisations communes à un groupe d'utilisateurs donné. La génération de ces politiques communes sera réalisée dans le *cloud* et sera adaptée aux besoins de chaque utilisateur en local.

Table des figures

Figure 1.1 L'évolution du nombre d'utilisateurs de smartphones de 2014 à 2020 5
Figure 1.2 Les différents niveaux d'optimisation d'énergie dans les systèmes mobiles 8
Figure 1.3 La consommation de'énergie de différents utilisateurs dans l'état actif et *idle* [120]. 10
Figure 1.4 La consommation d'énergie des différents utilisateurs dans l'état actif [120] 11
Figure 1.5 La consommation énergétique de différents composants hardware dans Google Nexus One smartphone 12
Figure 1.6 Les étapes de l'étude des activités de l'utilisateur . . . 13
Figure 1.7 Modèle CPA pour l'optimisation de la consommation énergétique des systèmes mobiles 16

Figure 2.1 Processus de définition du modèle de la consommation d'énergie 41

Figure 3.1 Architecture simplifiée de l'IECSDK 48
Figure 3.2 Mesure de la consommation de puissance d'une plateforme via l'ESRV 49
Figure 3.3 L'utilisation du *Modeler* par l'ESRV 50
Figure 3.4 L'utilisation de différentes ILs 51
Figure 3.5 Renommage des inputs dans l'*Input-Bus* 51
Figure 3.6 Exemple de communication entre ILs et ALs 52
Figure 3.7 Installation de l'Ultrabook avec le Yokogawa WT210 . 53
Figure 3.8 Consommation d'énergie moyenne des composants hardware de l'Ultrabook 54

Figure 4.1 Abstraction de l'architecture globale de SBOC basée sur le modèle CPA 61
Figure 4.2 Architecture modulaire de SBOC 62
Figure 4.3 Correspondance entre capteurs physiques et abstractions logicielles 64
Figure 4.4 Scénarios détectés par le SPM 67
Figure 4.5 Les variations des métriques de l'inclinomètre 69
Figure 4.6 Seuil de l'inclinaison 69
Figure 4.7 Comparaison entre les métriques du gyromètre et l'accéléromètre 70

Figure 4.8 Résultats de l'étude expérimentale sur les données du microphone . . . 72
Figure 4.9 Seuils des amplitudes du bruit ambiant . . . 73
Figure 4.10 Les entrées / sorties du SPM . . . 73
Figure 4.11 Mode de fonctionnement du RMM . . . 74
Figure 4.12 Consommation de puissance avec la gestion de luminosité proposée avec SBOC . . . 77
Figure 4.13 Résultats de la consommation de puissance pour le scénario de la luminosité . . . 79
Figure 4.14 Consommation d'énergie normalisée pour la gestion de la luminosité . . . 79
Figure 4.15 Consommation de puissance des configurations du volume . . . 81
Figure 4.16 Consommation d'énergie normalisée pour les configurations du volume . . . 81
Figure 4.17 Coût de SOC . . . 82
Figure 4.18 Coût de la collecte des métriques du bruit ambiant . . . 83
Figure 4.19 Coût de la collecte des métriques pour la gestion de la luminosité . . . 83

Figure 5.1 Abstraction de l'architecture globale de ANBOC basée sur le modèle CPA . . . 87
Figure 5.2 Collecte de données pour la classification des applications . . . 89
Figure 5.3 Collecte de données pour la prédiction . . . 91
Figure 5.4 Exemple de fonctionnement du DVFS . . . 95
Figure 5.5 Les sorties de GSP . . . 104
Figure 5.6 Prédiction des applications par l'OA . . . 105
Figure 5.7 Fonctionnement de l'OA . . . 106
Figure 5.8 La phase d'ajustement réalisée par l'OA . . . 107
Figure 5.9 Différence entre *ondemand governor* de l'OS avec ANBOC 110
Figure 5.10 La phase d'ajustement réalisée par l'OA . . . 111
Figure 5.11 Séquences d'applications d'utilisateurs réels . . . 112
Figure 5.12 Consommation d'énergie pour chaque utilisateur . . . 114
Figure 5.13 Réductions d'énergie obtenues avec chaque application 115
Figure 5.14 Réduction d'énergie normalisée pour chaque composant 116

Figure 6.1 Abstraction de l'architecture globale d' Enormous basée sur le modèle CPA . . . 123
Figure 6.2 Modules logiciels d'Enormous . . . 124
Figure 6.3 Contexte de l'utilisateur collecté par le DCM . . . 124
Figure 6.4 Changement du contexte de l'utilisateur en fonction de l'application en premier plan . . . 126
Figure 6.5 Classifieurs Ad-Hoc . . . 130
Figure 6.6 Processus de sélection des sorties souhaitées des classifieurs . . . 134
Figure 6.7 MLP 3 – 2 – 1 . . . 137
Figure 6.8 Propagation de l'information . . . 139
Figure 6.9 Retropropagation de l'erreur . . . 139
Figure 6.10 Feed Forward Back-Propagation dédié au volume . . . 143

Figure 6.11 Cascade Back Propagation dédié à la luminosité . . . 143
Figure 6.12 Comparaison des architectures des ANN's pour l'utilisateur 1 144
Figure 6.13 Architecture d'Enormous basée sur l'IECSDK 146
Figure 6.14 Gestion de la fréquence CPU de l'OS vs Enormous . . 149
Figure 6.15 Gestion du volume de l'OS Vs Enormous 149
Figure 6.16 Gestion de la luminosité de l'OS Vs Enormous 149
Figure 6.17 Résultats de la consommation de puissance avec le déploiement d'Enormous 150
Figure 6.18 Scénarios d'utilisation pour les utilisateurs synthétiques 151
Figure 6.19 Consommation de puissance des utilisateurs 1,2 et 3 . 152
Figure 6.20 Consommation de puissance des utilisateurs 4,5 et 6 . 154
Figure 6.21 Synthèse des gains énergétique sur les 6 scénarios . . 155
Figure 6.22 Consommation de puissance de l'EDCM 156

Figure 7.1 Architecture fonctionnelle d'URBOC 159
Figure 7.2 Architecture modulaire d'URBOC 161
Figure 7.3 Base de connaissances SKD 162
Figure 7.4 Politiques d'optimisations extrêmes HOP 163
Figure 7.5 Retour aux optimisation d'Enormous à chaque changement de contexte 163
Figure 7.6 Application des HOP par l'UOA 164
Figure 7.7 Le rôle de la prédiction dans URBOC 166
Figure 7.8 Scénario 1: Applications des *Power Policies* d'Enormous sans l'utilisation des HOP 168
Figure 7.9 Scénario 2: Utilisation du premier niveau d'optimisation de HOP 168
Figure 7.10 Scénario 3: Application du second niveau d'optimisation HOP 169
Figure 7.11 Scénario 4: Application du troisième niveau d'optimisation HOP 170
Figure 7.12 Scénario 4: Application du troisième niveau d'optimisation HOP 172
Figure 7.13 Scénario 4: Application du troisième niveau d'optimisation HOP 173

Figure 8.1 Future architecture d'Enormous 182

Liste des tableaux

Tableau 1.1 Exemples de paramètres utiles pour l'étude du CPU et de l'écran 14
Tableau 1.2 Synthèse des contributions de cette thèse 18

Tableau 2.1 Systèmes d'exploitation soucieux de l'énergie 23
Tableau 2.2 Profileurs de ressources 25
Tableau 2.3 Optimisation des capteurs embarqués dans les systèmes mobiles 31
Tableau 2.4 Travaux sur l'interaction de l'utilisateur avec les applications et les ressources hardwares 37
Tableau 2.5 Travaux sur l'interaction de l'utilisateur avec les batteries 40
Tableau 2.6 Exemple de paramètres fixés pour le CPU [77] 41

Tableau 3.1 Intel 2in1 Ultrabook Features 53

Tableau 4.1 Détection du niveau de la lumière ambiante en fonction de la métrique de ALS 71
Tableau 4.2 Heuristiques pour la gestion de la luminosité 75
Tableau 4.3 Table de correspondance pour la gestion de la luminosité en fonction de la lumière ambiante 75
Tableau 4.4 Heuristiques pour la gestion du volume 76
Tableau 4.5 Scénario pour la gestion de la luminosité 78
Tableau 4.6 Configurations appliquées au scénario de gestion de la luminosité 78
Tableau 4.7 Scénario pour la gestion du volume 80
Tableau 4.8 Configurations appliquées au scénario pour la gestion du volume 80

Tableau 5.1 Exemple de la classification Wi-Fi des applications . . 94
Tableau 5.2 Les différentes P-states de l'*Ultrabook* 97
Tableau 5.3 Exemple de classification CPU pour un utilisateur labmda 100
Tableau 5.4 Exemple de données en entrée pour GSP 103
Tableau 5.5 Classification Wi-Fi des applications 113
Tableau 5.6 Résultats de la classification CPU pour les 6 utilisateurs 113
Tableau 5.7 Coût du ANBOC 116
Tableau 5.8 Résultats Du DFS 117

Tableau 6.1 Catégories des applications 133
Tableau 6.2 Formalisme utilisé pour nos MLP 136

Tableau 6.3 Codage du réseau MLP 3 – 2 – 1 138
Tableau 6.4 Architecture des réseaux de neurones pour tous les utilisateurs 145
Tableau 6.5 Contextes d'utilisateurs pour six utilisateurs réels . . . 148
Tableau 6.6 Résultats de la classification 148
Tableau 6.7 Comparaison de la gestion de la Wi-Fi 150
Tableau 6.8 Classification results for users 1,2 and 3 152
Tableau 6.9 Classification results for users 4, 5 and 6 153
Tableau 6.10 Mesure de la phase d'apprentissage de nos classifieurs 157

Tableau 7.1 Informations sur l'état actuel du système mobile . . . 167
Tableau 7.2 Scénarios utilisant uniquement les HOP 167
Tableau 7.3 Informations sur l'état actuel du système 171
Tableau 7.4 Prédiction des applications 171
Tableau 7.5 Informations de la SKD 171
Tableau 7.6 Scénarios utilisant la prédiction 171

Liste des Algorithmes

Algorithm 1 Algorithme de fonctionnement du DCM 92
Algorithm 2 Pseudo-code de l'algorithme ondemand governor de Windows 109
Algorithm 3 Pseudo-code de l'algorithme ondemand governor d'ANBOC 110

Algorithm 4 Algorithme de BPLA 142

Bibliographie

[1] G. Abowd, A. Dey, P. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer, 1999.

[2] M. Al-Maolegi and B. Arkok. An improved apriori algorithm for association rules. 2014.

[3] S. Alawnah and A. Sagahyroon. *Modeling smartphones power*. IEEE, 2013.

[4] M. Anand, E. B. Nightingale, and J. Flinn. Ghosts in the machine: Interfaces for better power management. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 23–35. ACM, 2004.

[5] A. Asthana and R. Asthana. ios 5, android 4.0 and windows 8–a review. In *IEEE Code of Ethics*, 2012.

[6] R. B. Atitallah, S. Niar, A. Greiner, S. Meftali, and J. L. Dekeyser. Estimating energy consumption for an mpsoc architectural exploration. In *International Conference on Architecture of Computing Systems*, pages 298–310. Springer, 2006.

[7] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

[8] N. Banerjee, A. Rahmati, M. Corner, S. Rollins, and L. Zhong. Users and batteries: interactions and adaptive energy management in mobile systems. *UbiComp 2007: Ubiquitous Computing*, pages 217–234, 2007.

[9] F. Bellosa. The benefits of event: driven energy accounting in power-sensitive systems. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, pages 37–42. ACM, 2000.

[10] F. Ben Abdesslem, A. Phillips, and T. Henderson. Less is more: energy-efficient mobile sensing with senseless. In *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, pages 61–62. ACM, 2009.

[11] L. Benini, A. Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE transactions on very large scale integration (VLSI) systems*, 8(3):299–316, 2000.

[12] L. Benini and G. DeMicheli. *Dynamic power management: design techniques and CAD tools*. Springer Science & Business Media, 1997.

[13] S. Bhattacharya, H. Blunck, M. B. Kjærgaard, and P. Nurmi. Robust and energy-efficient trajectory tracking for mobile devices. *IEEE Transactions on Mobile Computing*, 14(2):430–443, 2015.

[14] K. Bhatti, C. Belleudy, and M. Auguin. Power management in real time embedded systems through online and adaptive interplay of dpm and dvfs policies. In *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, pages 184–191. IEEE, 2010.

[15] S. Bohez, T. Verbelen, P. Simoens, and B. Dhoedt. Discrete-event simulation for efficient and stable resource allocation in collaborative mobile cloudlets. *Simulation Modelling Practice and Theory*, 50:109–129, 2015.

[16] P. Borne, M. Benrejeb, and J. Haggège. *Les réseaux de neurones: présentation et applications*, volume 15. Editions OPHRYS, 2007.

[17] L. S. Brakmo, D. A. Wallach, and M. A. Viredaz. μsleep: a technique for reducing energy consumption in handheld devices. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 12–22. ACM, 2004.

[18] D. Brodowski and N. Golde. Cpu frequency and voltage scaling code in the linux (tm) kernel. *Linux kernel documentation*, 2013.

[19] A. Bundy and L. Wallen. Breadth-first search. In *Catalogue of Artificial Intelligence Tools*, pages 13–13. Springer, 1984.

[20] C. Bunse. On the impact of user feedback on energy consumption. In *EnviroInfo*, pages 759–764, 2014.

[21] D. Burgstahler, U. Lampe, N. Richerzhagen, and R. Steinmetz. Push vs. pull: An energy perspective (short paper). In *Service-Oriented Computing and Applications (SOCA), 2013 IEEE 6th International Conference on*, pages 190–193. IEEE, 2013.

[22] D. Burgstahler, N. Richerzhagen, F. Englert, R. Hans, and R. Steinmetz. Switching push and pull: An energy efficient notification approach. In *Mobile Services (MS), 2014 IEEE International Conference on*, pages 68–75. IEEE, 2014.

[23] O. Carlier, J. Tayeb, M. Desertot, S. Lecomte, and S. Niar. Run-time users/applications interaction analysis for power consumption optimization. In *Energy Aware Computing Systems and Applications (ICEAC), 2013 4th Annual International Conference on*, pages 181–186. IEEE, 2013.

[24] A. Carroll and G. Heiser. Unifying dvfs and offlining in mobile multicores. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014 IEEE 20th*, pages 287–296. IEEE, 2014.

[25] Z.-G. Che, T.-A. Chiang, Z.-H. Che, et al. Feed-forward neural networks training: a comparison between genetic algorithm and back-propagation learning algorithm. *International Journal of Innovative Computing, Information and Control*, 7(10):5839–5850, 2011.

[26] D. Chu, A. Kansal, J. Liu, and F. Zhao. Mobile apps: It's time to move up to condos. In *HotOS*, 2011.

[27] I. Constandache, S. Gaonkar, M. Sayler, R. R. Choudhury, and L. Cox. Enloc: Energy-efficient localization for mobile phones. In *INFOCOM 2009, IEEE*, pages 2716–2720. IEEE, 2009.

[28] S. K. Datta, C. Bonnet, and N. Nikaein. Minimizing energy expenditure in smart devices. In *Information & Communication Technologies (ICT), 2013 IEEE Conference on*, pages 712–717. IEEE, 2013.

[29] M. Á. De La Concepción, L. S. Morillo, L. Gonzalez-Abril, and J. O. Ramírez. Discrete techniques applied to low-energy mobile human activity recognition. a new approach. *Expert Systems with Applications*, 41(14):6138–6146, 2014.

[30] T. Denoeux, O. Kanjanatarakul, and S. Sriboonchitta. Ek-nnclus: a clustering procedure based on the evidential k-nearest neighbor rule. volume 88, pages 57–69. Elsevier, 2015.

[31] A. K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.

[32] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction*, 16(2):97–166, 2001.

[33] V. K. P. Dheeraj S. Badde, Anil k. Gupta. Cascade and feed forward back propagation artificial neural network models for prediction of compressive strength of ready mix concrete. *Journal of Mechanical and Civil Engineering*, 1997.

[34] B. K. Donohoo. *Machine learning techniques for energy optimization in mobile embedded systems*. PhD thesis, Colorado State University, 2012.

[35] I. C. Draa, E. Grislin-Le Strugeon, and S. Niar. An energy-aware learning agent for power management in mobile devices. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 242–245. Springer, 2017.

[36] I. C. Draa, S. Niar, J. Tayeb, E. Grislin, and M. Desertot. Sensing user context and habits for run-time energy optimization. *EURASIP Journal on Embedded Systems*, 2017(1):4, 2017.

[37] I. C. Draa, M. Nouiri, S. Niar, and A. Bekrar. Device context classification for mobile power consumption reduction. In *Digital System Design (DSD), 2016 Euromicro Conference on*, pages 682–685. IEEE, 2016.

[38] I. C. Draa, J. Tayeb, S. Niar, and M. Desertot. User information analysis for energy consumption optimization in mobile systems. In *Proceedings of the Workshop on High Performance Energy Efficient Embedded Systems (HIP3ES) colocated with HIPEAC*, 2015.

[39] I. C. Draa, J. Tayeb, S. Niar, and E. Grislin. Application sequence prediction for energy consumption reduction in mobile systems. In *Computer*

and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on, pages 23–30. IEEE, 2015.

[40] N. Eagle and A. S. Pentland. Reality mining: sensing complex social systems. *Personal and ubiquitous computing*, 10(4):255–268, 2006.

[41] D. Eastlake 3rd and P. Jones. Us secure hash algorithm 1 (sha1). Technical report, 2001.

[42] C. S. Ellis. The case for higher-level power management. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 162–167. IEEE, 1999.

[43] E. C. Evarts. Lithium batteries: To the limits of lithium. *Nature*, 526(7575):S93–S95, 2015.

[44] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *Advances in neural information processing systems*, pages 524–532, 1990.

[45] H. Falaki, R. Govindan, and D. Estrin. Smart screen management on mobile phones. *Center for Embedded Network Sensing*, 2009.

[46] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A first look at traffic on smartphones. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 281–287. ACM, 2010.

[47] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 179–194. ACM, 2010.

[48] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99. Second IEEE Workshop on*, pages 2–10. IEEE, 1999.

[49] J. Flinn and M. Satyanarayanan. Managing battery lifetime with energy-aware adaptation. volume 22, pages 137–179. ACM, 2004.

[50] V. Freycon. Les gps principes de fonctionnement et conseils d'utilisation. 1996.

[51] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison, and J. A. Landay. Myexperience: a system for in situ tracing and capturing of user feedback on mobile phones. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 57–70. ACM, 2007.

[52] N. Gautam, H. Petander, and J. Noel. A comparison of the cost and energy efficiency of prefetching and streaming of mobile video. In *Proceedings of the 5th Workshop on Mobile Video*, pages 7–12. ACM, 2013.

[53] S. Giordano and D. Puccinelli. When sensing goes pervasive. *Pervasive and Mobile Computing*, 17:175–183, 2015.

[54] R. Godet. Avis mobiles,https://avismobiles.fr/astuces/a-quoi-servent-les-capteurs-proximite-luminosite-accelerometre-gps-gyroscope-magnetometre/, 2017.

[55] M. Goraczko, A. Kansal, J. Liu, and F. Zhao. Joulemeter: Computational energy measurement and optimization, 2011.

[56] T.-M. Gronli, J. Hansen, G. Ghinea, and M. Younas. Mobile application platform heterogeneity: Android vs windows phone vs ios vs firefox os. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, pages 635–641. IEEE, 2014.

[57] R. Hans, U. Lampe, D. Burgstahler, M. Hellwig, and R. Steinmetz. Where did my battery go? quantifying the energy consumption of cloud gaming. In *Mobile Services (MS), 2014 IEEE International Conference on*, pages 63–67. IEEE, 2014.

[58] R. Hans, M. Zahn, U. Lampe, R. Steinmetz, and A. Papageorgiou. Energy-efficient web service invocation on mobile devices: The influence of compression and parsing. In *Proceedings of the 2nd International Conference on Mobile Services*, 2013.

[59] S. Hao, D. Li, W. G. Halfond, and R. Govindan. Estimating mobile application energy consumption using program analysis. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 92–101. IEEE, 2013.

[60] T. Heath, E. Pinheiro, J. Hom, U. Kremer, and R. Bianchini. Code transformations for energy-efficient device management. *IEEE Transactions on Computers*, 53(8):974–987, 2004.

[61] N. Hirofumi, N. Naoya, and T. Katsuya. Wt210/wt230 digital power meters. 2003.

[62] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[63] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[64] W. G. Howard, C. L. Schmidt, and E. R. Scott. Lithium-ion battery, July 21 2009. US Patent 7,563,541.

[65] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing application performance differences on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 165–178. ACM, 2010.

[66] G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995.

[67] D. Kadjo, U. Ogras, R. Ayoub, M. Kishinevsky, and P. Gratz. Towards platform level power management in mobile systems. In *System-on-Chip Conference (SOCC), 2014 27th IEEE International*, pages 146–151. IEEE, 2014.

[68] M. B. Kjærgaard, J. Langdal, T. Godsk, and T. Toftkjær. Entracked: energy-efficient robust position tracking for mobile devices. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, pages 221–234. ACM, 2009.

[69] S. Kotsiantis. supervised machine learning: a review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 2007.

[70] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas. Supervised machine learning: A review of classification techniques. volume 160, pages 3–24, 2007.

[71] M. H. Kutner, C. Nachtsheim, and J. Neter. *Applied linear regression models*. McGraw-Hill/Irwin, 2004.

[72] K. Kwon, S. Chae, and K.-G. Woo. An application-level energy-efficient scheduling for dynamic voltage and frequency scaling. In *Consumer Electronics (ICCE), 2013 IEEE International Conference on*, pages 3–6. IEEE, 2013.

[73] Y.-W. Kwon and E. Tilevich. Energy-efficient and fault-tolerant distributed mobile execution. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 586–595. IEEE, 2012.

[74] A. B. Lago and I. Larizgoitia. An application-aware approach to efficient power management in mobile devices. In *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, page 11. ACM, 2009.

[75] Laterdroid. Juice defender, 2013.

[76] J. Lee, H. Joe, and H. Kim. Smart phone power model generation using use pattern analysis. In *Consumer Electronics (ICCE), 2012 IEEE International Conference on*, pages 412–413. IEEE, 2012.

[77] J. Lee, H. Joe, and H. Kim. Automated power model generation method for smartphones. *IEEE Transactions on Consumer Electronics*, 60(2):190–197, 2014.

[78] M. Lemoine and F. Pelgrin. Introduction aux modèles espace-état et au filtre de kalman. *Revue de l'OFCE*, (3):203–229, 2003.

[79] X. Li, G. Yan, Y. Han, and X. Li. Smartcap: user experience-oriented power adaptation for smartphone's application processor. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 57–60. EDA Consortium, 2013.

[80] K. Lin, A. Kansal, D. Lymberopoulos, and F. Zhao. Energy-accuracy trade-off for continuous mobile device location. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 285–298. ACM, 2010.

[81] X. Liu, P. Shenoy, and M. Corner. Chameleon: application level power management with performance isolation. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 839–848. ACM, 2005.

[82] R. Loomba, L. Shi, B. Jennings, R. Friedman, J. Kennedy, and J. Butler. Energy-aware collaborative sensing for multiple applications in mobile cloud computing. *Sustainable Computing: Informatics and Systems*, 8:47–59, 2015.

[83] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell. The jigsaw continuous sensing engine for mobile phone applications. In *Proceedings of the 8th ACM conference on embedded networked sensor systems*, pages 71–84. ACM, 2010.

[84] L. Luo. *Designing energy and user efficient interactions with mobile systems*. Carnegie Mellon University, 2008.

[85] M. Marjanović, L. Skorin-Kapov, K. Pripužić, A. Antonić, and I. P. Žarko. Energy-aware and quality-driven sensor management for green mobile crowd sensing. *Journal of network and computer applications*, 59:95–108, 2016.

[86] F. Masseglia, M. Teisseire, and P. Poncelet. Sequential pattern mining. In *Encyclopedia of Data Warehousing and Mining*, pages 1028–1032. IGI Global, 2005.

[87] S. Menard. *Applied logistic regression analysis*, volume 106. SAGE publications, 2018.

[88] L. M. S. Morillo, L. Gonzalez-Abril, J. A. O. Ramirez, and M. A. A. de la Concepcion. Low energy physical activity recognition system on smartphones. *Sensors*, 15(3):5163–5196, 2015.

[89] S. Mustière. *Apprentissage supervisé pour la généralisation cartographique*. PhD thesis, Paris 6, 2001.

[90] D. Narayanan and M. Satyanarayanan. Predictive resource management for wearable computing. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 113–128. ACM, 2003.

[91] R. Neugebauer and D. McAuley. Energy is just another resource: Energy accounting and energy pricing in the nemesis os. In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pages 67–72. IEEE, 2001.

[92] B. Noble, M. Price, and M. Satyanarayanan. A programming interface for application-aware adaptation in mobile computing. *Computing Systems*, 8(4):345–363, 1995.

[93] M. Nosrati, R. Karimi, and H. A. Hasanvand. Mobile computing: principles, devices and operating systems. *World Applied Programming*, 2(7):399–408, 2012.

[94] E. Oliver. Diversity in smartphone energy consumption. In *Proceedings of the 2010 ACM workshop on Wireless of the students, by the students, for the students*, pages 25–28. ACM, 2010.

[95] E. Oliver and S. Keshav. Data driven smartphone energy level prediction. *University of Waterloo Technical Report*, 2010.

[96] J. Paek, J. Kim, and R. Govindan. Energy-efficient rate-adaptive gps-based positioning for smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 299–314. ACM, 2010.

[97] S. K. Pal and S. Mitra. Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on neural networks*, 3(5):683–697, 1992.

[98] D. Panigrahi, C. Chiasserini, S. Dey, R. Rao, A. Raghunathan, K. Lahiri, et al. Battery life estimation of mobile embedded systems. In *VLSI Design, 2001. Fourteenth International Conference on*, pages 57–63. IEEE, 2001.

[99] R. Pérez-Torres, C. Torres-Huitzil, and H. Galeana-Zapién. Power management techniques in smartphone-based mobility sensing systems: A survey. *Pervasive and Mobile Computing*, 31:1–21, 2016.

[100] G. P. Perrucci, F. H. Fitzek, and M. V. Petersen. Energy saving aspects for mobile device exploiting heterogeneous wireless networks. In *Heterogeneous Wireless Access Networks*, pages 1–27. Springer, 2008.

[101] H. Petander. Energy-aware network selection using traffic estimation. In *Proceedings of the 1st ACM workshop on Mobile internet through cellular networks*, pages 55–60. ACM, 2009.

[102] N. Pettis, L. Cai, and Y.-H. Lu. Statistically optimal dynamic power management for streaming data. *IEEE Transactions on Computers*, 55(7):800–814, 2006.

[103] A. Pit. Ambient luminosity for screen brightness, 2015.

[104] N. Priyantha, D. Lymberopoulos, and J. Liu. Eers: Energy efficient responsive sleeping on mobile phones. volume 2010, pages 1–5, 2010.

[105] A. Pröbstl, P. Kindt, E. Regnath, and S. Chakraborty. Smart2: Smart charging for smart phones. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2015 IEEE 21st International Conference on*, pages 41–50. IEEE, 2015.

[106] A. Rahmati, A. Qian, and L. Zhong. Understanding human-battery interaction on mobile phones. In *Proceedings of the 9th international conference on Human computer interaction with mobile devices and services*, pages 265–272. ACM, 2007.

[107] A. Rahmati and 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*, pages 165–178. ACM, 2007.

[108] D. Rakhamtov and S. Vrudhula. Time-to-failure estimation for batteries in portable electronic systems. In *Proceedings of the 2001 international symposium on Low power electronics and design*, pages 88–91. ACM, 2001.

[109] D. Rakhmatov, S. Vrudhula, and D. A. Wallach. Battery lifetime prediction for energy-aware computing. In *Proceedings of the 2002 international symposium on Low power electronics and design*, pages 154–159. ACM, 2002.

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

[111] P. Rebentrost, M. Mohseni, and S. Lloyd. Quantum support vector machine for big data classification. *Physical review letters*, 113(13):130503, 2014.

[112] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. Energy management in mobile devices with the cinder operating system. In *Proceedings of the sixth conference on Computer systems*, pages 139–152. ACM, 2011.

[113] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning. Saving portable computer battery power through remote process execution. *ACM SIGMOBILE Mobile Computing and Communications Review*, 2(1):19–26, 1998.

[114] R. J. Schalkoff. *Artificial neural networks*, volume 1. McGraw-Hill New York, 1997.

[115] R. R. Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.

[116] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. N. Padmanabhan. Bartendr: a practical approach to energy-aware cellular data scheduling. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, pages 85–96. ACM, 2010.

[117] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum. Livelab: measuring wireless networks and smartphone users in the field. *ACM SIGMETRICS Performance Evaluation Review*, 38(3):15–20, 2011.

[118] A. Shye. *The end user in computer architecture and systems research*. Northwestern University, 2010.

[119] A. Shye, B. Scholbrock, and G. Memik. Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceed. of the 42nd Annual IEEE/ACM Int. Symposium on Microarchitecture*, MICRO 42, pages 168–178, New York, NY, USA, 2009. ACM.

[120] A. Shye, B. Scholbrock, G. Memik, and P. A. Dinda. Characterizing and modeling user activity on smartphones. 2010.

[121] D. J. Simons and D. T. Levin. Change blindness. *Trends in cognitive sciences*, 1(7):261–267, 1997.

[122] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser. Koala: A platform for os-level power management. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 289–302. ACM, 2009.

[123] W. Song, N. Sung, B.-G. Chun, and J. Kim. Reducing energy consumption of smartphones using user-perceived response time analysis. In *Proceed. of the 15th Workshop on Mobile Computing Systems and Applications*, HotMobile '14, pages 20:1–20:6, New York, NY, USA, 2014. ACM.

[124] R. Srikant and R. Agarwal. Mining sequential patterns: Generalizations and performance improvements. In *Proceed. of the 5th Int. Conf. on EDT:Advances in Database Technology*, pages 3–17.

[125] S. Taleb, M. Dia, J. Farhat, Z. Dawy, and H. Hajj. On the design of energy-aware 3g/wifi heterogeneous networks under realistic conditions. In *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, pages 523–527. IEEE, 2013.

[126] W. L. Tan, F. Lam, and W. C. Lau. An empirical study on the capacity and performance of 3g networks. *IEEE Transactions on Mobile Computing*, 7(6):737–750, 2008.

[127] J. Tanha, M. van Someren, and H. Afsarmanesh. Semi-supervised self-training for decision tree classifiers. volume 8, pages 355–370. Springer, 2017.

[128] R. Tarjan. Depth-first search and linear graph algorithms. In *Switching and Automata Theory, 1971., 12th Annual Symposium on*, pages 114–121. IEEE, 1971.

[129] F. Tech. Microsystème électromécanique, http://www.futura-sciences.com/tech/definitions/electronique-microsysteme-electromecanique-5717/, 2016.

[130] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci. Measuring serendipity: connecting people, locations and interests in a mobile 3g network. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 267–279. ACM, 2009.

[131] C.-P. Tsai and T.-L. Lee. Back-propagation neural network in tidal-level forecasting. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, 125(4):195–202, 1999.

[132] A. Vahdat, A. Lebeck, and C. S. Ellis. Every joule is precious: The case for revisiting operating system design for energy efficiency. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, pages 31–36. ACM, 2000.

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

[134] N. Vallina-Rodriguez, P. Hui, J. Crowcroft, and A. Rice. Exhausting battery statistics: understanding the energy demands on mobile handsets. In *Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*, pages 9–14. ACM, 2010.

[135] T. Vincent and O. Philippot. Software measurement of energy consumption on smartphones. In *Greening Video Distribution Networks*, pages 127–132. Springer, 2018.

[136] C. Wang, F. Yan, Y. Guo, and X. Chen. Power estimation for mobile applications with profile-driven battery traces. In *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, pages 120–125. IEEE Press, 2013.

[137] Y. Wen, R. Wolski, and C. Krintz. Online prediction of battery lifetime for embedded and mobile devices. In *PACS*, volume 3, pages 57–72. Springer, 2003.

[138] C. K. Williams and D. Barber. Bayesian classification with gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351, 1998.

[139] D. Willkomm, S. Machiraju, J. Bolot, and A. Wolisz. Primary user behavior in cellular networks and implications for dynamic spectrum access. *IEEE Communications Magazine*, 47(3), 2009.

[140] F. Xia, F. Ding, J. Li, X. Kong, L. T. Yang, and J. Ma. Phone2cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing. *Information Systems Frontiers*, 16(1):95–111, 2014.

[141] H. Xiong, D. Zhang, L. Wang, J. P. Gibson, and J. Zhu. Eemc: Enabling energy-efficient mobile crowdsensing with anonymous participants. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):39, 2015.

[142] L. Xu. Least mean square error reconstruction principle for self-organizing neural-nets. *Neural networks*, 6(5):627–648, 1993.

[143] M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. volume 42, pages 31–60. Springer, 2001.

[144] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Ecosystem: Managing energy as a first class operating system resource. volume 36, pages 123–132. ACM, 2002.

[145] Z. Zhuang, K.-H. Kim, and J. P. Singh. Improving energy efficiency of location sensing on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 315–330. ACM, 2010.