FEATURE-ORIENTED MODEL-DRIVEN DEVELOPMENT OF ENERGY-AWARE SELF-ADAPTIVE SOFTWARE

Thesis

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

MARIMUTHU C

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
SURATHKAL, MANGALORE-575025

June, 2021

D E C L A R A T I O N

by the Ph.D. Research Scholar

I hereby *declare* that the Research Thesis entitled **Feature-oriented Model-driven Development of Energy-aware Self-adaptive Software** which is being submitted to the National Institute of Technology Karnataka, Surathkal in partial fulfilment of the requirements for the award of the Degree of Doctor of Philosophy in **Computer Science and Engineering** is a *bonafide report of the work carried out by me*. The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.

[illegible]

. . . .

(Register Number, Name and Signature of the Research Scholar)

(155126 CS15FV08, Marimuthu C)

Department of Computer Science and Engineering

Place: NITK SURATHKAL

Date: June 15, 2021

C E R T I F I C A T E

This is to *certify* that the Research Thesis entitled "**Feature-oriented Model-driven Development of Energy-aware Self-adaptive Software**" submitted by **Marimuthu C** (Register Number: 155126 CS15FV08) as the record of the work carried out by him, is *accepted as the Research Thesis submission* in partial fulfilment of the requirements for the award of degree of Doctor of Philosophy.

[illegible]
15/06/2021

Dr. K. CHANDRA SEKARAN Ph.D.
Professor, Dept. of Computer Sc. & Engg.
National Institute of Technology Karnataka
Surathkal, Srinivasnagar - 575 025
Mangalore, Karnataka State, India.

Prof. K. Chandrasekaran
Research Guide
(Signature with Date and Seal)

[illegible]
17/06/2021

Chairman - DRPC
(Signature with Date and Seal)

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. K. Chandrasekaran for continuously supporting me throughout my Ph.D. study with motivation and patience. His immense knowledge and guidance has always helped me. Without his precious support, it would not have been possible to conduct this research.

Besides my advisor, I would like to thank the rest of my research progress assessment committee members: Prof. A. Kandasamy, and Dr. Shashidhar G Koolagudi, for their insightful comments, encouragement, and hard questions which incented me to widen my research from various perspectives.

I would like to acknowledge Dr. Sridhar Chimalakonda (IIT Tirupati) for being instrumental in defining the path of my research. He has played a major role in polishing my research writing skills. His endless guidance is hard to forget throughout my life.

I would always remember my fellow labmates and other research scholars for the fun-time we spent together, sleepless nights that gave us the courage to complete tasks before deadlines. In particular, I am thankful to my friend Mr. Raghavan for helping in proofreading my research papers and Ph.D. thesis.

I express my deep gratitude to the present and previous Heads of the Department, Department of Computer Science and Engineering, NITK Surathkal for their constant co-operation, support, and for providing necessary facilities throughout my Ph.D. program. I would like to take this opportunity to express my thanks to the teaching and non-teaching staff in the Department of Computer Science and Engineering, NITK Surathkal for their valuable help and support during my study.

Marimuthu C

Abstract

Smartphone applications are equipped with energy-hungry resources such as display, GPS, and GPU. Mishandling of these resources and associated APIs might result in an abnormal battery drain. In recent years, researchers have adopted self-adaptive strategies to extend battery life with context information. However, the existing solutions focus on the development and testing phases of software development. Handling the energy-awareness and self-adaptive behavior directly at the development phase would increase the development efforts. Therefore, there is a need to consider these requirements in the early phases of software development life cycle. Thus, in this research work, the concepts of feature modeling, domain-specific modeling languages, and code generation has been adopted to model and develop energy-aware self-adaptive software. The location-based applications have been selected as an application domain to prove the efficacy of the ideas presented in this research work. In addition, a self-adaptive system has been selected as a system domain, and Android has been selected as an operating domain. The first objective aims to empirically analyze and organize the developer's existing knowledge about energy-saving solutions for location-based applications. The second objective aims to aid the domain analyst with an energy-aware modeling framework by extending the popular feature-oriented domain analysis framework. The third objective aims to develop a domain-specific modeling tool (*e*SAP) for the energy-aware modeling framework. The fourth objective aims to design and develop a tool named *e*GEN, which includes a textual domain-specific modeling language and automatic code generator. *e*GEN helps the domain analyst and developers specify energy-related requirements and generates battery-aware source code that can be used in the existing location-based Android applications. The efficacy of the energy-aware modeling framework and developed tools has been validated *qualitatively* using the *case studies* in software engineering. The obtained results show that the developed tools *e*SAP and *e*GEN help the domain analyst and developers reduce the development efforts for introducing energy-awareness and self-adaptivity in the early phases of *software development*.

Keywords: Energy-aware modeling framework; energy-saving self-adaptation planning; energy-saving code generator; energy-efficient software.

Contents

Abstract **ix**

Contents **xi**

List of Figures **xv**

List of Tables **xvii**

List of Listings **xix**

Contents **xix**

List of Abbreviations **xxi**

1 INTRODUCTION **1**
- 1.1 Motivation 2
- 1.2 Running Example: Location-based Applications 5
- 1.3 Research Objectives 6
- 1.4 Thesis Structure 8

2 PRELIMINARIES **11**
- 2.1 Smartphone Energy Inefficiencies 12
- 2.2 Energy-aware Self-adaptive Software 16
- 2.3 Modeling Frameworks 18
- 2.4 Model-driven Development 22
- 2.5 Feature-oriented Software Development 24
- 2.6 Summary 25

3 LITERATURE SURVEY **27**
- 3.1 Existing Empirical Studies 28

3.1.1 Stack Overflow data 28
3.1.2 GitHub data 29
3.2 Post-development Approaches 30
3.2.1 Energy profilers 31
3.2.2 Energy diagnosis engines 34
3.3 Pre-development Approaches 45
3.3.1 Feature modeling approaches 45
3.3.2 Model-driven development approaches 52
3.4 Research Gaps 57

4 Empirical Studies **59**
4.1 Research Questions and Methodology 60
4.1.1 Research methodology 61
4.1.2 Variable selection for empirical study 61
4.2 Controlled Experiments 62
4.2.1 Experiment protocol 62
4.2.2 Subject applications selection 63
4.2.3 Data collection and analysis 63
4.2.4 Answering RQ1: LBAs energy consumption 63
4.2.5 Answering RQ2: Inertial sensors usage and energy consumption 64
4.2.6 Answering RQ3: Inertial sensors usage by API 65
4.3 Mining StackOverflow 68
4.3.1 Mining protocol 68
4.3.2 Data collection and analysis 69
4.3.3 Answering RQ4: Energy-saving solutions 69
4.3.4 Answering RQ5: API usage patterns 73
4.4 Mining GitHub Commits 81
4.4.1 Mining protocol 81
4.4.2 Data collection 82
4.4.3 Data analysis 83
4.4.4 Answer to RQ6: Development efforts 84
4.5 Implications 87
4.6 Summary 88

5 ENERGY-AWARE MODELING FRAMEWORK **89**
5.1 Basics of Feature Modeling 90
5.2 Energy-aware Modeling Framework 92
5.2.1 Energy-aware context modeling 92
5.2.2 Energy-aware feature modeling 97

5.2.3 Energy-saving adaptation planning 100
5.3 Validation with a Case Study 102
5.3.1 An example scenario 102
5.3.2 Energy-aware context model 106
5.3.3 Energy-aware feature model 113
5.3.4 Energy-aware adaptation model 118
5.3.5 Discussions 120
5.4 **Threats to Validity** 122
5.5 Summary 124

6 ENERGY-SAVING ADAPTATION PLANNER **127**
6.1 Extending FeatureIDE 128
6.2 *e*SAP Description 130
6.3 Validation with Case Study 135
6.3.1 An example scenario: Reference requirements 136
6.3.2 Identifying energy-aware application configurations 137
6.3.3 Identifying valid triggering situations 141
6.3.4 Planning energy-saving adaptations 143
6.3.5 Discussions 145
6.4 Summary 147

7 ENERGY-SAVING CODE GENERATOR **149**
7.1 Domain Analysis 150
7.1.1 Identifying self-adaptive location strategies 150
7.1.2 Identifying commonalities and variabilities 154
7.1.3 *e*GEN domain model 159
7.2 *e*GEN Description 160
7.2.1 Technical background 162
7.2.2 *e*GEN grammar 162
7.2.3 Energy-aware code generator 167
7.3 Validation 168
7.4 Summary 171

8 CONCLUSIONS AND FUTURE WORKS **173**
8.1 Thesis Summary 173
8.2 Conclusion 177
8.3 Future Work 178

A PUBLICATIONS **179**

B BIO-DATA **181**

Bibliography **183**

List of Figures

4.1 Experiment protocol of controlled experiment. 62
4.2 Impact of sensor usage on overall battery drop. 65
4.3 Overall mining protocol of mining Stack Overflow. 69
4.4 Most common energy-saving solutions. 70
4.5 Energy-hungry (UP01) 74
4.6 Unregistering GPS (UP02) 74
4.7 Continuous GPS sensing (UP03) 75
4.8 Energy-efficient alternatives (UP04) 75
4.9 Checking if energy-efficient alternative is recent (UP05) 76
4.10 With sensing interval (UP06) 76
4.11 With alarm manager or scheduler (UP07) 78
4.12 With activity recognition (UP08) 78
4.13 Use of fused location API (UP09) 79
4.14 Adaptive location-sensing (UP10) 79
4.15 Overall mining protocol of GitHub commits. 81
4.16 Development efforts needed to make changes in source code. 85

5.1 Sample feature model. adapted from (Benavides et al., 2010) 91
5.2 Overview of energy-aware modeling framework 93
5.3 A basic feature model of map navigation application 103
5.4 Energy-aware context model of map navigation application 107
5.5 Energy-aware feature model of map navigation application 114
5.6 A sample energy-aware adaptation plan for valid context changes 119

6.1 Primary components and generated artifacts of *e*SAP 130
6.2 A screenshot of project manager that shows different types of projects . 131
6.3 A snapshot of assigning energy-aware labels 132
6.4 A screenshot of energy-saving adaptation plan editor 134
6.5 Energy-aware feature model of MapNav application 138
6.6 Energy-aware context model of MapNav application 143

7.1 *e*GEN eco-system 161
7.2 Improvement in battery saving 170
7.3 Degrade in accuracy 171

List of Tables

3.1 Summary of developer-centric finding approaches 39
3.2 Summary of developer-centric fixing approaches 43
3.3 Comparison of existing approaches with respect to specifying energy-awarenss and self-adaptation. 50
3.4 Comparison of model-driven development approaches for mobile app development 55

4.1 Selected subject applications for controlled experiment 63
4.2 Energy contribution of each LBA to overall energy consumption of device 64
4.3 Inertial sensor usage per API at run-time 66
4.4 Application active time vs GPS and sensors active time 67
4.5 Search strings and search results of commits search phase 82
4.6 Data collection results of mining GitHub commits 86

5.1 Application Configurations (AC) for deriving family of map navigation applications from basic feature diagram 105
5.2 Sample valid triggering situations of map navigation application 109
5.3 Summary of converting basic feature model to energy-aware feature model 111
5.4 Energy-aware application configurations of map navigation application . 117

6.1 Notations of energy-aware feature model 132
6.2 Notations of energy-aware context model 133
6.3 MapNav's functional requirements 136

7.1 Initial exploration of possible product categories (incomplete list) . . . 157

Listings

6.1 An energy-saving adaptation plan generated by *e*SAP representing ESA1 144
7.1 Domain model for modeling location-sensing 160
7.2 Structure of the features values 163
7.3 Structure of feature definition 164
7.4 Structure of the context values 164
7.5 Structure of the context constraints 165
7.6 Structure of the adaptation policy 166
7.7 An Excerpt from the egen model of subject application 2 169

List of Abbreviations

ABD	Abnormal Battery Drain
ADB	Android Debug Bridge
API	Application Programming Interface
DSL	Domain-specific Language
DSML	Domain-specific Modeling Language
eGEN	Energy-saving Code Generator
eSAP	Energy-saving Adaptation Planner
FODA	Feature-oriented Domain Analysis
FOSD	Feature Oriented Software Development
GPS	Global Positioning System
GPU	Graphics Processing Unit
ICT	Information and Communications Technology
LBA	Location-based Application
MDD	Model-driven Development
UML	Unified Modeling Language

Chapter 1

INTRODUCTION

The Green IT (Murugesan, 2008) research aims to reduce the negative impact of Information and Communication Technology (ICT) on the environment. In the last decade, research in Green IT was more on hardware-based techniques than software-based techniques (Mahmoud and Ahmad, 2014). The software aspects of Green IT have not been investigated thoroughly because of their virtual nature and indirect impact on the environment (Mahmoud and Ahmad, 2014). A study by Penzenstadler et al. (2014) highlights the importance of considering greenness and sustainability as important non-functional requirements like safety and security. Unfortunately, greenness and sustainability are not considered in the traditional software life cycle process. Hence, to address this issue, a new research field called *green and sustainable software engineering* (Kern et al., 2015) has emerged recently. It is defined as *the art of designing, developing, using, and disposing of the long-lasting and energy-efficient software to reduce the negative impact on the environment*. In this thesis, the *greenness* aspect of the software is considered, and it refers to the software systems' energy-efficiency. Energy-efficiency is not a new terminology to the software research community. Researchers have previously proposed several notable energy optimization techniques as part of source code optimization, especially for embedded software. Some of the code optimization techniques are cache skipping, use of register operands (Bellas et al., 2000), instruction clustering (Naik, 2010), instruction re-ordering and memory addressing (Su et al., 1994), loop optimizations (Delaluz et al., 2000), eliminating recursion (Mehta et al., 1997), etc. All these techniques require knowledge about low-level details and complex data structures to achieve energy-efficiency. In practice, all application developers might not be aware of all energy-saving techniques. Therefore, developers with less background knowledge find it difficult to write energy-efficient embedded applications. It is even more challenging in smartphone devices, as badly written application code may lead to a shortened battery life. A possible solution to address this issue would be to consider energy-efficiency from the initial stages instead of considering it later in the software development life cycle. Therefore, this thesis aims to consider energy-efficiency as an essential non-functional requirement in the early stages of software development. Specifically, it aims to consider the energy-efficiency related requirements in the domain analysis phases of smartphone apps.

1.1 Motivation

Energy-efficiency is one of the important software quality parameters (Pinto and Castor, 2017) in smartphone devices. The empirical studies such as Pinto et al. (2014), Moura et al. (2015), and Manotas et al. (2016) highlight the importance of reducing software energy consumption while developing the software. Specifically, the studies Li et al. (2014), Bao et al. (2016), and Chowdhury et al. (2018) report the role of developer written code on the overall energy consumption of Android applications. As reported in these studies, the improper usage of energy-hungry components such as GPS (Kim et al., 2016b), Display (Wan et al., 2017), GPU (Kim et al., 2016a) and mobile data (Li et al., 2014) consumes a lot of energy and drains the battery quickly. Researchers typically refer to this situation as "Abnormal Battery Drain" (ABD) (Ma et al., 2013). In the literature, it was reported that "*unnecessary resource usage, faulty GPS behavior, background activities, advertisements, and high GPU usages*" (Pang et al., 2016, Pinto et al., 2014) were the primary causes of abnormal battery drain issues in smartphone devices. These ABD issues could be solved by adopting well-defined battery-aware practices (Georgiou et al., 2019) while developing the software. A popular solution to avoid ABD is to avoid *wakelock bugs and unnecessary usage of energy-hungry components* Pathak et al. (2011) in Android applications. In a recent study conducted by Cruz and Abreu (2018), and Morales et al. (2018), it is reported that optimizing the object-oriented code smells and Android-specific code smells that contribute to the energy consumption of Android applications would be a better energy-saving solution.

Over time, the researchers have adopted software engineering solutions such as program analysis techniques (Liu et al., 2014), automated bug detection techniques (Morales et al., 2018, Zhu et al., 2019), automated software repair (Banerjee et al., 2018), energy-efficient programming practices (Sahar et al., 2019) and refactoring (Banerjee and Roychoudhury, 2016) to identify and fix the known energy bugs in Android applications. In addition, few researchers have utilized the concept of self-adaptivity to provide suitable solutions for energy-efficiency of Android applications (Cañete et al., 2020, Datta et al., 2014, Mizouni et al., 2012, Moghimi et al., 2012, Ortiz et al., 2019). The researchers have used the battery information to decide the run-time adaptation that may lead to energy-saving.

Though self-adaptive solutions are proposed in the literature for energy-efficiency, developing such self-adaptive behavior is difficult for the developers (Krupitzer et al., 2015) as it has to deal with a dynamically changing environment. Specifically, identifying suitable context information for energy-saving adaptation is not a straight forward task (Capurso et al., 2018). Self-adaptive software depends on sensing the contextual changes and dynamic adaptation policies (Cheng et al., 2009a). Typically, the requirements of self-adaptive systems are uncertain, and it has to be managed at a higher level of abstraction (De Lemos et al., 2013). In the literature, there were several research efforts (Yang et al., 2017) on requirements engineering and analysis of the self-adaptive software requirements. Specifically, several methods (Ahmad et al., 2012, Brown et al., 2006, Cheng et al., 2009b, Inverardi and Mori, 2011, Zhuo-Qun and Zhi, 2012) have been found for requirements analysis of self-adaptive software. Overall, these methods use goal-modeling methods such as KAOS (Dardenne et al., 1993), i* framework (Franch et al., 2016), Tropos (Bresciani et al., 2004), and feature models (Kang et al., 1990) for requirements specification and analysis of self-adaptive software systems. Therefore, the *feature model* has been adopted in this research work to specify and analyze the energy-related requirements for deciding energy-saving self-adaptation. In the literature, several studies (Asadi et al., 2014, Desmet et al., 2007, Inverardi and Mori, 2010, Mauro et al., 2018, Pascual et al., 2015a, Siegmund et al., 2012, Soltani et al.,

2012) have been found that adopts feature models for analyzing the self-adaptive software. But none of the existing methods have focused explicitly on the energy-related requirements, which are highly critical in battery-powered devices.

As highlighted in Morales et al. (2018), *reducing battery consumption* is one of the essential quality factors of smartphone apps as it runs on limited-resource devices. The existing energy-efficient solutions for smartphone applications primarily work at the development and testing phases of the software development life cycle. Overall, existing solutions suggest ways to improve coding practice for energy-efficiency, demanding developers to learn additional energy-optimization skills. In combination with the development of self-adaptive behaviors, the introduction of energy-aware requirements directly at the development phase would increase the development efforts if the developers are unaware of the energy-saving practices. A possible solution to this problem is to consider the energy-efficiency and self-adaptive requirements early to reduce the development efforts in the development and testing stages. The following research questions arise while considering energy-related requirements early in the subject application development life cycle:

- **RQ1**: What solutions do developers apply to reduce the energy consumption of commonly used energy-hungry components?
- **RQ2**: Is it possible to consider energy-efficiency as an essential non-functional requirement in the early stages of software development?
- **RQ3**: What is necessary to support domain analysts with suitable tool support to consider energy-efficiency in the early stages?
- **RQ4**: Is it possible to support developers by transforming design-time models to source code for reusable components to reduce the development efforts?

The research questions mentioned above motivate this thesis to consider energy and self-adaptive requirements early to reduce the complexity in the development phase. Hence, this research primarily focuses on incorporating energy-efficiency as a critical non-functional requirement during the early stages before making any lower-level code development decisions.

1.2 Running Example: Location-based Applications

A family of location-based applications is selected as an application domain as location-sensing is an energy-hungry requirement in smartphone apps. In recent years, there is a significant increase in the usage of location-based services in mobile apps. These apps drain the smartphone battery faster if it uses the Global Positioning System's (GPS) location-sensing continuously. GPS provides high accuracy of user location compared to other location-sensing techniques such as Wi-Fi positioning (Choi et al., 2017) and Cell ID based positioning (Ibrahim and Youssef, 2012). One downside of GPS is its abnormal amount of energy consumption for its operation. Therefore, GPS usage must be reduced (Paek et al., 2010) when the smartphone is running on a low battery. In the literature, several research works (Capurso et al., 2017, Kim et al., 2016b, Morillo et al., 2012) use self-adaptive strategies to manage the energy consumption of location-sensing in smartphones. The adaptive strategies dynamically select a suitable location-strategy based on the dynamic accuracy and energy requirements. In addition to academic research efforts, the official Google Location APIs continuously evolve to improve location-sensing energy-efficiency. For instance, Google location APIs such as *Fused Location Provider API*, *Fused Location Provider Client* are evolving in every release of API with new energy-saving methods.

The abnormal battery-draining behavior of GPS makes location-based applications as the best candidate for conducting case studies in this thesis. The application of location-sensing is found in almost all types of smartphone applications. Specifically, the following category of smartphone applications uses location to provide user-friendly services: *Navigation, Fitness, Social Networking, Game, Automotive, Travel, and Advertising*. In this thesis, these application categories are considered as a family of location-based applications to conduct the case studies presented in other chapters. Overall, location-sensing is a common feature among the application categories, though the core business logic differs. As this application domain shares commonality among the family of location-sensing applications, it is suitable to provide a domain-specific solution for the commonly occurring problem. Location-sensing is the process of fetching the user locations from an available location-sensor. Typically, the most widely used location-sensor for outdoor smartphone applications is GPS. However, alternative location-sensing sources such as Wi-Fi, Bluetooth, and Cell ID are also found in a few smartphone applications to reduce GPS usage. Furthermore, the recent generation of smartphone uses inertial sensors such as accelerometer, gyroscope, magnetometer, etc., to reduce GPS usage. Generally, the location-sensing source code is not written by the developers from scratch as it has to handle many location-sensors. Therefore, the developers have used popular APIs such as Android native location API, Google Location Service API, and Open Street Map API. However, using location APIs in an energy-efficient way is the responsibility of the application developer. Unfortunately, the abnormal battery-draining behavior of GPS makes it difficult for the new developers to decide battery-optimization at the development phase directly. It would be beneficial to new developers if battery-related requirements are considered from the early phases. It will also help new developers if location-sensing source code with battery-optimization is generated from the design phase artifacts. Therefore, the developers can spend more time on improving the core business logic in the development phase. This thesis considers reducing the energy consumption of location-sensing as a non-functional requirement, and it is common for all application categories. Therefore, providing an energy-saving solution for one application category could be reused in the other application categories as well. With a brief analysis of the application domain, the location-based applications have been selected to illustrate this thesis's contributions.

1.3 Research Objectives

The primary objective of this research work is to *assist the design and development of energy-aware self-adaptive software through appropriate domain-specific modeling language and automatic code generator*. The solutions provided in this thesis might help the domain analyst and developers to explore the different energy-saving opportunities at design time to reduce the developer efforts. The main research objectives of this research work are given below:

RO1 To empirically analyze and organize existing knowledge about energy-consumption and energy-saving solutions of the application domain.
Here, the domain of *Location-based Android applications* is selected as an application domain to conduct empirical studies. This objective aims to find out

the cause-effect relationship between location sensors and their impact on energy consumption. This objective also aims to organize the existing knowledge of energy-saving solutions and API usage patterns from Stack Overflow discussions. In addition, an analysis of GitHub commits has been performed to know the development efforts invested by the developers to improve the apps energy-efficiency after deploying the application.

RO2 To propose a suitable domain analysis framework for designing Energy-aware Self-adaptive Software.
This objective aims to propose a modeling framework to conduct energy-aware domain analysis of software under development. It provides a classification mechanism to classify the application's context and features, at the early stages of software development. An extension to Feature-oriented Domain Analysis Framework (FODA) (Kang et al., 1990) has been provided to aid the domain analyst to conduct the domain analysis of energy-aware self-adaptive software.

RO3 To design and develop a Domain-specific Modeling Language and a tool support for modeling energy-aware self-adaptive software.
This research objective aims to develop tool support for domain analysts, which would be used in the domain analysis phase. The tool support has been developed on top of FeatureIDE (Thüm et al., 2014), so that it allows to plan platform-independent energy-saving adaptations for energy-aware self-adaptive software.

RO4 To design and develop an automatic code generation tool for developing energy-aware self-adaptive software.
The new developers may face difficulty in introducing self-adaptive behavior along with energy-awareness directly at the code level. The development efforts would increase further, if the energy-aware code is being developed from scratch. Therefore, to reduce the developer's efforts, this objective aims to provide a textual Domain-Specific Language (DSL) for adaptive location-sensing. Finally, the code generation tool might automatically generate energy-aware self-adaptive software artifacts reused in the family of location-based applications.

1.4 Thesis Structure

This thesis is structured into eight chapters, excluding the reference section and appendix. This section provides an overview of each chapter to guide the readers.

- **Chapter 1** starts with a brief overview of the research field "*Green and Sustainable Software Engineering*". Then, it presents the research questions and motivation to conduct this research work. It also highlights the four research objectives and thesis contributions. In addition, it mentions the target audience and the running example of location-based smartphone applications (application domain) to show the efficacy of the methodology and tools presented in this thesis.
- **Chapter 2** presents the fundamentals relevant to the concepts presented in this thesis. This chapter begins with a summary of energy-related issues specific to smartphone applications and existing solutions. Then, it provides an overview of self-adaptive software as it is one of the important enabling technologies of the concepts presented in this thesis. Furthermore, this chapter presents a brief overview of feature-oriented software development and model-driven software development.
- **Chapter 3** summarizes the related research works found in the literature. It begins with summarizing the empirical studies involving the energy consumption of smartphone applications. Then, it describes the post-development research efforts to reduce the energy consumption of smartphone applications. Further, it compares the pre-development approaches that use feature modeling for specifying self-adaptive requirements. Finally, it summarizes the research works relevant to the model-driven development of smartphone applications.
- **Chapter 4** is dedicated to present the empirical studies conducted on location-based Android applications. Three data sources have been considered for empirical studies: *Controlled Experiments, Stack Overflow, and GitHub*. This chapter describes the research questions, data collection process, data analysis methods, and answers to considered research questions. Overall, this chapter presents the

relation between location-sensors and energy consumption, energy-saving strategies used by expert developers, and development efforts needed for optimizing location-sensing for energy-efficiency.

- **Chapter 5** introduces the concept of an energy-aware modeling framework for specifying energy-saving self-adaptive plans. This chapter presents the extended feature modeling rules for energy-aware context-modeling and feature-modeling. Specifically, it explains the classification method for energy-aware context and energy-aware features. Finally, it shows the presented modeling framework's efficacy with a case study on location-based smartphone applications.
- **Chapter 6** presents the tool support developed for the energy-aware modeling framework presented in Chapter 5. This chapter begins with an overview of using the tool Energy-saving Adaptation Planner *e*SAP. Then, it explains the notations for context and feature categories of the energy-aware modeling framework. Furthermore, it presents the validation and discussions through a case study on map navigation applications.
- **Chapter 7** describes the implementation of a tool named *e*GEN. First, it describes location-based applications' domain analysis to identify the common features to be included in the domain-specific modeling language and code generator. Then, it explains the domain model and grammar of textual domain-specific modeling language of *e*GEN. Finally, the artifacts generated by *e*GEN has been validated through a suitable case study on the activity tracking Android applications.
- **Chapter 8** summarizes the thesis contribution. It concludes the thesis with the results of the experiments and case studies conducted. Furthermore, it enumerates and highlights future research directions to motivate prospective researchers.

Chapter 2

PRELIMINARIES

This chapter presents the background information relevant to this thesis. The concepts presented in this thesis considers the energy-efficiency requirements as essential non-functional requirements for smartphone apps. In this thesis, Android platform is considered as operating domain, and energy-efficiency related issues and existing solutions are highlighted in Section 2.1. One of the significant underlying concepts of this thesis is introducing self-adaptivity in software. The underlying concepts of self-adaptive software are described in Section 2.2, along with few mentions about existing self-adaptive solutions for energy-efficient smartphone apps. Furthermore, the enabling technologies underlying in this thesis, namely Model-driven Software Development (MDD) and *Feature-oriented Software Development (FOSD), are described in Section 2.4 and 2.5 respectively.*

2.1 Smartphone Energy Inefficiencies

This thesis aims to consider the energy-efficiency of smartphone applications as one of the essential non-functional requirements. Hence, discussing the smartphone's energy-inefficiencies would add more clarity to the contents presented in this thesis. There are different categories of energy inefficiencies reported in the literature for smartphone applications: *energy bugs, energy leaks, energy hogs, and energy hotspots*. Energy bugs are referred to as an error in the system, either application, OS, hardware, firmware, or external, that causes the system's unexpected energy consumption as a whole (Pathak et al., 2012b). Energy leaks are the form of energy inefficiencies where the energy consumed by the actions never produced any useful output to the system (Zhang et al., 2012). Energy leaks are also known as energy waste, as reported in Kim and Cha (2013), Wang et al. (2014). Energy hog refers to the situation where the application's average discharge rate during the running state will be significantly higher than the average discharge rate when the application is not running (Oliner et al., 2013). Energy hotspots are the places that have the potential to save a significant amount of energy (Wan et al., 2015). In other words, the execution of energy hotspots results in an abnormal amount of battery drain even if the hardware resources are utilized significantly less (Banerjee et al., 2016). This section describes the energy inefficiencies caused by programming errors and mishandling of Android APIs.

Wakelock related issues

Wakelock is one of the *PoweManager* system service features in Android. Wakelock keeps the necessary system resources running and prevents the device from entering the sleep state (Kim and Cha, 2013). The developers can control the device's power state by adding `WAKE_LOCK` permissions in the manifest file (Liu et al., 2016). There are several types of wakelocks (Wang et al., 2014): `PARTIAL_WAKE_LOCK`, `SCREEN_DIM_WAKE_LOCK`, `SCREEN_BRIGHT_WAKE_LOCK`, `FULL_WAKE_LOCK`, `etc`. However, few wakelock types are depreciated in the recent versions of the Android operating system. These wakelock types have different wake levels and power consumption on the following system resources: *CPU, Screen, Keyboard*. Any wakelock activated has to be deactivated after finishing the execution of a particular task. Few buggy applications fail to release the acquired wakelock, and results in unnecessary energy consumption. Pathak et al. (2012b) referred this situation as *no sleep energy bug*. In ELITE (Liu et al., 2016), the following wakelock misuse patterns are discussed: *unnecessary wakeup, wakelock leakage, premature lock releasing, multiple lock acquisition, inappropriate lock type, problematic timeout setting, inappropriate flags, and permission error*. These wakelock misuse patterns are considered to be one of the popular energy inefficiencies in the Android platform.

Sensor related issues

Usage of smartphone sensors is an energy-hungry operation in smartphones (Carroll and Heiser, 2010). Sensors with faster sensing rates consume more power. There are two classifications of sensor-related energy-inefficiencies (Liu et al., 2014): (1) missing sensor deactivation, and (2) sensory data underutilization. The registered sensor listeners must be unregistered when the application no longer needs sensor data. Failed to unregister the sensor listener referred to as *missing sensor deactivation*. This type of energy inefficiency leads to unnecessary sensing operation and corresponding unwanted power consumption. On the other hand, *sensory data under utilization* refers to a situation where context information provided by the sensors is unused and results in abnormal battery drain. For instance, the sensing rate of the sensing-related game must be changed at run-time according to the following context information of accelerometer: MOVING and MOTIONLESS (Li et al., 2015). Therefore, the sensor APIs must be carefully used while setting the sensing interval and registering the sensor listener to avoid unwanted battery drain.

Graphics related issues

Graphics-intensive mobile apps such as video players and games keep the screen active during its execution (Jabbarvand and Malek, 2017). In recent Android platforms, `FLAG_KEEP_SCREEN_ON` keeps the screen on, during the foreground execution of the application. As mentioned in Google developer web page[1] this flag must be used inside an

[1]https://developer.android.com/training/scheduling/wakelock#screen

activity rather than programming components. Unfortunately, few developers use this screen flag inside services and results in abnormal battery draining. On the other hand, the GPU usage by games to render the graphics has increased for recent generation games (Kim et al., 2016a). The unnecessary GPU usage in graphics-intensive mobile applications may result in an abnormal battery drain. The following situations are identified as the energy-hungry graphics operations (Kim et al., 2016a): (1) repeated texture transfer inside render loop, (2) transferring multiple smaller images to GPU, and (3) sending identical frames for rendering. The developers must be aware of best practices to address these issues to avoid unwanted battery draining.

Connectivity related issues

The connectivity components such as GSM, WiFi, and Bluetooth usage also contributes to the energy consumption of the Android application. Improper usage of few connectivity components results in abnormal energy consumption. In TIDE (Dao et al., 2017), the authors have suggested that energy consumption behavior depends on the type and quality of the network used by the application. For instance, downloading a large amount of data must be suspended when a user is on a slow network connection . Therefore, by adjusting the application behavior based on the type of network at runtime, unwanted battery consumption can be avoided. In ADEL (Zhang et al., 2012), the authors have highlighted several root causes of network-related energy inefficiencies. It includes *misinterpretation of call back APIs, inefficient data refreshing, repetitive downloads, and aggressive prefetching*. Misinterpretation of callback APIs refers to wasteful downloads due to a separate download thread after closing the main thread. Inefficient data refreshing behaviors ignore the device and app state before initiating any network operations. Repetitive downloads cause energy inefficiency as the energy is spent downloading the same data every time instead of caching it. The aggressive prefetching mechanism uses the prediction mechanism to download the data used by the user in the future. Sometimes, downloaded data by a prefetching application might not be used by the user and results in energy waste. Similar to wakelock related issues, these connectivity components must be closed when the app is in the background or closed by the user to avoid unwanted energy consumption.

The energy inefficiencies discussed so far are about the impact of programming errors or mishandling of APIs on Android applications' energy consumption. Besides these inefficiencies, a few issues related to layouts, user behaviors, and android-specific code smells have also been found. The layout defects (Cruz and Abreu, 2018) refer to the energy inefficiency present in the UI definition of the Android applications. In literature, the impact of different user behaviors also proved to one of the contributors to energy inefficiency (Abbasi et al., 2018, Dao et al., 2017). In recent years, several researchers investigated the energy impact of object-oriented and Android-specific code smells on Android applications' energy consumption. In recent studies (Carette et al., 2017, Morales et al., 2018), the following object-oriented smells are reported to be energy-hungry: *blob, lazy Class, long-parameter list, refused bequest, speculative generality*. Besides, the Android-specific anti-patterns such as *binding resources too early, HashMap usage, private getters, and setters* also reported as energy-hungry anti-patterns. In a study conducted by Palomba et al. (2019), the following code smells are considered to be energy-smells: *leaking thread, member ignoring method, slow loop, data transmission without compression, and internal setter*. Overall, the energy inefficiency studies are slowly shifting from resource-specific investigation to code smells investigation. An Android application must be free from the program-specific energy inefficiencies before deploying it on the user devices. Identifying these wide varieties of energy inefficiencies becomes a significant challenge in the new generation of smartphone application development.

Traditionally, energy profilers (Ahmad et al., 2015) were used by the developers and the researchers to reduce the negative impacts caused by the energy bugs. An energy profiler is a tool that provides sub-component-wise and app-wise energy consumption values of smartphone devices (Hoque et al., 2016). These tools estimated the power consumption through a well-defined power model, which considers various sub-components of a mobile device (Hoque et al., 2016). Some of the examples include Trepn Profiler, Powerbooter (Zhang et al., 2010), Sesame (Dong and Zhong, 2011), DevScope (Jung et al., 2012), AppScope (Yoon et al., 2012), eProf (Pathak et al., 2012a), eLens (Hao et al., 2013), and Google Battery Historian. However, the developer needs more technical knowledge to locate the code segments responsible for abnormal battery drain. Hence, in recent years, the research community shifted its focus on adopting automated software engineering methods to solve the energy-related issues at the code level. The adopted software engineering methods include program analysis (Jiang et al., 2017, Liu et al., 2014), software bug-localization (Banerjee and Roychoudhury, 2016), refactoring (Gottschalk et al., 2016), mutation testing (Jabbarvand and Malek, 2017), and automated software repair (Banerjee et al., 2018) to address the identified energy-related issues. Overall, these tools are used to automatically locate the code segments that are likely to have buggy code segments that affect the device's energy-efficiency. Researchers have proposed tools to automatically repair the buggy code segments in recent years to make them energy-friendly. Overall, the current research works provide solutions at the development or testing phases of the software development life cycle.

2.2 Energy-aware Self-adaptive Software

In this thesis, the term "*energy-aware self-adaptive software*" refers to software that applies self-adaptive behavior to save energy at run-time. The concept of self-adaptive software is not new to the software engineering research community. A self-adaptive software (SAS) can automatically change its behavior according to changes in its operating environment (Kephart and Chess, 2003, Oreizy et al., 1999). Self-adaptive behavior development depends on basic system properties such as self-awareness and context-awareness (Salehie and Tahvildari, 2009). Self-awareness describes the ability of a system to be aware of itself, i.e., to monitor its resources, state, and behavior (Hinchey and Sterritt, 2006). Context-awareness means that the system is aware of its operational environment, the so-called context (Schilit et al., 1994). According to (Dey, 2001) any information can be considered a context, if it is used to characterize the situation of an entity. An entity can be either a user or an application or an application or an interacting user's operating environment. In general, context information can be classified under four categories: computing, user, physical (Schilit et al., 1994) and time (Chen and Kotz, 2000) context. *Computing context* can be processors, memory, network bandwidth, and other nearby computing resources. *User context* can be the location, user profile, nearby people, current situation. *Physical context* can be lightning, noise level, temperature, etc. *Time context* can be time, day, week, month, year. All this context information might not be relevant to a single application. The nature of the application decides the type of relevant context information. For instance, user location may play a major role in location-based services, whereas it may be irrelevant in other applications. Therefore, it is important to select relevant context information for the desired context-awareness in context-aware systems.

Self-adaptive behaviors are introduced in software systems to satisfy non-functional requirements such as maintainability, functionality, availability, portability, and usability (Salehie and Tahvildari, 2009). The energy is limited in smartphone devices, and extending battery life becomes a primary non-functional requirement(Cañete et al., 2020). Several self-adaptive approaches exist in the literature (Cañete et al., 2020, Datta et al., 2014, Mizouni et al., 2012, Moghimi et al., 2012, Ortiz et al., 2019) to address the energy-related issues in smartphone applications. These approaches aim at changing the application behavior at run-time towards extending the battery life of the smartphone. Specifically, the results presented in the recent study (Cañete et al., 2020) show that self-adaptation would be a suitable solution for addressing energy-related issues in smartphone applications. Furthermore, several research works found in the literature apply self-adaptive strategies for energy-savings in smartphone applications' location-sensing. Notably, Zhuang et al. (2010) proposed an adaptive location-sensing framework that uses substitution, suppression, piggybacking, and adaptation. Here, *substitution* uses the alternatives to GPS; *suppression* low-power intensive sensors such as accelerometer; *piggybacking* uses the collaborative strategies; and *adaptation* adjust the sensing interval to save energy at run-time. In this approach, all the other previous approaches are used in an adaptive way to improve the energy and accuracy requirements. A-Loc (Lin et al., 2010) automatically manages location sensor availability, accuracy, and energy by selecting the most suitable location-sensing mechanism at run-time. Another approach (Morillo et al., 2012), dynamically adapts between GPS, WiFi-based localization, and accelerometer at run-time based on the user's outdoor exit detection to improve the GPS efficiency. In Virtual GPS (Thokala et al., 2014), a middleware chooses a suitable location strategy (GPS or WiFi or Cell ID) for a given accuracy requirement with minimum energy consumption. Kim et al. (2016b) uses the context information such as the category of applications executed, the user's movement pattern, and the battery level to select the suitable location detection scheme (GPS or Cell-tower). In recent research work, Capurso et al. (2017) proposed indoor-outdoor detection techniques to switch between GPS and other indoor-localization methods to improve the location-based energy efficiency applications. Typically, these approaches consider the battery level of the smartphone applications to decide on energy-saving adaptation.

Though self-adaptive solutions are proposed in the literature for energy-efficiency, developing such self-adaptive behavior is difficult for the developer (Krupitzer et al., 2015). Specifically, identifying relevant context information for energy-saving adaptation is not a straight forward task (Capurso et al., 2018). It involves sensing, learning, and acting upon the raw data produced by the smartphone sensors (Capurso et al., 2018). In addition, planning the adaptation strategies needs a deep understanding of the problem domain and the affected software quality factors. Therefore, analyzing the requirements and context changes before development phases is essential to develop suitable energy-saving self-adaptive smartphone applications. Therefore, having a dedicated requirements analysis phase in place for self-adaptation planning for energy-efficiency is highly recommended (Yang et al., 2017).

2.3 Modeling Frameworks

This section presents the popular modeling frameworks widely adopted for modeling self-adaptive software.

KAOS

Knowledge Acquisition in autOmated Specification (KAOS) (Dardenne et al., 1993) is a requirements engineering methodology that allows analysts to construct requirements models and derive requirements documents from KAOS models. The KAOS has the following components: *the conceptual model, acquisition strategies, and the acquisition assistant*.

- *The conceptual model* is a meta-model associated with a requirements acquisition language for acquiring and structuring requirement models. Unlike other formal requirement languages, KAOS allows capturing both functional and non-functional requirements of the system under consideration. The KAOS meta-model involves three levels of abstractions: *meta level, domain level, and instance level*. The *meta level* is a domain-independent abstractions that consists of *meta-concepts, meta-relationships, meta-attributes, and meta-constraints*. The *instance level* is considered to be the instance of the domain level concepts.

- *Acquisition strategy* consists of steps to create requirements models from the instances of meta-model components. It helps the analyst to traverse the meta-model graph with associated nodes and relationships. This strategy allows traversing the meta-model either forward or backward. The acquisition strategy steps might contain the following finer steps, such as question-answering, input validation, deductive inferencing, analogical inferencing, conflict resolution, etc.

- *Acquisition assistant* provides automated support to the analyst in applying acquisition strategies. It depends on two components: *requirements database and requirements knowledge base*. The requirements database contains all the requirement models acquired by the analyst so far. The requirements knowledge base consists of application domain-models and meta-level knowledge.

KAOS's requirements model considers the critical system components, such as goals, agents, and alternatives. Overall, this approach defines an initial set of goals, agents, and their actions.

i* Framework

i* (Yu, 1997) framework is a system modeling approach used to understand the problem domain in the early requirements engineering phase. In the seminal work of i* (Yu, 1997), the authors have stated that the early requirements engineering could be supported with modeling stakeholders interest while later requirement engineering phases supports completeness, consistency, and automated verification of requirements. The i* approach adopts the concepts of actor-oriented modeling and goal modeling which were predominantly used for modeling information systems of organization environment. This framework consists of two modeling components: *Strategic Dependency model and Strategic Rationale model*.

- *Strategic Dependency (SD) model* captures the dependency relationship between the actors in the organization environment. The model primarily uses the notations for *actor* and *dependency* among them. The node of the SD model refers to the actor of the system and the link in the SD model refers to the dependency relationship. In this model two types of actors have been used namely, *depender, and dependee*.
- *Strategic Rationale (SR) model* captures the stakeholders' concerns, interest, and ways to satisfy them. This model specifically shows how an actor could achieve their goals and soft goals. Formally, the SR model specifies the *goals, soft goals, task, and resource* of the system. In contrast to SD model, the SR model gives detailed information on why an actor would involve in using a resource to complete the task, goals, and soft goals to express the stakeholder's interest and concern.

In addition, this framework supports the analysis and reasoning for ability, workability, viability, and believability of the systems requirements in the early phases. Over time, the researchers have extended i* framework to support different needs in the requirements engineering phase (Franch et al., 2016). Several notable extensions of i* frameworks are social modeling Tropos (Bresciani et al., 2004), (Eric, 2009), and goal-oriented requirements modeling (Surhone et al., 2010). The recent modified versions of i* framework have been used for specifying the non-functional requirements through goals and soft goals elements of the SR model.

Feature models

Feature models are originally introduced by Kang et al. (1990) in the Feature-Oriented Domain Analysis (FODA) report (1990). Since then, feature models are considered to be an important information model widely used in SPL engineering. A *feature model* is a formal way of modeling the commonalities and variabilities of SPL (Batory, 2005, Kang et al., 1998). A feature model represents the set of features, relationships, and constraints among them. A *feature* can be formally defined as user-visible property (Kang et al., 1990) of the system. Feature models can be graphically represented through a tree-like diagram called *feature diagram*. It represents the relationships among features through parent-child relationships. The following relationships are the part of basic feature models (Benavides et al., 2010) *mandatory, optional, alternative, and OR*.

- *Mandatory* relationship signifies that the child feature must be included in all the product configurations if its parent feature is included.
- *Optional* relationship signifies that the child feature presence is optional even if its parent feature is added into the product configuration. Hence, the child feature with *optional* relationship may or may not be included in all the products.
- *Alternative* relationship signifies that exactly one child feature from a set of child features can be added to the product configuration if its parent is part of the product.
- *OR* relationship signifies that one or more child features can be included in the product configurations along with its parent feature.

These basic relationships between parent and child features are sufficient to capture all interdependencies among all features. Apart from these basic relationships, the crosstree constraints must also be specified in the feature diagrams. The following are the two basic constraints to capture the interdependencies or cross tree constraints of a feature model: *Requires and Excludes*.

- *Requires* constraint signifies that the inclusion of particular feature requires inclusion of other related features also.
- *Excludes* constraint signifies that the inclusion of a particular feature must discard or exclude the other irrelevant features from the product configuration.

Overall, the approaches for modeling non-functional requirements of self-adaptive software use goal-modeling methods such as KAOS (Dardenne et al., 1993), i* framework

(Franch et al., 2016), and feature models (Kang et al., 1990). However, none of the existing methods have specifically focused on the energy-related requirements, which are highly critical in battery-powered devices. Among these modeling frameworks, *feature model* have been adopted in this research to introduce the proposed concepts because of its expressiveness, automated reasoning support, and formalism. Therefore, in this research work *feature model* have been adopted to specify and analyze the energy-related requirements for deciding energy-saving self-adaptation.

2.4 Model-driven Development

Model-driven Development (MDD) is a software development paradigm that aims to reduce software complexity and improve software quality by enabling developers to work at a higher level of abstraction (Hailpern and Tarr, 2006). The notable characteristic of MDD is that it primarily focuses on models instead of programs (Selic, 2003). The models aid the domain analyst or developers to specify the problem domain without bounding to the actual programming language used for implementation. The models are easier to specify, understand, and maintain even by the domain experts without help of developers written code (Selic, 2003). The term Model-driven Development is interchangeably used in the literature with the following terms: *Model-driven Engineering (MDE) (Schmidt, 2006), Model-driven Architecture (MDA) (Kleppe et al., 2003), Model-Driven Software Engineering (MDSE) (Brambilla et al., 2017), and Model-driven Software Development (MDSD)*[2]. All these terms refer to the same concepts as that of Model-driven Development. Therefore, in this thesis, the term "Model-driven Development (MDD)" is used to refer to the concept of using models for the development of energy-aware self-adaptive software.

According to MDA concepts provided by Object Management Group (OMG) [3], the models can be broadly classified into *Computation-independent Model (CIM), Platform-independent Model (PIM), and Platform-specific Model (PSM)* (Truyen, 2006). The *Computation Independent Model (CIM)* is a business requirement or domain model which is not linked with the underlying technology stack. The *Platform Independent*

[2]https://martinfowler.com/bliki/ModelDrivenSoftwareDevelopment.html
[3]https://www.omg.org/

Model (PIM) refers to the model that is built with formal notations during the design phase that is independent of the implementation platform of the software. The *Platform Specific Model (PSM)* refers to a model that is more close to the programming language in which the system will be implemented. It includes many details about libraries, programming elements, database, interfaces that will be used in the development of the software system. The concept model transformation is used in MDA predominantly to transfer from CIM to PIM and PIM to PSM. Generally, MDA uses UML (France et al., 2006) and meta-modeling (Atkinson and Kuhne, 2003) concepts to define Platform-independent models and CASE tools for model transformation.

In recent years, the Domain-specific Modeling Language (DSML) and automatic code generators are widely used in MDD for improving the productivity of the developers. The Domain-specific Modeling Language (DSML) is a Domain-specific Language (DSL) tailored to a specific problem and application domain that have the modeling element corresponding to domain-specific terms (Frank, 2013). The automatic code generators use model-to-text transformation to convert the models produced by DSML to a source code (Kelly and Tolvanen, 2008). The DSML could be both graphical and textual. The models developed using the DSML could be considered as platform-independent models. Typically, these models undergo a model-model-transformation to produce the platform-specific models (, Völter). Finally, the platform-specific models could be converted to source code using model-to-text transformation (Kelly and Tolvanen, 2008). The model-driven development approaches in practice aim to provide a CASE tool that automates the process of modeling and transforming the model to source code. These tools help to reduce the development efforts and increase the productivity of the developers. In industries, The tools like MetaEdit+[4], and Xtext[5] are the popular language workbenches for defining DSMLs. The tools like ATL[6], Acceleo[7], and Xtend[8] are widely used for defining a code generator. In research, the DSML and code generators are used for domain such as context-aware web applications (Ceri et al., 2007), Concurrent Programming (Marand et al., 2015), cyber-physical manufacturing systems (Thramboulidis and Christoulakis, 2016), smart service systems

[4]http://www.metacase.com/products.html
[5]https://www.eclipse.org/Xtext/
[6]http://eclipse.org/atl/
[7]https://eclipse.org/acceleo/
[8]https://www.eclipse.org/xtend/

(Beverungen et al., 2019), etc. In the literature, several studies have been found that use MDD concepts for mobile app development (Tufail et al., 2018, Vaupel et al., 2018) and self-adaptive software (Vogel and Giese, 2014). In this thesis, the MDD concepts have been adopted for modeling of self-adaptive behaviour with energy-saving as one of the essential goal.

2.5 Feature-oriented Software Development

Feature-Oriented Software Development (FOSD) (Apel and Kästner, 2009) is a paradigm for the construction, customization, and synthesis of large-scale software systems. In FOSD, features are used as the primary units to analyze, design, and implement software systems (Apel and Kästner, 2009). According to Kang et al. (1990), the feature is a prominent or distinctive user-visible, quality, or characteristic of a software system. The basic idea of FOSD is to decompose a software system in terms of the features it provides. FOSD is supported by four different phases: *domain analysis, domain design and specification, domain implementation, and product configuration and generation* (Kästner and Apel, 2011). The different features that are part of the software system will be determined by domain analysis with help of feature models. In the domain design and specification phase, structural and behavioral properties of each identified feature from domain analysis will be modeled through formal modeling languages. Based on the domain knowledge and design specifications a developer can manually code or automatically generate code for each identified feature in the domain implementation phase. In this phase, the concept of feature-oriented programming (FOP) (Prehofer, 1997) was widely used by the researchers to separate the feature-related code and base programs. In recent years, researchers have made significant advancements to FOP in the form of dynamic slicing of feature-oriented programs with execution trace files (Sahu and Mohapatra, 2017), concurrent feature-oriented programs (Sahu and Mohapatra, 2019), and feature-oriented programs with aspect-oriented extensions (Sahu and Mohapatra, 2020). Finally, an efficient product can be generated based on the feature selection by the users.

The proposed approach in this thesis follows the FOSD paradigm. In this research work, the energy-aware self-adaptive software is decomposed into different features. The features are selected in such a way that it consumes a different amount of energy under different operating conditions. The final product(s) derived from the features would consume different energy for its operation. Through domain analysis, the developers would be able to understand the different energy-hungry features and their alternatives. In addition, the developers might have a clear idea of different operating conditions. The different product configurations will be mapped to different operating conditions at run-time for better energy-savings. The domain design and specification is supported by the domain-specific modeling language tailored for energy-aware software. The DSML proposed in this research work can model and specify different features and operating conditions. The domain implementation of the FOSD is supported by the automatic code generation of MDD where the code will be generated from the textual domain-specific language DSML. Finally, the final generated code would be used to construct energy-aware software to complete the FOSD paradigm.

2.6 Summary

This chapter introduces the background knowledge relevant to the methodologies adopted in the design and development of energy-aware self-adaptive software. In summary, as reported in the literature, the existing research work has proved that changing the application behavior at runtime based on battery-awareness would aid in developing energy-efficient mobile apps. The energy-related context information such as battery level, battery charging state, etc will be used as a primary source of context in this research work. Therefore, using context-awareness and self-adaptivity to the application will ensure that the energy-efficiency requirements are met. From the development perspective, this research work uses the FOSD and MDD paradigm for the development of energy-aware self-adaptive software. The FOSD paradigm is used in this research work as the development paradigm and the feature models are used for domain analysis. For domain design and domain implementation, Model-driven Development is used. Further, the modeling aspects of MDD are covered through features, context, and adaptation modeling.

Chapter 3

LITERATURE SURVEY

This chapter discusses and compares the concepts proposed in this thesis to the research works published in the literature. The literature survey is presented under three different categories. The first category covered the literature related to empirical studies about mobile apps' energy consumption and summarized in Section 3.1. Notably, it covers the literature that considers data from Stack Overflow and GitHub to analyze data related to mobile apps' energy consumption. The second category includes the research efforts that can be considered after the development phase to reduce Android applications' energy consumption and is summarized in Section 3.2. Specifically, it covers the tool support available for energy profiling, diagnosis, and automated repair of energy-inefficiencies or energy bugs in Android apps. The third category considers the research efforts applied in the early stages of software development and outlined in Section 3.3. Mainly, it covers the research work related to the following topics: (1) modeling frameworks for self-adaptive software; (2) feature modeling approaches for specifying non-functional requirements; (3) model-driven development of mobile apps. To the best of our knowledge, none of the research works presented in this category have attempted to propose a method for modeling smartphone applications' energy-efficiency requirements at design time. However, the discussed research works motivate us to provide a method for considering energy-efficiency and self-adaptivity in the early phases of smartphone app development.

3.1 Existing Empirical Studies

This subsection summarizes empirical studies related to the Android application's energy consumption, considering data available on Stack Overflow and GitHub.

3.1.1 Stack Overflow data

In recent years, there has been an increase in the number of empirical studies on developer discussions. In general, these studies are conducted using online surveys (Pang et al., 2016), interviews (Manotas et al., 2016), and mining questions and answers on online discussion forums (Malik et al., 2015, Pathak et al., 2011, Pinto et al., 2014). Off late, online repositories such as Stack Overflow have proven to be a reliable source to summarize a developer's knowledge (Ahmed and Bagherzadeh, 2018, Rahman et al., 2018). In general, mobile development teams are relatively small, and consist of developers with less background knowledge (Manotas et al., 2016, Pang et al., 2016) on best practices of using location APIs. Therefore, the new developers post their issues on developer discussion forums like Stack Overflow to get suitable and working solutions (Calefato et al., 2018). Stack Overflow is one of the larger Q&A discussion forums for developers with over 10 million users and 18 million posted questions. As Stack Overflow is the largest developer discussion forum, it has become the ideal data source for mining developer discussions. Stack Overflow has been used as a data source by researchers to summarize the developer discussions in the recent past (Ahmed and Bagherzadeh, 2018, Tahir et al., 2018). The Stack Overflow data has been analysed using card sorting (Zimmermann, 2016), thematic analysis (Braun and Clarke, 2013), and topic modeling methods (Blei et al., 2003, Deerwester et al., 1990, Lee and Seung, 1999). This shows the importance of considering Stack Overflow to organize the existing developers' knowledge of new application platforms or programming languages. In recent years, several empirical studies were (Malik et al., 2015, Manotas et al., 2016, Pathak et al., 2011) conducted on Stack Overflow to summarize the knowledge on energy consumption of Android applications. This subsection summarizes current research work related to mining Stack Overflow data to analyze developers' experience and challenges related to the energy consumption of software.

The first empirical study on categorizing the energy related issues of smartphones was published by Pathak et al. (2011) in 2011. The authors have mined four online forums to categorize energy-related issues of smartphones. The authors presented a comprehensive taxonomy ranging from problems related to battery, SIM card, OS configurations to no-sleep bugs covering hardware, software, and external conditions.

Pinto et al. (2014) mined StackOverflow for software energy consumption-related questions and answers. The authors have identified seven major causes for energy consumption problems, varying from unnecessary resource usage, background activities, and synchronization. In addition, the authors have discussed several possible solutions to address the energy related issues.

Malik et al. (2015) explored the quantitative and qualitative aspects of energy-related questions specific to the Android platform on StackOverflow. The authors have summarized energy-related issues into four main categories and explores the APIs that are significantly discussed in the energy-related posts. However this study does not concentrate on energy and accuracy related issues of location-based Android applications.

Although there have been several empirical studies on the StackOverflow data, none of the studies have concentrated on the issues related to location-based Android applications. Hence, in contrast to these research efforts, this thesis attempt to mine the developers' discussions about location-based Android applications.

3.1.2 GitHub data

The data available from open-source Android repositories hosted on GitHub would help us understand the developers' perceptions from posted issues. The researchers have considered GitHub as the majority of the open-source Android apps from F-Droid [1] is hosted in GitHub. Also, GitHub allows non-members to browse through the source code of the whole repository and post an issue. Therefore, the researchers have collected the required data from the GitHub platform to answer the research questions. In literature, two related studies has been found (Moura et al., 2015) and (Bao et al., 2016) that

[1]https://f-droid.org/

use GitHub data for answering their research questions. Both these studies focused on finding a solution to energy consumption in various software fields.

Moura et al. (2015) focused on mining energy-aware commits of 317 non-trivial real-world applications. An initial dataset of 2,189 commits containing 371 energy-aware commits was used to perform a thorough qualitative study. The researchers have found answers to certain questions regarding the solutions that developers use to save energy. They found that frequency scaling and exploring multiple levels of idleness were being used. Another important finding was the software quality attributes which would be given precedence over energy consumption. However, their studies do not focus on Android applications in specific.

On the other hand, Bao et al. (2016) has performed an empirical study on power management commits of 154 different Android applications obtained from F-Droid and crawling 468 commits on GitHub. They have filtered the commits using a set of keywords and conducting a manual analysis on it. Further, they used open card sorting to categorize the commits into six groups. The developers have chosen to answer questions on how the developers manage power consumption in Android applications and the type of applications that are more concerned about power management. Their analysis said that there are six main power management activities, namely: Power Adaptation, Power Usage Monitoring, Power Consumption, Optimizing Wake Lock, Adding Bug Fix, and Code Refinement. They have also found that Connectivity, Phone, and SMS category applications are much concerned with power management commits.

In contrast to these studies, this thesis focus mainly on location-based Android applications' energy-related commits to summarize the developer's knowledge and help the upcoming developers on energy and accuracy-related matters.

3.2 Post-development Approaches

The existing research works majorly contribute to tool support to reduce battery consumption of Android applications. Typically these tools are used after the development phase to check for energy-inefficiencies at run-time. This subsection discusses the research works related to finding and fixing energy-inefficiencies in Android applications.

3.2.1 Energy profilers

Energy profilers are the tools that help in providing the component wise energy consumption of mobile devices. Initially, external hardware-based profilers were used to estimate the energy consumption of mobile devices. The popular hardware-based profiler are Monsoon Power Monitor (Monsoon Solutions, 2018), BattOr (Schulman et al., 2011), and NEAT (Brouwers et al., 2014). Here, an external power meter will be connected to the mobile device battery to measure the overall energy drawn by the device. The accuracy is good and gives better measurement results in all situations. The disadvantage of this approach is, it gives only the device energy consumption. It does not give sub-components wise (CPU, 3G, GPS, etc.) or App wise energy consumption values. On the other hand, optimizing smartphone applications for energy efficiency requires fine granular level energy estimation of application components. Therefore, software-based profilers were introduced to provide fine granular level energy estimation (Ahmad et al., 2015). The software-based energy profilers can be broadly classified into two categories: On-device profilers and off-device profilers (Hoque et al., 2016). On-device profilers will be installed on the mobile device, and Off-device profilers will be installed on the computer for estimation. The power model construction plays a major role in deciding the accuracy of the profiler. Similar to deployment, the power models can be constructed either on-device or off-device. After constructing a power model, the profilers will be deployed on the mobile or computers. As summarized in Hoque et al. (2016), there are three different type of approaches for energy profilers: (1) On-device deployment with on-device model construction (Dong and Zhong, 2011, Jung et al., 2012, Yoon et al., 2012, Zhang et al., 2010), (2) on-device deployment with off-device model construction (Kjærgaard and Blunck, 2011), and (3) off-device deployment with off-device model construction (Pathak et al., 2012a). In this sub-section, a few notable research works on software-based energy profilers are discussed.

PowerBooter (Zhang et al., 2010), is an automatic power model generation technique for mobile phone. This approach uses the built-in battery voltage sensors and battery discharge behavior to generate the device-specific power models as the single device power model is not suitable for other devices. Further, the authors described a tool PowerTutor[2], which uses PowerBooter to estimate the energy consumption of Android

[2]http://ziyang.eecs.umich.edu/projects/powertutor/

platform smartphones. PowerTutor is an Android application that runs on Android devices to estimate the device's hardware component level energy consumption. PowerTutor provides high accuracy only for HTC G1, HTC G2, and Nexus one device as it was built on top of these devices' power models.

Sesame (Dong and Zhong, 2011), a self-modeling approach, produces mobile systems' energy models using self-modeling. Here, the mobile systems' smart battery interface is used to generate the power models to avoid external metering or interventions to construct the power models. Sesame was developed for laptops and mobiles running Linux based kernels. The system was evaluated using Lenovo ThinkPad T61 laptop and Nokia N900 smartphone. The authors showed 95% accuracy for laptops and 86% accuracy for smartphones.

DevScope (Jung et al., 2012) generates the accurate power model of Android hardware components automatically. The authors solved the existing problem with Battery Management Unit (BMU) and produced dynamic power models. DevScope analyzes the power characteristics of CPU, display, WiFi, cellular, and GPS. The power models generated by DevScope was used by AppScope (Yoon et al., 2012) to estimate the energy consumption of smartphone hardware components. Yoon et al. (2012) proposed an Android-based energy metering framework (AppScope). This approach estimates the energy consumption of device components by monitoring the hardware usage at the kernel level. The authors claim that their approach provides fine granular energy estimation with less overhead. Appscope is implemented as a dynamic module in the Linux kernel 2.6.35.7 and evaluated on the Google Nexus One device running Android platform version 2.3.

eLens (Hao et al., 2013) analyzes the Android applications at the source code level and estimates it's run-time energy consumption. This approach uses the concepts of static program analysis and per-instruction energy modeling. eLens doesn't require external additional hardware or modification at the operating system to estimate energy consumption. Being a lightweight approach, it works at the source code level and utilizes per-instruction energy models to provide fine-grained energy consumption estimation. eLens takes the input three inputs: (1) software artifact, (2) app's run-time workload, and (3) per-instruction energy models. The source code annotator component of eLens creates the annotated version of the source code that may help developers understand the source code's energy consumption behavior. This tool provides energy consumption at different granularity: the whole program, path, method, and source line. This tool was empirically evaluated using the Java byte of six Android applications from the Google Play Store. The results show that this approach is lightweight and accurate for the Android application's energy estimation at the source code level.

The previously described approaches are suitable for smartphones that use the earlier version of the Android platform. Whereas, the recent generation of mobile devices does not use those Android platform versions. Moreover, recent generation smartphones' hardware capabilities have drastically changed, which makes the power models generated in the previous works unsuitable. A recent profiling tool developed by Qualcomm developers called Trepn Profiler[3] is suitable for most of the recent generation Android smartphones. This tool is an Android application that runs on the device and monitors other applications' usage statistics running on the same device. This tool provides better accuracy only on Snapdragon-powered smartphones. Though the tool is effective, the additional overhead of running this tool on the smartphone remains a limitation of this tool.

Google Battery Historian[4] is the popular tool developed by Google developers for fine-grained energy consumption estimation. This tool uses the bug report produced by Batterystats[5]. This tool allows the developers to input their device bug report and visualize the battery related information on a timeline. The web interface provided by this tool provides the app wise, component-wise energy estimation values. Comparing two different bug reports allows the developer to visualize the key issues in battery related behaviors of two set of application events. The aggregated battery statistics provided by this tool may give key insights to developers to optimize the application for energy efficiency. This tool can only estimate energy consumption, and this tool cannot pinpoint the Android application's energy inefficiencies.

The other important profiling tool for profiling applications energy usage is Android studio energy profiler[6]. This tool monitors CPU, network, and GPS's energy usage, along with system events like wake lock, alarms, and job scheduler. This tool's visual

[3]https://developer.qualcomm.com/software/trepn-power-profiler
[4]https://github.com/google/battery-historian
[5]https://developer.android.com/studio/profile/battery-historian
[6]https://developer.android.com/studio/profile/energy-profiler

timelines help the developers locate the system events responsible for abnormal energy consumption. This tool also fails to automatically pinpoint the source code segment responsible for abnormal energy consumption. In addition to the support for developers, the Android operating system supports smartphone users to inspect the energy usage of their phone under settings. Users can see the energy usage of apps and take necessary actions to reduce abnormal energy consumption. The actions like terminating the background activities, background location sensing, and adjusting display settings are a few of the most common methods for reducing energy consumption. Like other energy profiling tools, this tool is also restricted only to work on the deployed application. This tool cannot pinpoint the energy inefficiencies in the developer written code.

The existing energy profilers are capable of providing sub-component-wise and app-wise energy consumption values. These tools estimated power consumption through a well-defined power model, which considers various mobile device sub-components. There are different power models available: utility-based models, event-based models, code analysis based models, etc. These approaches can be well utilized for optimizing the applications as it gives the energy consumption value per component. This approach's major drawback is that the energy profilers failed to automatically identify the particular code segment responsible for abnormal energy consumption. The tools need a developer to manually identify energy-hungry elements in the code segment by looking at the energy profiler report. Therefore, the research community started focusing on developing automated energy diagnosis tools to identify the Apps' energy-hungry code segments. Besides profiling, these tools give an idea about the energy-hungry components or code segments of the application. These energy diagnosis engines are handy tools during the development, helping the developers identify energy bugs present in the application before deployment.

3.2.2 Energy diagnosis engines

To overcome the limitations of energy profilers, researchers have introduced *energy diagnosis tools* to automate the process of locating energy inefficiencies in Android applications. In literature, there are significant efforts made towards automatically detecting the energy bugs of Android applications. There are several user-centric approaches

(Kim and Cha, 2013, Oliner et al., 2013, Wang et al., 2014) to optimize energy inefficiency issues at run-time. These tools do not require the developer's involvement to fix energy inefficiencies. However, the solutions suggested by these approaches are temporary as they do not modify the source code. On the other hand, there are a few developer-centric approaches detecting sensor data under-utilization (Liu et al., 2014,0, Wang et al., 2016). These approaches help developers optimize their app by pinpointing energy bugs' locations at Android applications' source code level. Besides finding, few approaches fix the energy inefficiencies automatically at source code level. These approaches use popular software engineering methods such as program analysis, software bug-localization, refactoring, and automated software repair to address the identified energy-related issues. Automated energy diagnosis tools help developers analyze the source code or intermediate code to pinpoint energy inefficiencies with minimal manual efforts. Leveraging energy diagnosis tools in the development and testing phases may produce energy-friendly applications for battery-powered devices. This subsection summarizes significant automated energy diagnosis engine-related research efforts that use program analysis and refactoring techniques. These approaches can be broadly classified into two categories: finding only approaches and fixing approaches. The finding only approaches include GreenDroid (Liu et al., 2014), E-GreenDroid (Wang et al., 2016), μDroid (Jabbarvand and Malek, 2017), eDelta (Li et al., 2017a), TailEnergy Abbasi et al. (2018), and EnergyDebugger Banerjee et al. (2016). The fixing aapproaches include EnergyPatch (Banerjee et al., 2018), Gottschalk et al. (2012), Banerjee and Roychoudhury (2016), Leafactor (Cruz and Abreu, 2018), and EARMO (Morales et al., 2018).

GreenDroid (Liu et al., 2014) is another approach that utilizes dynamic information flow analysis (Kemerlis et al., 2012) to detect energy inefficiencies of Android applications. The authors conducted a preliminary empirical study to identify common causes of energy-related issues: "*(1) missing deactivation of sensors or wakelock, and (2) under-utilizing the sensory data*" (Liu et al., 2014). GreenDroid takes input as Java bytecode produced either from the source code or converting apps Dalvik bytecode (Octeau et al., 2012). Further, GreenDroid analyzes the registering and unregistering of sensors and wakelocks at each explored state to decide on energy-related issues. In this approach, the sensor and wakelock management policies are used to detect the problems by adopting the resource leak detection techniques (Arnold et al., 2011, Weimer and Necula, 2004). The dynamic tainting approach was used to detect the sensory data underutilization. GreenDroid contains the following three phases: "*(1) tainting sensory data, (2) propagating taint marks, and (3) analyzing the sensory data underutilization*" (Liu et al., 2014). The GreenDroid tool was implemented on top of JPF, and the approach is evaluated with 13 popular Android applications.

E-GreenDroid (Wang et al., 2016) focused on addressing all the known limitations of GreenDroid Liu et al. (2014). The effectiveness of E-GreenDroid was validated with the same apps used to validate the original implementation of GreenDroid (Liu et al., 2014). The experimental results show that the E-GreenDroid effectively provides support to Android 5.0 while retaining the same effectiveness of GreenDroid. Another extension to the GreenDroid approach is CyanDroid (Li et al., 2017b), which aims at using white box sampling technique (Bao et al., 2012) for handling multi-dimensional data. The authors have highlighted two limitations of GreenDroid approach: (1) random generation of sensory data, and (2) not considering different execution paths during analysis. In CyanDroid (Li et al., 2017b), systematic generation of multi-dimensional sensor data was investigated along with fine-granular state-space concept. A case study was conducted on four popular Android apps to validate CyanDroid.

µDroid (Jabbarvand and Malek, 2017) is a mutation testing framework to reveal energy-related defects in Android applications. This framework seeds artificial defects in the program known as mutation operators to simulate commonly occurring energy defects. The energy consumed by the mutated version and the original subject application are compared to determine the defects. The framework consists of three components, a) Eclipse Plugin b) Runner/Profiler c) Analysis Engine. The authors have collected 28 types of energy anti-patterns to create the mutation. The algorithm runs each test multiple times on both the original and modified copies of the application and determines the corresponding power traces. The similarity between the power traces are compared by calculating the "*Dynamic Time Warping (DTW) distance*" (Jabbarvand and Malek, 2017). The authors evaluated this framework on nine apps and used it to instrument different mutation operators in subject applications. The framework was able to identify several unexplored energy bugs and showed an overall accuracy of 94% in correctly identifying these energy bugs.

eDelta (Li et al., 2017a) is an automated trace-based detection framework to address abnormal battery drain issues. It helps developers to pinpoint the energy-hungry APIs with the help of comparative trace analysis. The key idea in this approach is to analyze the energy consumption deviation in different user traces. eDelta works in two phases: "*1) online tracking on smartphones, and 2) offline diagnosis*" Li et al. (2017a). eDelta was evaluated using twenty apps running on Android version 4.4. The online tracking facility was installed on the user devices to collect the necessary information for analysis. The usage of data and power traces were collected from multiple users under different operating conditions and contexts. This tool suffers from the additional overhead of app instrumentation and power consumption by the online tracking component. The results show that eDelta reduces 94.6% of the code that the developer may search to fix the energy defects. Therefore, the results of eDelta might help developers reduce the time spent on analyzing and finding the root cause of Abnormal Battery Drain (ABD).

TailEnergy (Abbasi et al., 2018) introduce "*Application Tail Energy Bugs (ATEBs)*", which occur when an Android app consumes abnormal power than expected while running in the background. The authors have developed a desktop application to test the presence of ATEBs and conducted several experiments on five Android applications using the "*android debug bridge (adb)*"[7] command. This approach detects the ATEBs by comparing the system states before running and after terminating the subject applications. In this approach, the following five actions are considered to close the app: "*pressing the home button, pressing the back button, using the swipe-out gesture, using the force-stop option from the settings, and using the exit button if available*" (Abbasi et al., 2018). The log files are generated and saved in the server by the "*daemon (adbd)* on the smartphone under test. These commands are executed twice, once before starting the subject applications and once after terminating the applications. Later, the presence of ATEBs is detected by the text parser after comparing the contents of two log files generated by the *daemon (adbd)*.

EnergyDebugger (Banerjee et al., 2016) analyzed the run-time log messages and systems states to detect energy-related field failures. The authors have provided an instrumentation method to address energy-related issues where the data logging code will be added to the user application. The instrumented code is equipped with logging utility to

[7]https://developer.android.com/studio/command-line/adb

record the execution logs at run-time. Later, this logging utility automatically uploads the log files to the developer side tool for defect localization. This approach has the four step process to address the energy-related issues. In the first phase, the user app is instrumented with logging utility, which captures the event-handlers, API calls, status of wakelock, and GPS updates. In the second phase, log-messages shared by the user side tool were converted to an energy profile call graph. In the third phase, the contextual subgraph is extracted from the energy profile graph. In the fourth phase, the location or origin of the defect is identified in the source code. In addition, this approach provides patch suggestions to help developers improve energy efficiency. The developer tool of this approach was developed as an Eclipse plugin[8] and evaluated with two Android applications running on Samsung S4 smartphone running with Android version 4.4.2. The authors have shown 5% to 29% energy saving as the result of patching energy defects.

As summarized in Table 3.1, the finding approaches primarily consider the energy inefficiencies of resources such as wakelock, sensors, UI, and connectivity. The dynamic analysis approaches such as GreenDroid (Liu et al., 2014), E-GreenDroid (Wang et al., 2016), and CyanDroid (Li et al., 2017b) were used to identify the sensor-related issues such as *missing sensor deactivation and sensor data under utilization*. The network-related issues are identified in TailEnergy (Abbasi et al., 2018). In contrast to the previous approaches, EnergyDebugger (Banerjee et al., 2016), and MuDroid (Jabbarvand and Malek, 2017) consider energy issues related to multiple smartphone resources. This includes resource leaks, vacuous background services, immortality bug, loop energy bug, anti-patterns in code related to location, connectivity, callbacks, and sensor. Overall, researchers have adopted dynamic analysis approaches that include dynamic program analysis and comparison of trace files by executing real devices' applications. Therefore, the popular *Dynamic Information Flow Tracking (DFT)* (Suh et al., 2004) has been used in the approaches GreenDroid (Liu et al., 2014), E-GreenDroid (Wang et al., 2016), and CyanDroid (Li et al., 2017b) to track the tainted object dynamically.

[8]https://www.comp.nus.edu.sg/ rpembed/energydebugger/index.html

TABLE 3.1: Summary of developer-centric finding approaches

Approach	Identified energy-inefficiencies	Deployment	Tool input	Finding Method	Representation for analysis
Green Droid (Liu et al., 2014)	Missing deactivation, Sensor data under utilization	External Application	Java Byte Code and Configuration files	Dynamic information tracking	Applications's state space
E-Green Droid (Wang et al., 2016)	Sensor listener and wake lock misusage, Sensory data underutilization	External Application	Java Byte Code and Configuration files	Dynamic information tracking	Applications's state space
Cyan Droid (Li et al., 2017b)	Sensory data underutilization	External Application	Java Byte Code and Configuration files	Dynamic information tracking	Applications's state space
Energy Debugger (Banerjee et al., 2016)	Resource leaks, suboptimal resource binding, wakelock bug, tail-energy hotspot, vacuous services, expensive services, immortality bug, loop-energy hotspot	Modifications to App Executable	apk	Context-sensitive pattern based fault localization	Profile call graph
eDelta (Li et al., 2017a)	abnormal battery drain	Modification to App Executable	apk	Trace segmentation and statistical analysis	Action traces and power traces
MuDroid (Jabbarvand and Malek, 2017)	anti-patterns in code	Modification to Source Code	Source Code	Mutation analysis (Comparing power traces original and mutated version of the apps)	Power traces
TailEnergy (Abbasi et al., 2018)	ATEB (Application Tail Energy Bugs)	External Application	apk	Comparing pre-state and post-state file	Pre-state and post state file

Besides these approaches, few trace analysis approaches compare the real user traces of subject applications to determine energy inefficiencies. The *trace analysis* approaches uses *fault localization* (Banerjee et al., 2016), *trace segmentation* (Li et al., 2017a), mutation analysis (Jabbarvand and Malek, 2017), and *comparing power states* (Abbasi et al., 2018) to detect the energy inefficiencies in Android applications. These tools use the *configuration files* in addition to *Java Byte Code*. Further, the trace analysis approaches such as EnergyDebugger (Banerjee et al., 2016), eDelta (Li et al., 2017a), TailEnergy (Abbasi et al., 2018), (Jabbarvand and Malek, 2017) use the app executables (apk) or source code as the input. The inputs given to the analysis tools are converted into different representations for analyzing the subject applications for energy inefficiency. The source input of subject applications were further converted to a graph or trace file suitable for the analysis.

Overall, the previously discussed approaches can automatically find out the energy-inefficiencies in the apk source code of the Android applications. However, these tools cannot automatically fix the identified energy-inefficiencies, and it requires manual interventions of the developer. In contrast to the previously discussed approaches, there are several fixing approaches found in the literature. These approaches can automatically fix the identified energy inefficiencies besides finding them. The following approaches fix the energy inefficiencies automatically at source code level: EnergyPatch (Banerjee et al., 2018), Gottschalk et al. (2012), Banerjee and Roychoudhury (2016), Leafactor (Cruz and Abreu, 2018), and EARMO (Morales et al., 2018) . These approaches use popular software engineering methods such as *program analysis, software bug-localization, refactoring, and automated software repair* to address the identified energy-related issues. The remaining part of this subsection summarizes the approaches that automatically fix the energy inefficiencies.

Gottschalk et al. (2014,0) have presented the first research effort on considering energy inefficiencies as code smells and the importance of applying refactoring techniques. The authors have presented an approach to detect and restructure the energy-inefficient patterns in Android applications. The refactoring process begins by forming an abstract representation of the app's Java code and storing it into a central repository. The refactored source was compiled and deployed on the test device to show the efficacy of this approach. The authors have considered five code smells: "*a) data transfer, b) backlight, c) statement change, d) third-party advertising and e) binding resources too early*" (Gottschalk et al., 2014,0). The validation was performed on the applications GpsPrint and TreeGenerator using two smartphones Samsung Galaxy S4 and HTC One X. The energy-saving measurement obtained from delta-B measurements was 14%, from file-based measurements, it was 8.6%, and from energy, profile measurements were about 8.3%. The results show that it is feasible to adopt refactoring techniques for improving the energy-efficiency of Android applications.

Banerjee and Roychoudhury (2016) have presented a light-weight refactoring method to assist the energy-aware app development. This approach detects and refactors abnormal energy-draining behavior with the help of energy-efficiency guidelines. This approach relies on the following three components: "*(a) design extraction component (b) refactoring component and (c) code generation component*" (Banerjee and Roychoudhury, 2016). The "*design extraction component*" is responsible for generating the design-expression of the subject applications. The objective of the *refactoring component* is to refactor the design-expressions by evaluating them for any guideline violations. Further, the "*code generation component*" connects the changes introduced in the refactored design expression to the source code of the subject applications. The exact location in the app source code is identified, and modifications are applied as suggested by the refactored expressions. This approach's effectiveness was evaluated by identifying the existence of energy-hungry design patterns in ten open-source subject applications. This evaluation was performed on Samsung S4 supported by a case study on the application *Sensorium*. Experimental results show a 3% to 29% reduction in energy consumption of subjection application after applying the refactoring.

Cruz and Abreu (2018) presented an automated refactoring approach to improve the energy efficiency of Android applications. The authors have used a tool named Leafactor (Cruz et al., 2017) to analyze and refactor applications to make them energy efficient. The static code analysis was used to detect the Android-specific anti-patterns automatically. The anti-patterns related to the following elements were considered in this approach: *ViewHolder, DrawAllocation, Wakelock, Recycle, and ObsoleteLayoutParam*. The proposed automated refactoring approach was implemented as an extension to the Eclipse-based AutoRefactor Plugin. The AutoRefactor supports only Java refactoring, and XML refactoring was done using a separate refactoring engine called Leafactor. The authors have evaluated the proposed approach on 45 open-source Android applications and identified 222 anti-patterns using Leafactor. The experimental results show that the *ObsoleteLayoutParam* and *Recycle* are most frequently occurring energy-related anti-patterns in the subject apps.

EARMO (Morales et al., 2018) is an anti-pattern detection and correction approach for addressing improper energy consumption issues. This framework refactors anti-patterns if it affects the energy-efficiency of the system. It covers the following object-oriented anti-patterns: "*blob, lazy class, long-parameter list, refused bequest, and speculative generality*" (Morales et al., 2018). In addition, this approach addresses the following Android anti-patterns: "*binding resources too early, hashmap usage, and private getters and setter*" (Morales et al., 2018). EARMO consists of 4 steps: a) Energy consumption estimation, b) Code meta-model generation, c) Code meta-model assessment, and d) Refactoring. The measurement given by "*TiePie Handyscope HS5*"[9] was used to evaluate the results produced by EARMO. It adopts search-based algorithms with multiple objectives to select the optimal refactoring sequence from the available list of refactoring opportunities. This approach was evaluated with 20 open-source Android subject applications on LG Nexus 4 device. From the experimental results, it can be observe that this approach produces recommendations for refactoring in less time and removes 84% of code with energy-related issues.

EnergyPatch (Banerjee et al., 2018) is an *automatic software repair* **?** based approach to automatically fix the energy inefficiencies by adopting the suitable combination of static and dynamic analysis methods. EnergyPatch has three important phases: "*detection, validation, and repair*" (Banerjee et al., 2018). The *detection* phase identifies the program paths that may contain potential energy bugs using a static analysis approach. The *validation* phase explores the buggy program paths to ensure the presence of energy inefficiencies. Finally, the third phase *repair* identifies the location of validated energy bugs to fix them using the automatically generated repair expressions. The developed tool handles the energy bugs caused by *resource leaks, wakelock misuse, and vacuous background services*. This framework is implemented as a plugin to Eclipse, and freely available[10] for usage. The authors have evaluated this framework on 35 subject apps

[9] https://www.tiepie.com/en/usb-oscilloscope/handyscope-hs5
[10] https://www.comp.nus.edu.sg/~rpembed/epatch/home.html

TABLE 3.2: Summary of developer-centric fixing approaches

Approach	Identified energy inefficiencies	Input to analysis	Finding Method	Fixing Method
		Program Repair Approach		
EnergyPatch Banerjee et al. (2018)	Resource leaks, Wake-lock bugs, Vacuous Background Services, Immortality Bug and inappropriate usage of system call APIs	Event Flow Graphs (EFG)	Combination of static and dynamic program analysis	Inserting Repair Expressions
		Refactoring Approaches		
Gottschalk et al. Gottschalk et al. (2012)	Loop bugs, inline code, dead code, immortality bugs	TGraphs produced from source code	Static analysis and graph transformation	Refactoring
Banerjee et al. Banerjee and Roychoudhury (2016)	Inefficiency of app due to ineffective design patterns	Succinct representation (Design expressions)	Checking for violation of energy-efficiency guidelines	Guideline based refactoring of defect expressions
Leafactor Cruz and Abreu (2018)	Android specific energy smells	Android project	Static Program Analysis	Automatic code refactoring
EARMO Morales et al. (2018)	Object-oriented anti-patterns, Android anti-patterns	Code meta-model	Code meta-model assessment	Evolutionary multi-objective technique
		Instrumentation Approach		
Li et al. Li et al. (2015)	Sensor data underutilization	smali code	Static analysis	Insert adaptive sensing policies to smali code

running on LG Optimus E400 smartphone. Further, the buggy applications were repaired using the automated repair expressions, and the experimental results show the improvement in energy efficiencies.

As shown in Table 3.2, the developer-centric fixing approaches are summarized based on the following parameters: *identified energy inefficiencies, input to analysis, finding method, and fixing method*. The approaches under this category can be further classified into the following themes based on the fixing scheme: *instrumentation and refactoring* approaches. The approach presented in Li et al. Li et al. (2015) instruments existing Android apps with adaptive sensing policies to fix the sensor-related energy inefficiencies. There are few classical refactoring approaches also found in Gottschalk et al. Gottschalk et al. (2012), Leafactor Cruz and Abreu (2018) to refactor the code smells that causes energy issues in Android applications. In addition, evolutionary multi-objective techniques were used in EARMO Morales et al. (2018) to correct the anti-patterns. The EnergyPatch Banerjee et al. (2018) approach uses the automated program repair to fix the energy inefficiency issues. These approaches primarily rely on the *finding methods*, such as program analysis and code meta-model assessment. The *program analysis* techniques are used in Li et al. Li et al. (2015) and EnergyPatch Banerjee et al. (2018) to find the energy inefficiencies. Particularly, EnergyPatch Banerjee et al. (2018) uses the combination of static and dynamic program analysis to detect and validate energy inefficiencies. In Li et al. Li et al. (2015), only static program analysis method is used to find the energy inefficiencies. In addition to program analysis techniques, the following finding techniques has been found in other candidate studies: GREQL is used in Gottschalk et al. Gottschalk et al. (2012); code meta-model assessment is used in EARMO Morales et al. (2018); and checking for a violation of energy-efficiency guidelines is used in Banerjee et al. Banerjee and Roychoudhury (2016) to find out the energy inefficiencies.

To summarize the input to analysis, the approaches use *intermediate representations and Android project files* as an input to the analysis tool. For instance, Gottschalk et al. Gottschalk et al. (2012), Li et al. Li et al. (2015), Banerjee et al. Banerjee and Roychoudhury (2016), EARMO Morales et al. (2018), and EnergyPatch Banerjee et al. (2018) use the intermediate representations of Android applications. On the other hand, whole android project files are used by Leafactor Cruz and Abreu (2018) for analysis. With respect to *identified energy inefficiencies*, Li et al. Li et al. (2015) only deals with the sensor data underutilization problem. The EnergyPatch covers many inefficiencies, which includes "*resource leaks, wakelock bugs, vacuous background services, immortality bug and inappropriate usage of APIs*" Banerjee et al. (2018). In addition to resource-specific issues, the refactoring approaches introduced a new class of energy bugs by investigating the energy impact of object-oriented smells and Android-specific smells. For instance, EARMO deals with object-oriented anti-patterns such as "*blob, lazy class, long-parameter list, refused request, speculative generality*" Morales et al. (2018). In addition, the authors have considered Android-specific code smells such as "*binding resources too early, hashmap usage, private getter and setters*" Morales et al. (2018). In LeaFactor Cruz and Abreu (2018), the authors have introduced a new class of energy in-efficiencies, namely *energy code smells* by investigating the Java and Layout files of Android applications.

3.3 Pre-development Approaches

The research work proposed by Kelényi et al. (2014) supports the development of an energy-efficient mobile application with model-driven code generation. Mainly, the authors aim to assist the developer through a model and code library-based approach. In addition, the importance of considering energy efficiency at the modeling level rather than directly optimizing at the code level was pointed out clearly. The results presented in this motivates us to carry out this research work to propose suitable solutions through modeling. Our approach is different from this approach in terms of using context-awareness and self-adaptivity for energy savings.

3.3.1 Feature modeling approaches

In the literature, graphical feature model notations provided by FODA framework Kang et al. (1990) were widely used in the subsequent research efforts, such as FeatuRSEB Griss et al. (1998), FORM Kang et al. (1998), and generative programming Czarnecki and Eisenecker (2000). These research efforts have added extra modeling notations to the original FODA notations. These studies have the aim of analyzing commonality and variability in a software system. Several tools such as FDL Van Deursen and Klint (2002), FeaturePlugin Antkiewicz and Czarnecki (2004), FAMA Benavides et al. (2007), CVM Reiser (2009), TVL Classen et al. (2011), FAMILIAR Acher et al. (2013), and FeatureIDE Thüm et al. (2014) have focused on facilitating modeling of feature models. Previous secondary studies such as Schobbens et al. Schobbens et al. (2006), Sinnema et al. Sinnema and Deelstra (2007), Benavides et al. Benavides et al. (2010), and Noorian et al. Noorian et al. (2012) compared several extensions and tool support to basic feature models. Recently, several studies Soltani et al. (2012) Siegmund et al. (2012) Asadi et al. (2014) Mauro et al. (2018) have focused on leveraging the ability of feature models to analyze non-functional requirements and self-adaptive or context-aware requirements Desmet et al. (2007) Inverardi and Mori (2010) Mauro et al. (2018). This section systematically compares these approaches. Specifically, the support for energy-related requirements and self-adaptation planning has been investigated. First, the related works have been summarized, and then the approaches have been compared based on the following criteria: (1) configuration generation time, (2) type of representation, (3) support for energy-awareness, (4) support for context representation, (5) support for self-adaptation planning, and (6) tool support.

Context-Oriented Domain Analysis (CODA) (Desmet et al., 2007) approach is a systematic method for gathering and analyzing the requirements of context-aware systems. This approach presents the notations for specifying *variation point, context condition, and adaptation* along with the notations suggested in FODA (Kang et al., 1990). This approach depends on the graphical models and decision tables to specify corresponding adaptive actions for each allowed context changes. Here, the context information is modeled as an attribute to the feature diagrams. This approach does not provide explicit support for specifying energy-related requirements. Instead, it provides support to specify the energy-related context in the feature model through context conditions.

Inverardi and Mori (2010) proposed a feature-oriented approach to cover the *requirements* and *design* phases of developing dynamically evolving systems to plan run-time adaptations. This approach suggests to define the *features, system variants, and context* using feature models. Here, feature models are used to specify the functional requirements and to derive system configurations. A separate model is used to specify the context and to derive context-changes. A table-based method was used to map the context changes to its corresponding system variant to execute at runtime. This method does not have explicit support for energy-related requirements.

Soltani et al. (2012) proposed a framework that uses *annotations* to specify the non-functional properties. This approach computes *feature ranks* based on the impact of non-functional properties on the stakeholder's preferences. The calculated *feature ranks* are essentially used for optimizing the generated feature model configurations. This approach provides partial support for specifying energy-related requirements as it captures the non-functional properties. However, this approach does not provide explicit support for energy requirements, context modeling, and self-adaptation planning.

SPL Conqueror (Siegmund et al., 2012) attempts to optimize the non-functional properties during the product derivation phase of software product line engineering. The authors have introduced the importance of satisfying the user-specific quality requirements of product variants. The non-functional properties such as *reliability, code complexity, footprint, and performance* are measured and optimized in this approach. However, this approach does not provide explicit support for energy-related requirements while deriving the product variants. Besides, this approach has no provision for modeling the context information and planning run-time adaptation.

The approach presented by Asadi et al. (2014) also aims at optimizing the non-functional requirements while generating the feature model configurations. This framework is driven by the positive and negative impact of features on the non-functional requirements of the system. Unlike other approaches, this approach considers the interdependencies between the non-functional requirements. This approach provides *annotation* based extension to feature models for specifying the non-functional requirements. However, this approach does not support energy-awareness, context modeling, and self-adaptation planning.

Pascual et al. (2015a) used feature models to model the variability of mobile applications. The multi-objective evolutionary algorithms are used to optimize the product configurations at run-time. This approach considers the non-functional requirements, such as *usability, battery consumption, and memory footprint*. The cross-tree constraints are used to represent the inter-dependencies of non-functional properties. Finally, the optimal configuration is selected for the current context execution using the multi-objective algorithms. This approach partially supports energy-awareness and context modeling as it used feature models to specify these details. Besides, this approach produces run-time re-configurations for self-adaptation.

Common Variability Language (CVL) (Pascual et al., 2015b) generates run-time configurations and re-configurations for the self-adaptation of mobile systems. This approach uses CVL for generating design-time configurations, and the run-time configurations are generated using a genetic algorithm. The language proposed in this approach aims at modeling the architectural level variability of the self-adaptive system. In this approach, the energy-related information and mobile-specific context information are added into the feature models for generating optimal configurations.

Mauro et al. (2018) presented a method to model context-aware feature models for evolving software product lines. The authors have highlighted that all configurations given by the basic feature model would not be sufficient to deal with dynamic changes in the context and user preferences. To address these issues, the authors have proposed a framework and two tools, namely "*DARWINSPL*" and "*HyVarRec*". This approach suggests adding the context information in the feature diagram itself along with the features. This approach provides explicit support for context modeling and self-adaptation. However, this approach failed to consider the energy-awareness while defining context information or while generating re-configurations. '

CIM-CSS (Baddour et al., 2019) focuses on intelligent context-sensitive systems to formally model its context identification and management. This approach primarily focuses on the dynamically changing context of smart spaces and the Internet of Things (IoT). The authors have employed a goal-driven, entity-centered context identification method to find out the influential context information that can adapt the system behaviour at run-time. This approach uses Unified Modeling Language (UML), Object Constraint Language (OCL), and First-order Language (FOL) to formalize the context-identification and management. The authors have developed tool support and evaluated it through a smart meeting room application. This approach provides explicit support for context modeling and self-adaptation. The support for energy-awareness is not considered in this approach.

(Duhoux et al., 2019) covers the context model and behavioural adaptations that may be possible at run-time. This approach follows feature-based context-oriented programming and uses feature models for representing context and features separately. In addition, this approach covers the dependencies between the context model and feature model. The authors have developed a visualization tool that displays the activated or changed context information dynamically and the features affected by respective context changes. The developed tool has explicit support for self-adaptivity. The energy-awareness feature is not considered in this approach.

RETAkE (dos Santos et al., 2021) focuses on addressing the run-time defects and system failures caused by dynamic adaptation in self-adaptive systems. Specifically, this approach performs run-time testing of adaptive behavior, based on context variation and feature models to determine run-time system failure. Primarily, RETAkE approach provides test case generation and verification to verify adaptation rules at run-time. Additionally there is a tool support provided, that can be adapted to existing dynamically adaptive systems to control context and verify feature status. The authors have also provided support for modeling self-adaptation and energy awareness.

Comparing approaches

The related works has been compared based on the following criteria: (1) configuration generation time, (2) type of representation, (3) support for energy-awareness, (4) support for context representation, (5) support for self-adaptation planning, and (6) tool support. The results of the comparison of the approaches is presented in Table 3.3. With respect to *configurations generation time*, most of the approaches Mauro et al. (2018), Pascual et al. (2015b), Siegmund et al. (2012), Soltani et al. (2012) generate the configurations and re-configurations at design-time and run-time. However, employing a suitable mechanism at the requirements phase Inverardi and Mori (2010) would help the development team to plan a better energy-saving adaptation. With respect to *type of representation*, the majority of the approaches Asadi et al. (2014), Baddour et al. (2019), Desmet et al. (2007), dos Santos et al. (2021), Duhoux et al. (2019), Inverardi and Mori (2010), Soltani et al. (2012) used a graphical model over textual models. It has been observed that the graphical models are more suitable for the early stages of software development, while textual models are useful for later stages with model-to-model transformation support. With respect to *support for energy-awareness*, it can be observed from Table 3.3, few approaches provided partial support for considering energy-related requirements.

These approaches added energy-related information in the form of attributes Desmet et al. (2007) and annotations Soltani et al. (2012) for planning re-configurations. However, none of the existing approaches have considered energy as an important quality criterion. With respect to *context-representation*, energy-information have been added as an attribute Desmet et al. (2007) and annotation Soltani et al. (2012) in the existing feature model. In a few approaches dos Santos et al. (2021), it has been added in the feature model itself with additional notations. Few approaches Asadi et al. (2014), Siegmund et al. (2012), Soltani et al. (2012) does not provide any support for specifying context information. Notably, the approach by Inverardi et al. Inverardi and Mori (2010) uses separate artifact for modeling the context information separating it from the basic feature model. Inspired by this study, having a separate context model would be essential when the number of context information and feature grows in size.

TABLE 3.3: Comparison of existing approaches with respect to specifying energy-awarenss and self-adaptation.

Approach	**Configuration Time**	**Model Type**	**Energy-awareness**	**Context Model**	**Adaptation Planning**	**Tool support**
CODA (Desmet et al., 2007)	Design Time	Graphical	Partial	Attributes	Table	No
Inverardi and Mori (2010)	Requirements, Design Time	Graphical	No	Separate	Table	No
Soltani et al. (2012)	Design Time	Graphical	Partial	No	No	Yes
Siegmund et al. (2012)	Design, Run Time	NA	No	No	No	Yes
Asadi et al. (2014)	Design Time	Graphical	No	No	No	Yes
Pascual et al. (2015a)	Run Time	Graphical	Partial	Feature Model	Yes	Yes
CVL (Pascual et al., 2015b)	Design, Run-time	Graphical	Partial	Partial	Yes	Yes
Mauro et al. (2018)	Design-time	Graphical, textual	No	Hybrid	Yes	Yes
Baddour et al. (2019)	Design-time	Graphical	No	Goal Model	Yes	Yes
Duhoux et al. (2019)	Run-time	Graphical	No	Feature Model	Yes	Yes
dos Santos et al. (2021)	Run-time	Graphical	Partial	Feature Model	Yes	Yes
This Approach	Early Stages	Graphical, Textual	Primary	Feature Model	Yes	Yes

With respect to *support for self-adaptation planning*, it is observed from the related works, none of the approaches have a dedicated module for planning the adaptation. However, decision tables Desmet et al. (2007) and multi-objective algorithms Pascual et al. (2015a) approaches also were found for automated reconfiguration generation. The decision tables would not serve the purpose of adaptation planning as it does not have an automated validation method. On the other hand, having a sophisticated optimization technique would be more suitable for run-time configuration generation. As the scope of this research work is in the domain analysis phase, extension to the cross-tree constraints and configuration generator of FeatureIDE Thüm et al. (2014) has been provided to validate the configuration along with the dedicated module for planning the energy-saving self-adaptation. With respect to *tool support*, it has been learned that a tool is essential in this research area. Therefore, to provide the tool support, an extension to popular feature modeling tool FeatureIDE Thüm et al. (2014) has been developed.

Overall, the existing approaches Mauro et al. (2018), Pascual et al. (2015b), Siegmund et al. (2012), Soltani et al. (2012) are involved in optimizing the reconfiguration generations, which is needed during the run-time. In addition, few approaches focus on providing design-time models to support the context representation. However, since the energy-aware self-adaptive software is the new problem domain, it is better to have a framework that produces artifacts during the requirements time. The only approach by Inverardi and Mori (2010) provides suitable notations for specifying and analyzing the requirements and self-adaptivity at requirements time. Therefore, this thesis focuses on providing support at the early stages of software development. On the other hand, to the best of our knowledge, none of the existing approaches provides explicit support for specifying and analyzing energy-related information to provide energy-saving adaptation plans. Though few approaches provide partial support through attributes Desmet et al. (2007) and annotations Soltani et al. (2012) of feature diagrams, it may not be sufficient enough for planning better energy-savings. Therefore, to fill this research gap, an explicit framework and tool support has been provided for energy-aware requirements specification and energy-saving adaptation planning.

3.3.2 Model-driven development approaches

In literature, several academic research approaches have been found about model-driven development of mobile applications. Primarily, these approaches build tools that contain a Domain-specific language and code generator. Here, domain specific-languages are used for specifying the app functionalities. The code generators are used for generating source code for target platforms like Android and iOS. Several approaches generate only native application source code targeted at a single platform, while other approaches serve as a cross-platform code generator to generate code for multiple platforms. This subsection discusses relevant academic domain-specific language and code generator for smartphone applications.

MD^2 (Heitkötter et al., 2013) is an approach for developing mobile apps with model-driven development methods. It consists of a domain-specific language to specify the data-driven business apps. It also contains a separate code generator for generating native Android and iOS code. The language and code generated by MD^2 follow the Model-View-Controller (MVC) pattern. Here, the *Model* component allows the developers to define the application's data model. The *View* component helps in describing the user interface and its elements. The *Controller* component aids to describe the user interaction and events associated with the apps. The DSL was defined with Xtext, and Xtend defines the code generator. The recent version of MD^2 (Heitkötter et al., 2015) includes the following capabilities: *device-specific layout, extended control structures, and offline computing*.

Xmob (Le Goaer and Waltham, 2013) is a platform-independent DSL for creating mobile applications for multiple platforms. It is developed with three sub-languages (Xmob-data, Xmob-ui, Xmob-event) to follow the MVC pattern. The *Xmob-data* helps the developers to specify the way retrieving form database, web service, or other data sources. The *Xmob-ui* helps the developers to describe the UI elements such as widgets, forms, buttons, etc. The *Xmob-event* helps the developers to link the user interfaces and data sources. Xmob involves model-to-model transformation and model-to-text transformation to generate the source code of the desired platform. The model-to-model transformation converts the platform-independent model to a platform-specific model. The model-to-text generates the source code corresponding to the elements in a platform-specific model. Xmob uses Xtext for language definition, Kermeta for model-to-model transformation, and Xpand for the code generator.

ADSML (Jia and Jones, 2015) is an adaptive domain-specific modeling language for native mobile app development. The syntax and rules of ADSML evolve in line with the evolution of the target platform's evolutions. It relies on meta-model extraction, meta-model elevation, meta-model alignment, and meta-model unification to create target apps for the Android and iOS platforms. The *meta-model extraction* phase extracts the platform-specific meta-models from the targeted platforms native APIs. The *meta-model elevation* phase abstracts the platform-specific API models and select the subset for further analysis. The *meta-model alignment* phase find out the similar meta-model elements among different platforms. Finally, the *meta-model unification* phase creates the platform-independent DSL from the platform-specific models identified in the previous phase. The current implementation of ADSML does not have the support for code generation. The authors claim that the ADSML is adaptable to the target platform's evolution and develops the high-performance native application.

DSL-Comet (Vaquero-Melchor et al., 2017) is the active DSL that targets a smart city or IoT applications. It primarily runs on mobile devices to tag the location and contextual information on the model elements created by DSL-Comet. The DSL-Comet includes Open, Geo, and Contextual DSLs to form an active DSL. The *Open DSLs* interact with external APIs to retrieve the information related to model elements. The *Geo DSLs* render the models on the map interface to tag the current location on the models associated with geo-services. The *Contextual DSLs* are context-aware and helps to re-organize the model after encountering the contextual changes. It has iOS and Eclipse-based editors that permits the users to model either on the mobile or desktop. The iOS editor stores the models in JSON format, and the Eclipse-based editor stores the models in XMI format. The DSL-Comet does not have a code generator to generate source code for the targeted platform.

Rapid APPlication Tool (RAPPT) (Barnett et al., 2019) aids the developers in specifying the characteristics of mobile applications using domain-specific visual language and textual language. Initially, the developers can use visual language to specify the high-level architecture and the number of screens with navigation. The developers can then use textual language to add more specific information, such as data schema, authentication, web service, etc., to define the app. The combination of the model described by the visual language and textual language is called the App model. The model-to-model transformation then takes place to convert the app model to the Android model, where high-level specification will be transferred to Android-specific elements such as classes, activities, fragments, etc. Finally, the RAPPT generates the source code from the Android model that resembles the developer's written code. The generated code produces the working prototype, and developers need to add the business logic to deliver the working application.

MoWebA Mobile (Núñez et al., 2020) is a model-driven approach covering the mobile apps' data layer. This approach mainly defines the data source of application to develop offline access to business applications in case of network connectivity issues. This approach consists of three phases: (1) Problem Modeling, (2) Solution Modeling, and (3) Source Code Definition. The *problem modeling* phase uses the Computational-independent Model (CIM) and Platform-independent Model (PIM). The *solution modeling* phase uses the Architecture-specific Model (ASM) to specify the architectural requirements. It uses UML profiles to create Platform-independent models and EMF to convert the PIM to ASM. Finally, it uses Acceleo to transform models to generate code for developing native applications for Android and Windows platforms.

The comparison of the model-driven development of mobile apps is given in Table 3.4. The approaches are compared based on the following criteria: *DSL Type, Targeted Platform, Domain, Modeling Scope, Support for Context-awareness, and Support for Energy-awareness*. The existing approaches are compared with the concepts presented corresponding to the *e*GEN tool described in Chapter 7. As shown in Table 3.4, the considered approaches can be broadly classified into two categories, namely, *Textual and Graphical* based on the DSL Type. The approaches such as MD^2 (Heitkötter et al., 2013) , Xmob (Le Goaer and Waltham, 2013), ADSML (Jia and Jones, 2015) uses the *textual DSL* to specify the app functionalities. The *graphical DSL* is used in the approaches like DSL-Comet (Vaquero-Melchor et al., 2017), RAPPT (Barnett et al., 2019), and MoWebA Mobile (Núñez et al., 2020).

TABLE 3.4: Comparison of model-driven development approaches for mobile app development

Approach	**DSL Type**	**Targeted Platforms**	**Domain**	**Modeling Scope**	**Context-awareness**	**Energy-awareness**
MD^2 (Heitkötter et al., 2013)	Textual	Android and iOS	Data-driven Business Apps	Data, UI and User Interaction	No	No
Xmob (Le Goaer and Waltham, 2013)	Textual	Android, iOS, and Windows	All Mobile Apps	Data, UI and Events	No	No
ADSML (Jia and Jones, 2015)	Textual	Android and iOS	All Mobile Apps	All aspects	No	No
DSL-Comet (Vaquero-Melchor et al., 2017)	Graphical	NA	Smart City Applications	Business Functions	Yes	Partial
RAPPT (Barnett et al., 2019)	Graphical and Textual	Android	All Mobile Apps	Views	No	No
MoWebA Mobile (Núñez et al., 2020)	Graphical	Android and Windows	Offline Business Apps	Data Layer and Network Connectivity	No	No

In this research work, *e*GEN tool adopts the textual DSL for modeling the energy-saving self-adaptive requirements of smartphone applications. The *Target Platform* criteria refer to the mobile platform for which the source code generated by the code generator associated with the discussed tools. Most of the approaches generate code for multiple platforms such as *Android, iOS, and Windows*. The RAPPT (Barnett et al., 2019) approach considers only the Android platform for code generation. In this research work, the *e*GEN tool covers only the Android platform, and other platforms will be considered in the future releases of the tool. The *Domain* criteria refer to the application domain covered by the DSL and code generator. As shown in Table 3.4, most of the approaches cover all the aspects of mobile apps. In contrast approaches such as MD^2 (Heitkötter et al., 2013), DSL-Comet (Vaquero-Melchor et al., 2017), and MoWebA Mobile (Núñez et al., 2020) covers the specific application domains. Specifically, the MD^2 (Heitkötter et al., 2013) is for data-driven business apps, DSL-Comet (Vaquero-Melchor et al., 2017) is for smart city applications, and MoWebA Mobile (Núñez et al., 2020) is for business applications with offline access. As observed from the Table, none of the approaches have considered location-based Android applications. Therefore, in this approach, family of location-based applications has been considered as the application domain for DSL and code generator. The *modeling scope* criteria refer to the elements that can be modeled with the DSL provided in the related approaches. As shown in Table 3.4, most of the approaches cover the data and UI modeling of mobile apps. None of the existing approaches have considered modeling the location-sensing of mobile apps. In contrast, this research work's modeling scope covers the location-sensing of mobile apps. Finally, the existing approaches have been compared for *self-adaptivity and energy-awareness* support. As observed from Table 3.4, none of the existing approaches has considered the self-adaptivity and energy-awareness of the mobile apps, which is the essential non-functional requirements for the recent generation smartphones. Therefore, in this research work, the modeling of energy-awareness and self-adaptivity has been considered for location-based Android applications.

3.4 Research Gaps

Overall, the post-development solutions suggest ways to improve coding practice for energy-efficiency, demanding a better way of developing applications among the software developer communities. At the same time, educating such a large number of developers will be difficult, directly at the code level. Therefore, it is necessary to consider energy-efficiency as a critical component during the early stages before making any decisions on lower level code development. Under pre-development approaches, a significant amount of work has been done on design time and run-time models for self-adaptive software. Further, some research works have used feature models to model the system's features and context information for dynamic self-adaptivity. However, none of the research works attempted to model the energy-aware context and adaptation plans. This research work aims to provide a modeling approach and tool support based on feature models to consider energy-related requirements in the early software development phases. Here, our objective is to help the designer find out the energy-saving self-adaptations at design time with the feature-based modeling framework. This research work aims to address the following Research Gaps (RG):

RG1 Limited information existing energy-saving practices and API usage patterns

RG2 Lack of domain analysis framework to analyse the energy saving opportunities

RG3 Lack of domain-specific modeling tools to model the energy related context information and energy-savings adaptations

RG4 Lack of automated tools to assist the developers by generating energy-aware source code artifacts

Chapter 4

Empirical Studies

This chapter presents the results of Research Objective 1 (RO1), including the empirical studies conducted on the location-based Android applications. The Empirical software engineering studies such as controlled experiments, qualitative analysis of Stack Overflow data, and GitHub data are included in this objective. The primary purpose of RO1 is to analyze the effectiveness of location-based Android applications' existing energy-saving techniques. This objective's first contribution is to check the cause-effect relationship between location-sensors and energy consumption, as presented in Section 4.2. The second contribution is to analyze Stack Overflow data to document the existing knowledge base of energy-related issues of location-based Android applications, as reported in Section 4.3. The third contribution is to analyze GitHub commits to identify the research efforts needed to fix energy consumption-related issues of open-source location-based Android applications after deployment, as presented in Section 4.4.

Despite the widespread interest in using location-based applications, there is little knowledge of energy-saving programming practices that are being followed by expert developers. It was initially challenging to introduce self-adaptive solutions, feature-oriented domain analysis, and model-driven solutions without a conceptual starting point. Therefore, it has been decided to identify the contribution of location-based applications in the smartphone device's overall battery consumption. In addition, organizing the existing developer knowledge about Location-based Android applications' energy-saving practices is also considered before proposing a model-driven solution. Therefore, this objective employs empirical software engineering methods such as controlled experiments and mining software repositories to organize the existing knowledge. The remaining part of this chapter presents the research questions, methodology, and findings of the various empirical studies conducted on location-based Android applications.

4.1 Research Questions and Methodology

The following research questions have been formed to document the existing energy-saving techniques and their effectiveness in the location-based Android applications domain:

- **RQ1** What is the contribution of LBAs to the overall energy consumption of a Smartphone?
- **RQ2** Does the use of inertial sensors reduce energy consumption?
- **RQ3** Which App / API best utilizes the inertial sensors at run-time to save energy?
- **RQ4** What are the most common energy-saving solutions suggested by the developers?
- **RQ5** What are the popular energy-saving location API usage patterns?
- **RQ6** How much development effort does it take to improve energy-efficiency after deployment?

4.1.1 Research methodology

The guidelines given by Easterbrook et al. (2008) have been used to choose empirical methods, and it is found that a single data source and empirical method cannot answer all the research questions. Therefore, three different data sources have been used with a suitable empirical method to answer the research questions as below:

- **For RQ1, RQ2, and RQ3** Controlled experiments have been selected as answers to the questions demand data from existing applications.
- **For RQ4 and RQ5** Qualitative data analysis has been selected as the answers demand developers' experience.
- **For RQ6** Mining software repositories have been selected as the answer demands data from development effort at the source code level.

4.1.2 Variable selection for empirical study

As the main objective of this study is to analyze the cause-effect relationships between location sensors usage and energy consumption of location-based application, the relevant independent and dependant variables are selected as follows:

- **Independent variables** The *Number of sensors usage* has been selected as a primary independent variable as the usage of inertial sensors like accelerometer might reduce energy consumption. Further, *GPS and sensor active time* has also been used as an independent variable in a controlled experiment.
- **Dependent variables** The *Energy consumption or Battery Drop* of LBAs is the first affected variable because of GPS and location sensor usage. Therefore, it is selected as a dependant variable to verify the cause-effect relationship of sensors usage and energy consumption in the controlled experiment. Second dependent variable is *Accuracy* of location sensing. Improving the energy-efficiency of location-sensing might directly affect the accuracy of the fetched location. Therefore, *Accuracy* is considered a dependent variable for mining Stack Overflow discussions and GitHub commits.

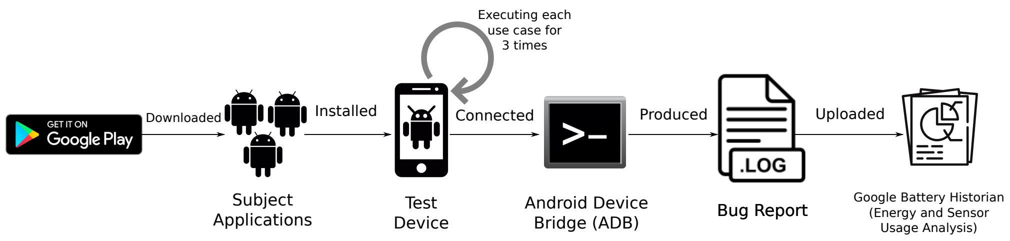

Executing each use case for 3 times
GET IT ON Google Play
Downloaded
Subject Applications
Installed
Test Device
Connected
Android Device Bridge (ADB)
Produced
.LOG
Bug Report
Uploaded
Google Battery Historian (Energy and Sensor Usage Analysis)

FIGURE 4.1: Experiment protocol of controlled experiment.

4.2 Controlled Experiments

The controlled experiment has been carried out by executing the subject map navigation applications on the testing device. The experiment protocol and results are presented in this sub-section.

4.2.1 Experiment protocol

The experiment has been conducted on the selected map navigation subject application running on Android devices. As shown in the experiment protocol (see Figure 4.1), the selected applications have been installed on the test device, *Google Nexus 6* running *Android 7.1.1*. The test device was prepared for the experiment by disabling all background and scheduler services. The device was allowed to access Mobile data only for the application under test to minimize unwanted battery consumption by other applications. Each selected subject application was installed and executed separately to collect the required data. A well-defined usage scenario has been used across all the applications to ensure the replicability of the experiments. The experiment was carried out near Surathkal town by navigating from *Surathkal Bus Stand* to *Sadashiva temple*. Each trial took approximately *2.1 Kilometers*, and the destination was reached by cycling from the source. The use case was repeated three times per application, and the aggregated value was considered for analysis. The *Android Device Bridge (ADB)* was used to export the bug reports or log files. The bug reports produced by ADB were considered for energy consumption and sensor usage analysis. The exported bug reports have been further uploaded to *Google Battery Historian* tool to extract the energy consumption data, GPS usage, and other inertial sensors usage data. Finally, the numerical results have been analyzed with suitable analysis techniques.

TABLE 4.1: Selected subject applications for controlled experiment

AppID	AppName	Category	Map Type	Location Provider
GM	Google Maps	Navigation	Online	Google
MM	Maps.ME	Navigation	Offline	Open Street Map
OA	OsmAnd	Navigation	Offline	Open street Map

4.2.2 Subject applications selection

Table 4.1 shows the list of subject applications that have been considered for a controlled experiment. The subject application is the popular map navigation application running on Android devices. The *Google Play Store* has been used as the repository to search and download the subject applications. The subject application is selected if it satisfies the following criteria:

- If the subject application detects the source and destination of use case locations
- If the subject application is compatible with Android versions 5.x to 8.x.
- If the subject application provides directions from source to destination

4.2.3 Data collection and analysis

The required data have been collected by uploading the bug report of each trial separately to Google Battery Historian. The data such as *energy consumption, number of GPS usage, used inertial sensors with its frequency of usage* have been collected from execution traces. The collected data have been analyzed quantitatively by selecting the appropriate data analysis technique. For RQ1, the data have been collected under each application and tabulated. A multi-axis bar and line plot have been selected and presented for the cause-effect relationship between sensor usage and energy consumption.

4.2.4 Answering RQ1: LBAs energy consumption

The aim is to determine the energy consumption contributed by LBAs to the smartphone device's overall power consumption for a given time. The energy consumption value

TABLE 4.2: Energy contribution of each LBA to overall energy consumption of device

Trial ID	Total Battery Drop	By App	By Screen	By Others (approx.)
GM1	5%	1.43%	2.8%	0.77%
GM2	4%	1.69%	2.11%	0.2%
GM3	5%	1.86%	2.35%	0.79%
MM1	10%	2.34%	3.32%	4.34%
MM2	8%	2.01%	2.11%	3.88%
MM3	8%	2.08%	3.79%	2.13%
OA1	8%	3.39%	3.42%	1.19%
OA2	8%	2.21%	3.57%	2.22%
OA3	8%	2.17%	3.48%	2.35%

is reported as a percentage of battery drop. Table 4.2 shows the percentage of battery drop by each selected subject application under each trial. In Table 4.2, *Total battery drop, Appwise battery drop, screen battery drop, and other battery drops (which is an approximated value)* are collected from the bug reports and tabulated. The battery drop by *screen* stands on top of all other components as the screen was ON during the navigation. The next major contributor is *Core application logic* of map applications. The battery drop by *Others* is derived by subtracting the sum of *App battery drop and screen battery drop* from *Total battery drop*. the results show that *screen* contributed *42.1%*, *app* contributed *30%* and *other components* contributed *27.9%* on an average to overall battery drop of the smartphone device. The tabulated results show that LBAs are one of the major contributors to the smartphone device's overall power consumption. Hence, the location sensing, navigation, map display of the LBAs must be optimized further to reduce the abnormal battery drain.

4.2.5 Answering RQ2: Inertial sensors usage and energy consumption

Here, the aim is to prove that inertial sensors' usage at run-time reduces energy consumption with empirical evidence from controlled experiments. The inertial sensors' usage has been extracted from each trial's bug report, and the extracted data is used for the analysis against *battery drop*. The data analysis has been carried out to report the impact of sensors used on the application's overall energy consumption. The data

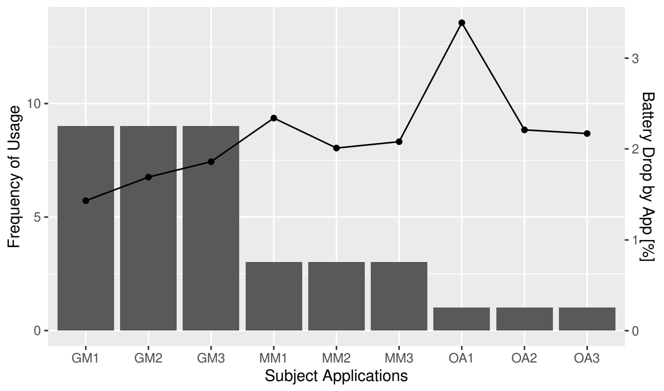


FIGURE 4.2: Impact of sensor usage on overall battery drop.

obtained from subject applications' sensor usage and battery drop percentage have been used to show the impact of sensor usage. Here, the variables *Frequency of sensors usage* is an independent variable while *Battery drop percentage* is dependant variable. The two selected variables data are plotted in a single graph containing a bar plot for the independent variable and one line plot for the dependent variable with the secondary Y-axis. Figure 4.2 shows the impact of the sensor's usage on the overall battery drop percentage. As shown in Figure 4.2, *Google maps* contributed less compared to the other two navigation applications. As nine inertial sensors are used along with GPS by the *Google maps* application, the battery drop is less in all three sample trials (GM1, GM2, and GM3). In case of *Maps me* application, the number of sensor usage reduced to *three*. Consequently, the app battery drop has increased on the other side (MM1, MM2, and MM3). Similarly, the sensor usage is further dropped to *one* in *OsmAnd* application, and the battery drop increased further as recorded in OA1, OA2, and OA3. Hence, from the preliminary results, it is inferred that *more sensor usage* will reduce the *overall battery drop*.

4.2.6 Answering RQ3: Inertial sensors usage by API

Here the aim is to determine the type of inertial sensors used by the navigation application (APIs) at run-time. Generally, the sensors are used through API calls, and the developers are unaware of the sensors used at run-time. This data will give developers a clear idea about the type of sensors used by the APIs to select the energy-efficient

TABLE 4.3: Inertial sensor usage per API at run-time

Sensor ID	Sensor Name	Google	Open Street Map
Ori	Orientation		✓
Mag	Magnetometer	✓	✓
Acc	Accelerometer	✓	✓
Gyro	Gyroscope		✓
StepD	Step Detector	✓	
RotV	Rotation Vector	✓	
GRot	Game Rotation Vector	✓	
GyroU	Gyroscope Uncalibrated	✓	
LAcc	Linear Acceleration	✓	
Grav	Gravity	✓	
MagU	Magnetometer Uncalibrated	✓	

location API. As reported in Table 4.3, the two location APIs, such as Google location API and Open Street Map API, have been primarily used by the subject applications. To be more specific, *Google Maps* uses Google location API while *Maps.Me* and *OsmAnd* use Open Street Map API. Google API uses more sensors such as *Magnetometer, Accelerometer, Step Detector, Rotation Vector, Game Rotation Vector, Gyroscope Uncalibrated, Linear Acceleration, Gravity, Magnetometer Uncalibrated* which is common for all the trials (GM1, GM2, and GM3). Whereas, the *Open Street Maps* API uses few sensors such as *Magnetometer, Accelerometer, Gyroscope, Orientation* sensor.

The data on sensor and GPS active time have been extracted for each trial and reported in Table 4.4 to show the API that better utilizes the sensors at run-time for energy-saving. The app *OsmAnd*, used only *Orientation sensor* in all trails (OA1, OA2, and OA3), and the sensor has been active along with GPS all the time, thus resulted in increased battery drop. Similarly, the *Maps.Me* application uses *Magnetometer* along with GPS all the time. In addition, this app kept *Accelerometer* active for more time (approx. 4 minutes) which contributes to less battery drop compared to *OsmAnd*. This shows the battery drop difference between using *Orientation sensor and magnetometer*. Further, as in the Table 4.4, *Google Maps* uses more sensors but for very less amount of time (less than a minute) and showed less battery drop among all subject applications. These results conclude that the battery drop of LBAs can be reduced by inertial sensors usage provided that suitable sensors are used only when required. Hence, *Google API* better utilizes the inertial sensor for energy-efficiency compared to *open street map API*.

TABLE 4.4: Application active time vs GPS and sensors active time

Trial ID	App Active	GPS	Ori	Mag	Acc	Gyro	StepD	RotV	GRot	GyroU	LAcc	Grav	MagU
GM1	9m 50s 827ms	9m 50s 805ms	NA	31s 193ms	6s 342ms	NA	31s 765ms	31s 261ms	31s 141ms	17s 144ms	17s 142ms	17s 116ms	14s 777ms
GM2	12m 6s 605ms	12m 7s 96ms	NA	45s 627ms	8s 421ms	NA	46s 660ms	45s 698ms	45s 604ms	21s 519ms	21s 511ms	21s 494ms	21s 61ms
GM3	11m 25s 239ms	11m 25s 255ms	NA	39s 260ms	7s 82ms	NA	39s 885ms	39s 301ms	39s 207ms	18s 576ms	18s 574ms	18s 546ms	16s 475ms
MM1	12m 35s 195ms	12m 17s 222ms	NA	12m 0s 333ms	4m 48s 893ms	24ms	NA	NA	NA	NA	NA	NA	NA
MM2	12m 16s 265ms	11m 55s 173ms	NA	11m 52s 156ms	4m 43s 149ms	6ms	NA	NA	NA	NA	NA	NA	NA
MM3	12m 41s 536ms	12m 33s 341ms	NA	11m 42s 43ms	4m 40s 631ms	7ms	NA	NA	NA	NA	NA	NA	NA
OA1	12m 43s 728ms	12m 42s 443ms	12m 42s 406ms	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
OA2	11m 40s 969ms	11m 39s 774ms	11m 38s 616ms	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
OA3	11m 26s 600ms	11m 25s 366ms	11m 22s 685ms	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

4.3 Mining StackOverflow

This empirical study aims to organize the developer's discussions on Stack Overflow [1] related to improving location-based Android applications' energy-efficiency. The qualitative data analysis method has been used to answer the research questions. The domain-specific terms and most common energy-saving solutions identified as part of the answer to RQ4 would serve as the body of knowledge in designing the case studies presented in the later chapters. The expert developer's energy-saving usage patterns identified as part of the answer to RQ5 have been used to design and develop Domain-specific language and code generator described in chapters 6 and 7. This sub-section presents the process of mining stack overflow data and answers to the research questions.

4.3.1 Mining protocol

The mining process is two-fold. The first one is an automated method to collect the data, and the second method is semi-automated qualitative data analysis. The detailed steps involved in the mining process are depicted in Figure 4.3. The first step in the mining process is to query the Stack Exchange Data Explorer [2], which is a data dump of all questions and answers of Stack Overflow. The ability to query the data source is given through the SQL query interface. The returned data sets can be downloaded as a .csv file for further analysis. The keywords *android, location, gps* on the fields *title, tags* have been used. The query was made on July 04, 2018 (12:45PM IST), with the relevant SQL queries. The SQL query returned a total of 11,911 questions in the form of a .csv file. This file is our raw data set, which is used for the further filtering process. The raw data set has been further processed with suitable Python scripts to get the candidate studies containing the energy-related questions. The python pre-processing returned almost 651 questions, which have been considered as candidate sets for data analysis.

[1]https://stackoverflow.com/
[2]https://data.stackexchange.com/

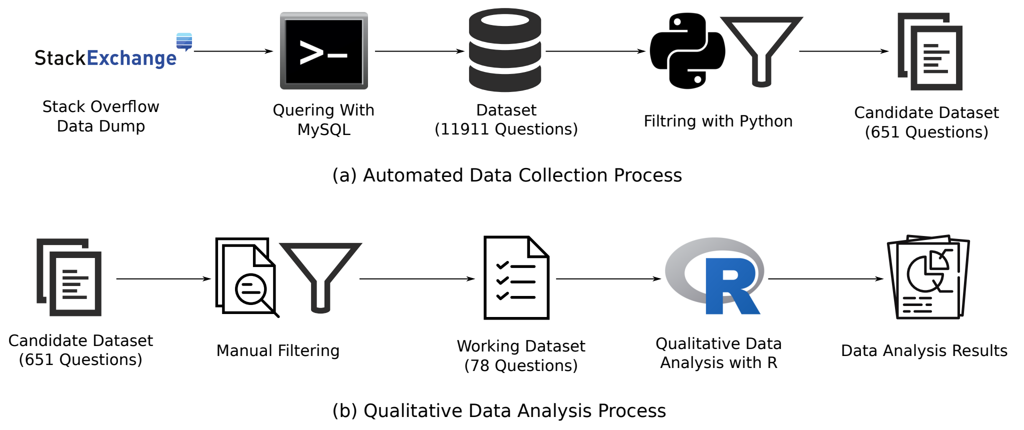

StackExchange
Stack Overflow Data Dump
Quering With MySQL
Dataset (11911 Questions)
Filtring with Python
Candidate Dataset (651 Questions)
(a) Automated Data Collection Process
Candidate Dataset (651 Questions)
Manual Filtering
Working Dataset (78 Questions)
Qualitative Data Analysis with R
Data Analysis Results
(b) Qualitative Data Analysis Process

FIGURE 4.3: Overall mining protocol of mining Stack Overflow.

4.3.2 Data collection and analysis

The 651 candidate Stack Overflow posts have been considered for manual analysis. The questions with *Unaccepted answers* have been rejected from the candidate data set, and the relevant questions are reduced to 78 in numbers. The working data set is analyzed using the R programming language[3]. Specifically, the R Package for Qualitative Data Analysis (RQDA)[4] has been used. The relevant codes have been manually marked by reading each question and its accepted answer. Broadly two code categories have been identified: *Goals Category* (Energy, Accuracy, Energy-Accuracy) and *Solutions Category* (Refer Figure 4.4). The collected data are further analysed to answer the research questions. The RQ4 has been answered by taking the frequency of occurrences of each energy-saving technique, and the data has been plotted in Figure 4.4. The RQ5 has been answered by manual source code analysis of the code segments, and the usage patterns are presented as Figures (From Figure 4.5 to Figure 4.14).

4.3.3 Answering RQ4: Energy-saving solutions

What are the most common energy-saving solutions suggested by the developers? The energy-saving solution category has been identified by reading the title, body filed

[3]https://www.r-project.org/
[4]http://rqda.r-forge.r-project.org/

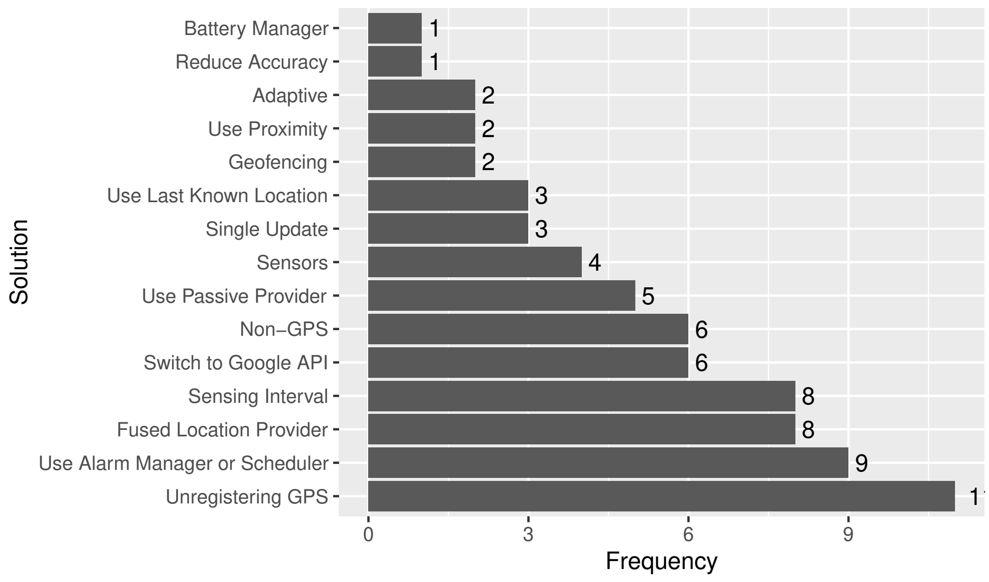


FIGURE 4.4: Most common energy-saving solutions.

of questions, and expert developers' responses. The thematic analysis has been used to answer this research question. Finally, *fifteen* themes have been found and each themes frequency is plotted as shown in Figure 4.4. Each theme in Figure 4.4 corresponds to an energy-saving solution suggested by the expert developers commonly occurring energy-related problems in location-based applications. Overall, the expert developers' solutions cover two popular location APIs, namely *Android Native Location API* and *Google Location API*. The following solutions correspond to *Android Native Location API*: *Unregistering GPS, Non-GPS, Reduce Accuracy*. The following solutions correspond to *Google Location API*: *Geofencing, Switch to Google API, Sensing Interval, and Fused Location Provider*. The other solutions such as *Battery Manager, Adaptive, Use Proximity, Use Last Known Location, Single Update, Sensors, Use Passive Provider, and Use Alarm Manager* correspond to both the APIs. These solutions specifically discuss the way of using existing API calls to reduce unnecessary battery consumption. A brief discussion of each energy-saving solution is given below:

- **Unregistering GPS**: This solution was suggested to reduce the unwanted energy drain by continuous GPS sensing. After unregistering, the GPS will immediately stop looking for location updates. This solution can be applied when the application requirements do not demand continuous GPS sensing.

- **Use Alarm Manager or Scheduler**: Unregistering GPS requires manual intervention to register it again. This is not suitable for the LBAs where the requirements change most often at run-time. Therefore, the usage of an alarm manager or scheduler was suggested by the developers. This solution provides the ability to programmatically schedule the GPS re-registering activity after a specific time interval or distance. Unfortunately, these two options have been depreciated in recent Android versions.

- **Fused Location Provider**: It is the best solution suggested by many developers to use Google location API instead of Android native location API. The Google location API calls choose the best location provider by considering the dynamic energy and accuracy requirements. The developer may not have control over selecting the location source at run-time as it is a black box technique.

- **Sensing Interval**: It has been used to stop the continuous polling of GPS requests when not needed. The sensing interval informs the time difference between the subsequent location-sensing requests. More prolonged the sensing interval results in lesser energy consumption.

- **Switch to Google API**: In the older version of Android (V3.x.x.), *Native location API* was used, and it was not energy-efficient. The developers have to write a lot of configurations to make it energy-efficient. Therefore, Google provided location API, which abstracts APIs' usage with custom parameters and state of the art energy-saving techniques. Google API consumes less energy compared to native APIs. Therefore, expert developers suggested migrating the location-sensing code from Android native API to Google location API.

- **Non-GPS**: The selection of location sources is controlled by the developer in the Android native location API. The expert developers suggested non-GPS solutions like Wi-Fi, CellID, Bluetooth, and other location sensors as they consume less energy than GPS. On the other hand, these sources do not provide accuracy equal to GPS and not suitable for applications that require high accuracy.

- **Use Passive Provider**: Instead of polling GPS for every new request, the location data can be obtained from the other app, which recently accessed the GPS. The number of GPS usage could be reduced with this technique. Thus it reduces energy consumption.

- **Sensors**: The most recent energy-saving technique widely used nowadays in most applications is to combine activity recognition with GPS. Sensors like accelerometers, magnetometers, gyroscopes, etc., are used for activity recognition. These sensors are used to detect user activity like moving to still. The sensors can be combined to check whether the user has moved from the last location before creating a new GPS request. A new GPS request can be created if the user has moved from the previously known location. Otherwise, the new GPS request can be avoided. Hence, the usage of the sensor reduces the GPS usage and supports energy-saving.

- **Single Update**: This is one of the energy-saving techniques used in Android native location APIs. This option updates the location only once and deactivates it automatically without a separate deactivating mechanism. This is helpful in the accidental misuse of GPS when not needed.

- **Use Last Known Location**: The last known location was useful when the user is not moved from the place when the previous location was fetched. Here, instead of using GPS polling, the last retrieved location can be directly used to save the battery.

- **Geofencing**: It is part of Google location API and uses users' current location and proximity to detect the user's entry and exit to the defined geofence boundary. The systems send events to the user when the entry or exit happens with the particular geofence.

- **Use Proximity**: It is an alert used in Android native location API to find out the user's entry or exit at some location to trigger the predefined events. Similar to Geofence, the interesting location will be predefined to set the alert of entry/exit.

- **Adaptive**: The adaptive methods are used for balancing energy-accuracy requirements. In general, this method combines existing energy-saving techniques and calls it when a suitable situation is sensed for the dynamically changing requirements.

- **Reduce Accuracy**: It is not recommended as it reduces the accuracy requirements to reduce energy consumption. Some applications may not give the expected output when the location accuracy is compromised.
- **Battery Manager** This is one of the recent methods introduced and quite successful in balancing energy-accuracy requirements. Here, the battery manager is used to get the battery percentage and state to decide the suitable method to be called adaptively at run-time.

As summarized above, most of the discussed energy-saving techniques are pretty old, and for Android native location API, which is no longer supported by the recent Android versions. Solutions like Fused location, Geofence are related to Google location APIs, and they are continuously evolving in the recent Android versions. The most promising solution for the new generation smartphones is using adaptive strategies with the battery manager to balance energy-accuracy requirements. Therefore, combining battery manager results and adaptive location-sensing strategies would reduce unwanted battery consumption while meeting dynamic accuracy requirements. Hence, the later chapter's case studies have been designed by considering battery percentage and charging state as the essential criteria to introduce self-adaptive location-sensing strategies for energy-savings.

4.3.4 Answering RQ5: API usage patterns

What are the popular energy-saving location API usage patterns?
The code segments from the accepted solutions of candidate Stack Overflow questions have been analyzed to answer this research question. The selected questions mostly discuss about using Android Native APIs and Google location APIs. Through manual analysis, ten different Usage Patterns (UP) have been identified and depicted in Figure 4.5 to Figure 4.14. These identified usage patterns will help to design the grammar for domain-specific modeling language and code generator in Research Objective III and Research Objective IV. The analysis of these usage patterns would help the software architects choose suitable usage patterns for their location-sensing requirements. This Subsection presents the identified usage patterns through the manual analysis of source code segments.

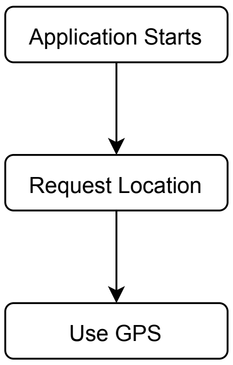


FIGURE 4.5: Energy-hungry (UP01)

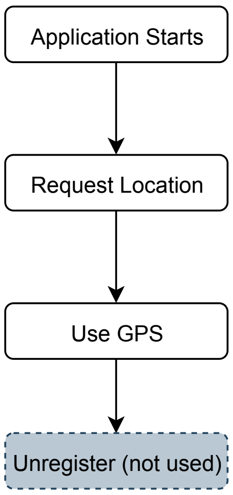


FIGURE 4.6: Unregistering GPS (UP02)

Energy-hungry (UP01)

This is a fundamental usage pattern and considered to be an energy-inefficient method as it is not unregistering the GPS after its use as shown in Figure 4.5. This pattern provides accurate location-sensing while results in abnormal battery drain by polling the GPS even when the application no longer needs the location updates.

Unregistering GPS (UP02)

This is an energy-efficient version of *UP01* as it unregisters the GPS explicitly after its use as shown in Figure 4.6. The unregister GPS call will explicitly kill the location-listener object and avoid the unnecessary polling of GPS when the application doesn't need the location. This method requires to restart the application when the application needs location while using the application. Because of the strict unregistering behavior, this method might not be suitable when the location requirement changes dynamically.

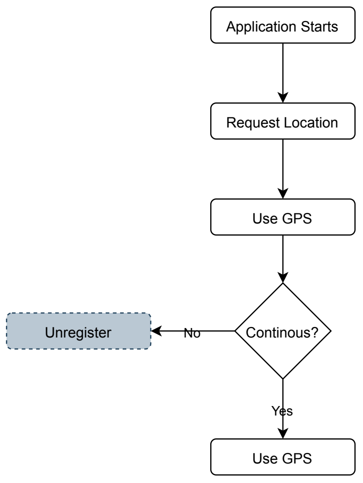


FIGURE 4.7: Continuous GPS sensing (UP03)

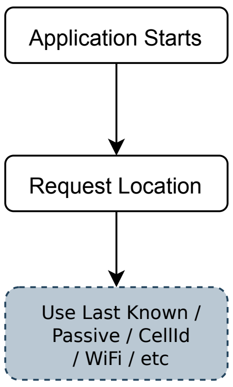


FIGURE 4.8: Energy-efficient alternatives (UP04)

Continuous GPS sensing (UP03)

This method is suitable for a situation where continuous GPS sensing is needed and can be unregistered when the application does not require location-sensing. Figure 4.7 shows that this usage pattern allows developers to check the need for continuous location-sensing. The location listener is kept active if the application requirements demand continuos location-sensing. Otherwise, the GPS request can be unregistered. This method is ideal for the application, which may later need the location data continuously. Later, the GPS can be unregistered when the application is terminated or in the background depends on its requirements. However, energy consumption by continuous GPS sensing cannot be avoided as the applications send high accurate location data continuously.

Energy-efficient alternatives (UP04)

This usage pattern uses alternative location sources instead of selecting GPS for location-sensing. As shown in Figure 4.8, the developers could use the last known location or passive location if the application does not require accurate location-sensing. This usage pattern might not be suitable where the application needs accurate location data. This usage pattern is considered to be an energy-efficient method as it reduces the new GPS

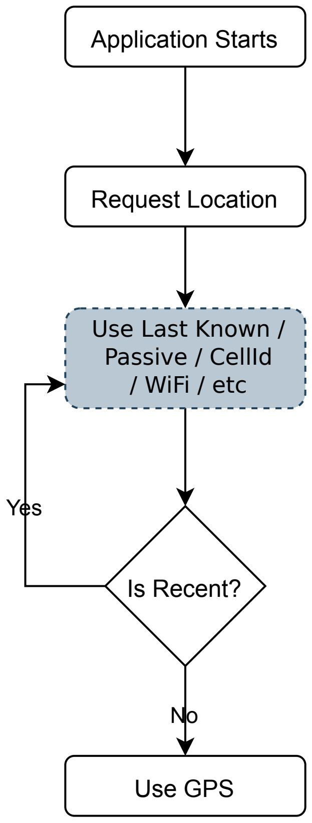


FIGURE 4.9: Checking if energy-efficient alternative is recent (UP05)

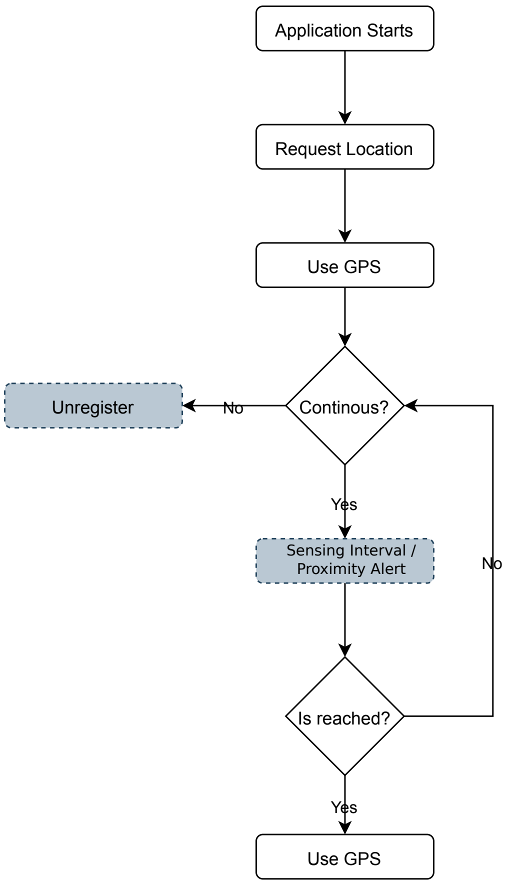


FIGURE 4.10: With sensing interval (UP06)

request. Similarly, this usage pattern also allows the developers to use passive location or Non-GPS methods such as Wi-Fi and CellID positioning to save energy.The location data given by alternative methods would not be useful if the last known location is not recent or meets the location accuracy requirements.

Checking if energy-efficient alternative is recent (UP05)

This usage pattern is an improved version of *UP04*. As shown in Figure 4.9, the developer could check if the alternative methods' location is recent and meets applications accuracy requirements. If the location is recent, this pattern uses the returned location without a new GPS request. Otherwise, it creates a new GPS request to fetch a recent and more accurate location. However, this method has the additional overhead of unregistering and re-registering GPS when the user is on the move.

With sensing interval (UP06)

The usage patterns discussed so far are suitable only in the application when the location is needed once or occasionally. Those usage patterns fail when the application requires location logs periodically. The expert developers have suggested using *sensing interval* to set the time difference between subsequent location requests. Therefore, the application would get location updates when needed. As shown in Figure 4.10, the developers suggest to use GPS initially and check if it is required continuously. Unlike *UP03*, this usage pattern replaces the continuous sensing with periodic sensing to avoid abnormal battery consumption. This usage pattern might not be valid if the application is terminated or in the background.

With alarm manager or scheduler (UP07)

As shown in Figure 4.11, this usage pattern uses Android's Alarm Manager to schedule the location updates. It is an improved version of *UP06*, as it replaces the *sensing interval* by *Alarm manager*. In the case of *sensing interval*, the value will be fixed and may not suitable when the application requires dynamically changing *sensing interval*. Therefore, the *Alarm manager* has been introduced, which automatically triggers the defined procedure when the specified event occurs. This usage pattern addresses the shortcomings of previously discussed usage patterns and provides better results comparatively. Unfortunately, the Android operating system no longer supports Alarms' usage and makes this usage pattern obsolete.

With activity recognition (UP08)

As shown in Figure 4.12, this usage pattern utilizes the results from activity monitoring sensors before making a new GPS request. This usage pattern is an improved version

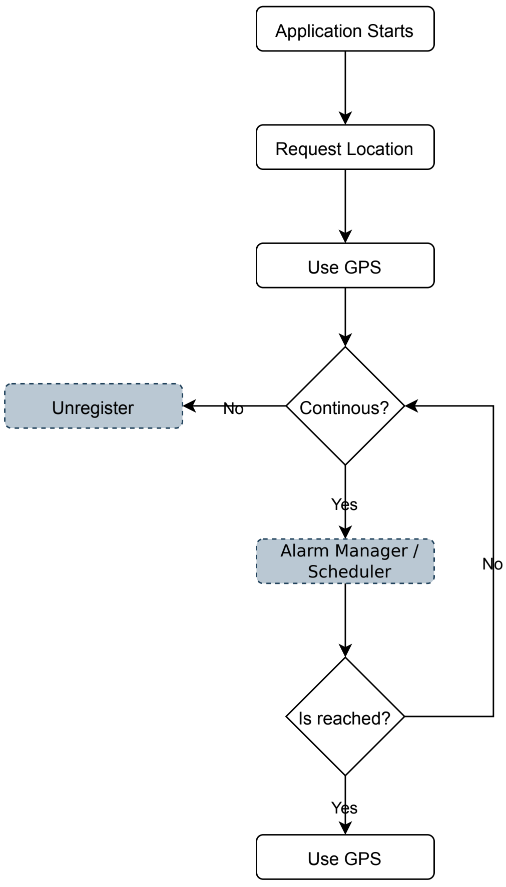


FIGURE 4.11: With alarm manager or scheduler (UP07)

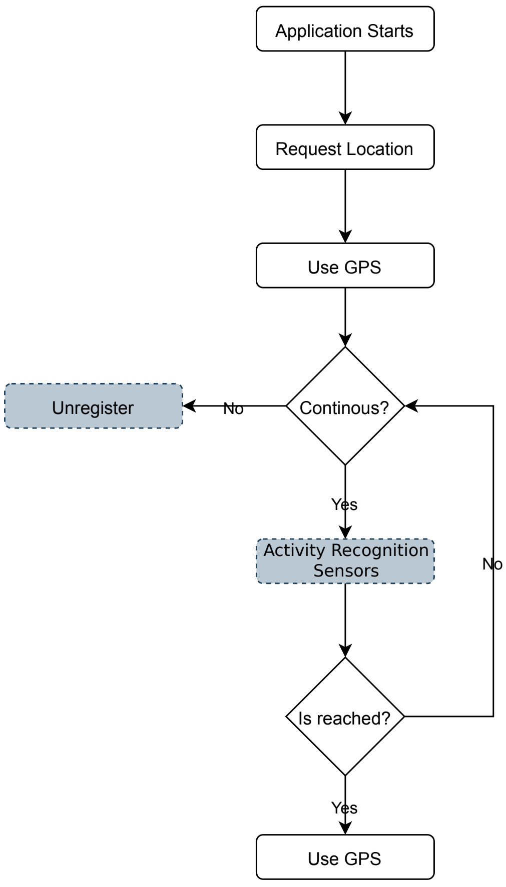


FIGURE 4.12: With activity recognition (UP08)

of *UP07*, and it is widely used in activity monitoring or trajectory tracking applications. Here, sensors like accelerometer, magnetometer, gyroscope, etc. are used as an alternative technique to *Alarm manager*. Here, the new GPS request is executed if the sensors' results imply that the user has moved from the previous location. Otherwise, the developers can hold the new GPS request to avoid unnecessary battery consumption through repeated GPS requests. This technique is energy-efficient and calls the GPS request when the application needs at run-time to satisfy the accuracy requirements.

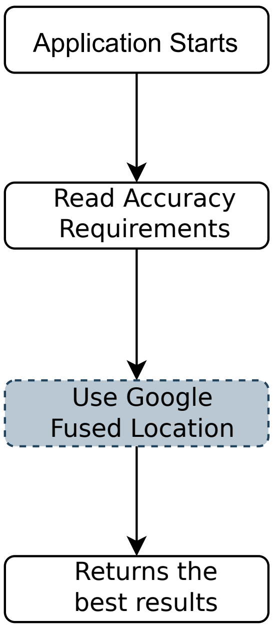


FIGURE 4.13: Use of fused location API (UP09)

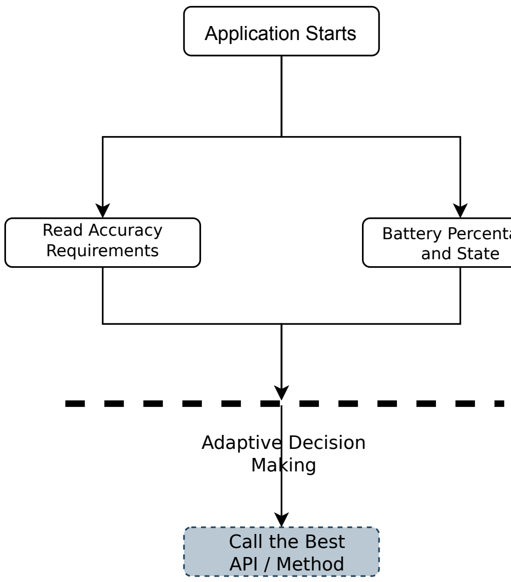

Application Starts
Read Accuracy Requirements
Battery Percentage and State
Adaptive Decision Making
Call the Best API / Method

FIGURE 4.14: Adaptive location-sensing (UP10)

Use of fused location API (UP09)

The usage patterns discussed so far are for Android Native Location API where developers involvement is more for controlling GPS and other location-sensing alternatives. Those usage patterns require more technical knowledge and might not be easy for the new developers to follow as it may involve a sequence of function calls. The location-sensing code might behave differently if the developer misses or misplaces the function calls. Few of the unforeseen, dynamically changing requirements might not be addressed by the previous usage patterns. The source code corresponding to the previous usage patterns are nearly hardcoded and does not consider dynamically changing operating conditions. Therefore, the Google developers have released a new version of location APIs, which abstracts the series of function calls with a single function call. As shown in Figure 4.13, this usage pattern replaces the Fused Location API by Google with several function calls and condition checks while using the Android native location API. This usage pattern is the most suitable version for location-sensing in dynamically changing requirements. The fused location provider selects the best available sensor by considering energy and accuracy at run-time. This usage pattern was suggested mostly by expert developers since 2015. The Fused Location APIs evolve with every Android operating system release and considered more energy-efficient than other location APIs.

Adaptive location-sensing (UP10)

In recent years, expert developers have coupled the location-sensing with the Android battery manager. As shown in Figure 4.14, this usage pattern utilizes the battery percentage and battery charging state values to decide the suitable location-sensing strategies. For instance, the developers can use an energy-efficient location-sensing strategy if the device battery state is discharging, and the battery percentage is low. In this situation, the application might degrade with the accuracy requirements. Similarly, when the device is charging and has a healthy battery level, a more accurate location-sensing strategy could be used to meet the accuracy requirements. This is an improved version of all other usage patterns as it uses the battery manager and accuracy requirements before choosing the location strategy. This usage pattern is more context-aware in nature and produces the best energy-accuracy trade-offs at run-time. This usage pattern gives more flexibility to the developers to balance between energy and accuracy requirements.

These reported usage patterns are the most widely discussed usage patterns by the developers on Stack Overflow. Further, these usage patterns have been analyzed to design the domain-specific modeling language and code generator for energy-saving self-adaptative location-sensing. The recent Stack Overflow posts and official Google developer documentation states that the *Android Native APIs* are energy-hungry and no longer supported. Therefore, the usage patterns related to *Android Native APIs* are not considered in this research work. The domain-specific modeling language and code generator designed to cover the energy-saving aspects of Fused Location API, which is part of *Google Location API*. In addition, the usage pattern, UP10, has been selected as a primary energy-saving pattern for defining the code generator. Further, these patterns will help new developers choose the usage pattern to meet their energy-accuracy requirements.

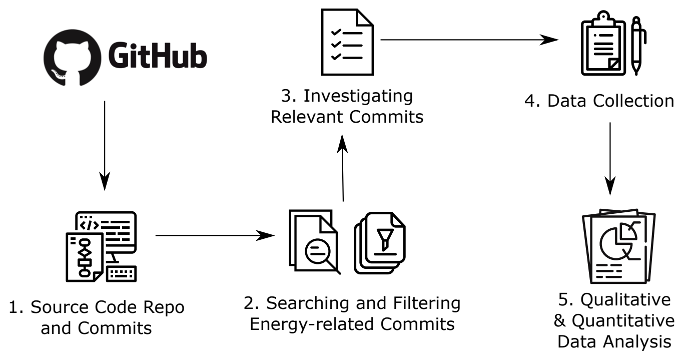

GitHub
1. Source Code Repo and Commits
2. Searching and Filtering Energy-related Commits
3. Investigating Relevant Commits
4. Data Collection
5. Qualitative & Quantitative Data Analysis

FIGURE 4.15: Overall mining protocol of GitHub commits.

4.4 Mining GitHub Commits

This empirical study aims to find out the development efforts needed to improve the application's energy-efficiency after deployment. The open-source location-based applications hosted on the GitHub platforms have been chosen for determining the development efforts. The version history and commit interface to keep track of changes made to GitHub's code base helped find out the source code level development efforts. The results obtained from this empirical study showed the importance of defining a code generator for commonly occurring programming errors towards reducing the development efforts after deployment. This sub-section presents mining protocol, data collection, results, and analysis of the relevant GitHub commits.

4.4.1 Mining protocol

The protocol for mining energy-related commits of open-source location-based Android applications on GitHub is shown in Figure 4.15. The search strings shown in Table 4.5 have been used to search for commits related to *Energy consumption of Location-based Android applications*. The search results returned irrelevant commits also. A filtering process has been carried out to remove the irrelevant search results to ensure that only relevant commits are considered for the final analysis. After choosing the relevant commits, the meta-data is collected, as shown in the data collection table 4.6. Finally, the collected data is taken for qualitative and quantitative analysis to answer the research question. The relevant search strings have been composed to search for energy-related

TABLE 4.5: Search strings and search results of commits search phase

SearchID	Search String	Total Commits	Related Commits
S01	android location battery	3460	3
S02	android gps battery	1997	2
S03	android location energy	2569	0
S04	android gps energy	49	0
S05	android location power	13782	2
S06	android navigation power	370	1
	Total	**22227**	**8**

commits of location-based Android applications. Therefore, the keyword "*android*" is chosen and fixed for search strings. The next important term is the selected application domain. Thus, words such as *location, GPS, navigation* have been added to the search string. The next important dimension of this research is to search for energy-related commits. Therefore, terms such as *battery, energy, power* have been added to the search string. Finally, the following search strings have been formed: *android location battery, android gps battery, android location energy, android gps energy, android location power, and android navigation power*. The Table 4.5 shows the search strings and their corresponding search results. As shown in Table 4.5, the search results returned 22227 commits in total. The irrelevant commits have been ignored immediately without taking it for further in-depth analysis. At the end of the filtering phase, *eight* commits have been found related to location-based Android applications' energy consumption.

4.4.2 Data collection

The relevant commits and their meta-data have been manually read through the interface provided by GitHub. The following data have been collected during the data collection phase:

- **Commit ID** has been assigned to each commits to identify them uniquely.
- **Application Name** has been extracted from each commit to trace it back to the corresponding open-source application.

- **Year** refers to the commit posted year, and it has been useful to ensure that the commits have been posted after 2015.
- **Theme** refers to the energy-saving solutions applied in the relevant commits.
- **Improved Variable** refers to the primary reasons for committing. This includes *energy, accuracy, and energy-accuracy*.
- **Files** refers to the number of files affected by source code changes corresponding to a particular commit.
- **Additions** refers to the number of lines added to the code base as a consequence of committing a change.
- **Deletions** refers to the number of lines deleted from the codebase.

The data such as *Files, Additions, and Deletions* have been primarily collected to determine the development efforts needed to improve energy-efficiency after deploying the application. The collected data for each commit is represented in Table 4.6.

4.4.3 Data analysis

The data analysis is two-fold. First, the qualitative data analysis is performed for the fields *Theme and Improved Variable*. The popular thematic analysis has been used to categorize each commit under *Theme Category* and *Improved Variable Category*. The following themes have been identified for *Theme Category*:

- **Battery Saving Option** theme refer to the commits that are about changing the settings to battery saving from GPS.
- **Changed to Google API** theme has been assigned to the commits that change the codebase from Android location API to Google location API.
- **Unregister Location Updates** theme has been assigned to commits that explicitly unregister the location updates from the existing location-sensing code base.

- **Adjusting Accuracy Requirements** theme refers to the source code level change that changes the accuracy requirements to balance between energy-accuracy requirements.
- **Use Last Known Location** refers to the commits that add an option to use last known location before making new GPs location request.
- **Use Fused Location** refers to the commit that changes the location-sensing API from Android location API to the most recent version of Google Location API, *Fused Lost API*.

For the *Improved Variable* field, the following categories have been found through thematic analysis:

- **Energy**: The commits have been classified under this category if it is explicitly concerned about reducing the battery usage by ignoring accuracy.
- **Accuracy**: The commits have been classified under this category if it is explicitly concerned about increasing the accuracy by ignoring battery-draining behavior.
- **Energy-Accuracy**: The commits have been classified under this category if it is about balancing between energy and accuracy-related requirements.

The relevant themes have been assigned to each commit for *Theme* and *Improved Variable* to check the developer's preferences and applied energy-saving solutions in the open-source Android applications.

4.4.4 Answer to RQ6: Development efforts

The second data analysis phase has been conducted using the quantitative analysis on the field *Files, Additions, Deletions*. The numerical values returned for each field have been considered to calculate the development efforts numerically. The analysis result is plotted in Figure 4.16.

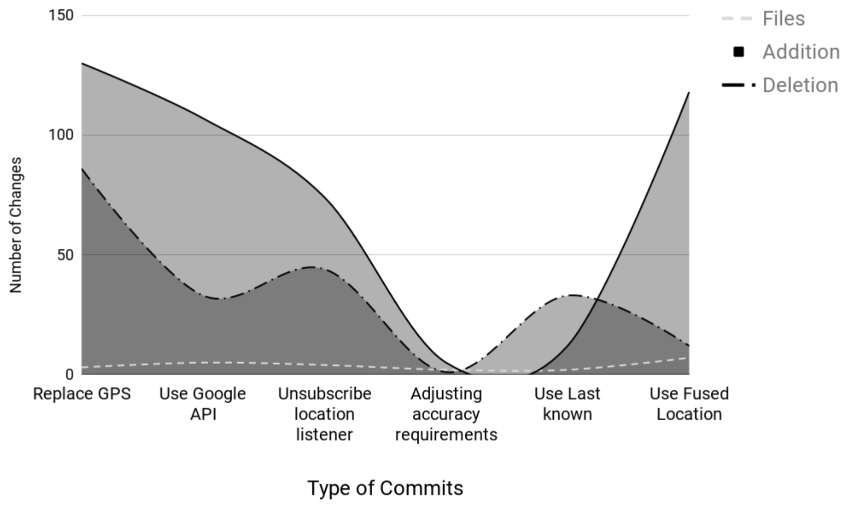


FIGURE 4.16: Development efforts needed to make changes in source code.

How much development effort does it take to improve energy-efficiency after deployment? As shown in Table 4.6, the following categories have been found from the collected data: *Battery saving option, Changing to Google API, Unregistering Location Updates, Adjusting accuracy requirements, Using Last known location, and Using Fused Location*. All these solutions are widely accepted solutions by the developers from the results of *Qualitative data analysis of Stack Overflow Posts*. Therefore, these solutions and corresponding commits are suitable for calculating the development efforts needed for open-source applications. This research question aims to find out how much development efforts are needed to update the source code for each category of commits. The analysis results have been plotted in area plot, as shown in Figure 4.16. As shown in Figure 4.16, the data such as changed files, added, or deleted lines from each commit has been collected. The analysis shows that *changing GPS to battery saving* needs more development efforts than other types of commits. Similarly, *Changing from native to Google API* also needs more development efforts as it has been suggested in 3 commits and involves changes in 5 files, addition of 107 lines, deletion of 33 lines on an average. As shown in Table 4.6, *using Fused location* requires changes from 7 files. Other types of commits need moderate development efforts as it involves less number of files. Overall, the energy-saving efforts are more in terms of source code changes. Hence it is recommended for developers to use energy-efficient practices while developing the application itself.

TABLE 4.6: Data collection results of mining GitHub commits

Commit ID	Application Name	Year	Theme	Improved Variable	Files	Additions	Deletions
C1	android-packages-apps-SetupWizard	2015	Replace GPS option with Battery Saving option	Energy	3	130	86
C2	Project FollowWe	2016	Changed to Google API	Energy	7	154	37
C3	StorkApp	2017	Changed to Google API	Energy	2	60	29
C4	mapsme-omim	2017	Unregister location updates	Energy	4	96	75
C5	openlocate-android	2017	Adjusting accuracy requirements	Energy-Accuracy	2	5	1
C6	bo-android	2017	Use Last known	Energy	2	12	33
C7	LocationStuff	2018	Use Fused Location	Energy	7	118	12
C8	GeoNotes	2018	Unregister location updates	Energy	3	51	13

4.5 Implications

This section presents the implications of the empirical studies results. These implications helped in positioning the ideas and designing the case studies presented in the upcoming chapters.

Implications from Controlled Experiment : The important finding from the controlled experiments is that the location-sensing and GPS usage are the second major contributors to overall energy consumption. Therefore, reducing the energy consumption of location-sensing would significantly reduce the energy consumption of location-based Android application. Hence, selecting location-based Android applications as application domain is a valid research direction for showing the efficacy of energy-aware modeling framework and code generator. The answer to RQ2 shows that the use of *inertial sensors* and GPS location-sensing reduces energy consumption. At the same time, the selection of suitable sensors and the frequency of their usage play a significant role in overall energy consumption. As shown in the results, the inappropriate use of inertial sensors sometimes could lead to abnormal battery drain like Open Street Map Location API. This finding helped in choosing an energy-efficient strategy for using sensors in the code generator. From the answer to RQ3, it has been found that *Google Location API* is energy-efficient than *Open Street Map API*, as it uses suitable sensors only when needed instead of making it active all the time. Therefore, the code generator is developed to generate location-sensing code for *Google Location API* calls.

Implications from Stack Overflow : The results of analyzing Stack Overflow post show that the self-adaptive location-sensing, coupled with battery-awareness, is a promising strategy for battery-powered devices. Therefore, the self-adaptive strategy has been adopted in the energy-aware modeling framework and code generator. In addition, balancing between energy and accuracy requirements is an essential criterion for the application's success. Hence, the subject applications used in the case studies have been designed to balance energy and accuracy requirements depending on the operating conditions.

Implications from Mining GitHub commits : The results from mining GitHub commits show that cost of improving energy-efficiency is high after deployment in terms of several modified files and function calls. Therefore, considering energy-related requirements in the early stages would help developers to write energy-aware code to reduce the development cost after deployment. In addition, as reported in answer to RQ6, the energy-saving solutions found in the GitHub repositories have been repeating and following a common usage pattern. Therefore, having a code generator that generates code from design-time models would reduce the developer's efforts during software construction.

4.6 Summary

The contributions of this objective can be summarized as follows:

- The controlled experiment has been carried out on three popular location-based Android applications to show the energy consumption data and the cause-effect relationship between energy consumption and location sensors.
- Qualitative data analysis has been performed on relevant Stack Overflow posts to determine the most widely suggested energy-saving solutions and energy-saving API usage patterns to balance energy-accuracy requirements. The self-adaptive location-sensing pattern is promising, and the same has been used in the remaining part of this thesis. In addition, the discussed energy-saving usage patterns will be helpful for software architects to make suitable architectural design decisions.
- Energy-related commits on GitHub sources have been performed on open-source location-based Android applications to determine the development efforts needed to fix energy-related issues after deployment. The study results show that energy improvement after deployment is costly in modified files and source code.

Overall, the results presented in this objective satisfactorily answered the research questions. The presented results and data will be a clear conceptual starting point for designing an energy-aware modeling framework and code generator.

Chapter 5

ENERGY-AWARE MODELING FRAMEWORK

This chapter presents the energy-aware modeling framework for domain analysts to conduct the domain analysis of applications under development. The modeling framework includes a context and feature classification method along with planning energy-saving self-adaptations. The presented modeling framework's primary goal is to derive Valid Triggering Situations (VTS) and corresponding Energy-aware Application Configurations (EAC). Further, the framework suggests a way to plan energy-saving self-adaptation strategies based on Event-Condition-Action (ECA) rules. Overall, this framework adopts feature models to carry out the domain analysis considering energy-aware requirements. The extended rules and categories for energy-aware context models and energy-aware feature models are described in Section 5.2. As part of empirical evidence, the efficacy of the proposed approach has been demonstrated through a case study on the family of map navigation applications. In Section 5.3, the case study based validation and discussions are presented. The case study results show that the presented modeling framework can be effectively used to generate re-usable design-time artifacts and energy-saving adaptation plans for a family of software systems.

5.1 Basics of Feature Modeling

Feature models are originally introduced by (Kang et al., 1990) in the Feature-Oriented Domain Analysis (FODA) report (1990). Since then, feature models are considered to be an important information model widely used in SPL engineering. A *feature model* is a formal way of modeling the commonalities and variabilities of SPL (Kang et al., 1998), Batory:2005:FMG:2162231.2162236. A feature model represents the set of features, relationships, and constraints among them. A *feature* can be formally defined as user-visible property (Kang et al., 1990) of the system. Feature models can be graphically represented through a tree-like diagram called *feature diagram* as shown in Figure 5.1. It represents the relationships among features through parent-child relationships. The following relationships are part of basic feature models (Benavides et al., 2010):

- *Mandatory* relationship signifies that the child feature must be included in all the product configurations, if its parent feature is included. For instance, from Figure 5.1, the child feature *Calls* will be added to all product configurations as it has *mandatory* relationship with its parent feature *Mobile*.
- *Optional* relationship signifies that the child feature presence is optional even if its parent feature is added into the product configuration. Hence, the child feature with *optional* relationship may or may not be included in all the products. For instance, from Figure 5.1, the optional child features *GPS* and *Media* may be added or discarded in all the product configurations of *Mobile*.
- *Alternative* relationship signifies that exactly one child feature from a set of child features can be added to the product configuration if its parent is part of the product. For instance, from Figure 5.1, any one of *Basic, Color, High Resolution* can be selected when its parent *Screen* is part of the product.
- *OR* relationship signifies that one or more child features can be included into the product configurations along with its parent feature. For instance, from Figure 5.1, either of *Camera*, or *MP3*, or both can be added to the product configuration whenever the *Media* feature gets selected.

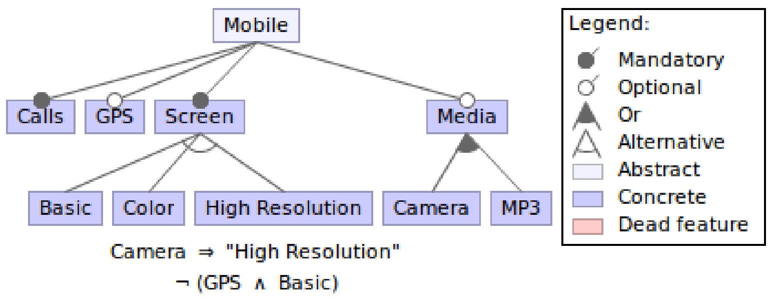


FIGURE 5.1: Sample feature model. adapted from (Benavides et al., 2010)

These basic relationships between parent and child features are sufficient to capture all interdependencies among all features. Apart from these basic relationships, the cross-tree constraints also must be specified along with the feature diagrams. Following are the two basic constraints to capture the cross-tree constraints of a feature model:

- *Requires* constraint signifies that the inclusion of particular feature requires inclusion of other related features also. For instance, from Figure 5.1, *requires* constraint is established between *Camera* and *High Resolution* as *Camera=>"High Resolution"*. This signifies that the child feature *Camera* under *Media* requires the presence of *High Resolution* under *Screen* in the same product configuration.

- *Excludes* constraint signifies that the inclusion of a particular feature must discard or exclude the other irrelevant features from the product configuration. For instance, from Figure 5.1, the *excludes* constraint is established between *GPS* and *Basic* as $\neg(GPS \wedge Basic)$. This signifies that the child feature *GPS* under *Mobile* and child feature *Basic* under *Screen* cannot be selected together in the same product configurations. That is feature *GPS* is incompatible with *Basic*.

The energy-aware modeling framework adopts this feature model to specify energy-related requirements of the software. For this purpose, the semantics of the relationships between parent and child features have been modified, considering each feature's energy impact. Specifically, the semantics of relationships varies based on the energy-aware categories introduced in this framework. The remaining part of this chapter presents detailed information about adopting feature models for the energy-aware modeling framework.

5.2 Energy-aware Modeling Framework

The energy-aware modeling framework aims at introducing energy-awareness into the reference requirements of the application domain under consideration. This framework aids domain analyst to identify energy-hungry requirements and to introduce its energy-efficient alternatives. As illustrated in Figure 5.2, the framework produces the following artifacts: *energy-aware context model, energy-aware feature model, and energy-saving adaptation plan*. The framework is suitable for the software that applies self-adaptive behavior for energy-saving by re-configuring the application to its energy-efficient version at run-time. Typically, the reconfiguration occurs when a critical battery situation (valid triggering situation) is identified at run-time. The *energy-aware context model* could be used to specify the dynamic context changes and corresponding battery critical situations to trigger the energy-saving adaptation. On the other hand, the *energy-aware feature model* could be used to derive different application variants for energy-saving self-adaptation that could be re-configured at run-time. The energy-aware feature models consist of energy-efficient alternatives added to the energy-hungry requirements of a specific application domain. As illustrated in Figure 5.2, the outcome of the energy-aware context model and energy-aware feature model can be combined to generate the energy-saving adaptation plans. The modeling framework's core idea is to derive valid triggering situations (VTS) from the energy-aware context model and the energy-aware application configurations (EAC) from the energy-aware feature model. Finally, the VTS and EAC can be combined using the ECA based rules to plan the adaptation at an early stage of software development. In this section, the conceptual idea behind the energy-aware modeling framework is elaborated.

5.2.1 Energy-aware context modeling

An *energy-aware context modeling* is the process of specifying context information of smartphones that positively or negatively impact the energy-efficiency. In the smartphone domain, the following can be considered important context information: *percentage of available battery, battery charging state, the status of smartphone resources, applications state, user's physical context, and user's preferences*. It is essential to capture the energy impact of battery-related context information in the early stages of

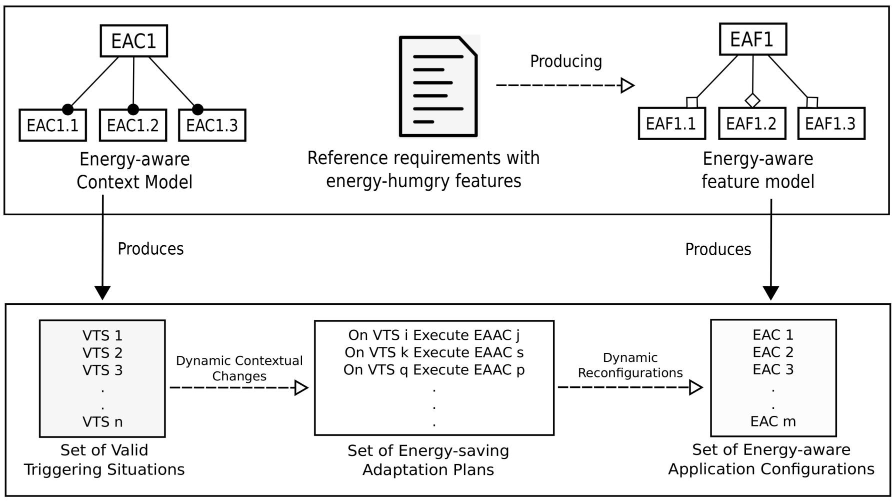

EAC1
EAC1.1
EAC1.2
EAC1.3
Energy-aware Context Model
Reference requirements with energy-humgry features
Producing
EAF1
EAF1.1
EAF1.2
EAF1.3
Energy-aware feature model
Produces
Produces
VTS 1
VTS 2
VTS 3
VTS n
Set of Valid Triggering Situations
Dynamic Contextual Changes
On VTS i Execute EAAC j
On VTS k Execute EAAC s
On VTS q Execute EAAC p
Set of Energy-saving Adaptation Plans
Dynamic Reconfigurations
EAC 1
EAC 2
EAC 3
EAC m
Set of Energy-aware Application Configurations

FIGURE 5.2: Overview of energy-aware modeling framework

software development to make the development team aware of different battery critical situations. A feature diagram is adopted to create an energy-aware context model, which includes *context information, context value, energy-aware context label, energy-aware context relationships, context constraints, and valid triggering situations*. The energy-aware context model can be created by domain analysts aware of smartphone resources, application requirements, and it's battery consumption behavior. An Energy-aware Context Model (EACM) can be defined as follows:

$$\mathtt{EACM} = \{\mathtt{CR}, \mathtt{CI}, \mathtt{CV}, \mathtt{ECL}, \mathtt{ECR}, \mathtt{CC}, \mathtt{VTS}\} \tag{5.1}$$

Where,

- $\mathtt{CR}$ represents the Context Root
- $\mathtt{CI}$ represents the Context Information
- $\mathtt{CV}$ represents the Context Values
- $\mathtt{ECL}$ represents the Energy-aware Context Labels
- $\mathtt{ECR}$ represents the Energy-aware Context Relationships
- $\mathtt{CC}$ represents the Context Constraints

- `VTS` represents the Valid Triggering Situations

The traditional feature model proposed by Kang et al. (1990) with a suitable extension to generate the energy-aware context model has been adopted.

Definition 5.1 (Context Information (CI))**.** A Context Information (CI) is a non-root and non-atomic entry of an energy-aware context model that must be specified with it's allowed context values. It can be defined as follows:

$$\mathbb{C} = \{\mathbf{C}_1,\ \mathbf{C}_2,\ \mathbf{C}_3,\ ...,\ \mathbf{C}_p\} \tag{5.2}$$

Where,

- $\mathbb{C}$ refers to Context Root (CR)
- $\mathbf{C}_{1...p}$, refers to set of all identified context information

Definition 5.2 (Context Value (CV))**.** A Context Value (CV) is an atomic and leaf entry of the energy-aware context model, which is considered to be the children of context information. It can be defined as follows:

$$\mathbf{C}_i = \{c_{i \cdot 1},\ c_{i \cdot 2},\ c_{i \cdot 3},\ ...\ c_{p \cdot q}\} \tag{5.3}$$

Where,

- $\mathbf{C}_i$ refers to Context Information
- $c_{i \cdot q}$ refers to context values of $\mathbf{C}_i$
- i is an integer and ranges from $1...p$
- q is an integer and may contain any finite value based on the number of context values

In this framework, domain analysts are suggested to assess the energy impact of each context information to assign suitable energy-aware labels. More information on energy-aware context labels are given below:

Definition 5.3 (Energy-aware Context Label (ECL))**.** An Energy-aware Context Label (ECL) is a tag assigned to each context information to express its energy impact on the system. For this purpose, the following labels are provided in this framework:

- **Primary-influencing Context** (PC) refers to the context information that directly relates to the system's energy consumption. The *primary influencing context* must be selected in a way such that a change in its value influences the other context information. In addition, the context values of *primary influencing context* have a more significant impact on the selection of application features.
- **Secondary-influencing Context** (SC) refers to the context information not directly associated with the system's energy consumption. However, this context information has a moderate impact on selecting energy-hungry or energy-friendly features of the system.
- **User-influencing Context** (UC) refers to the context information controlled or selected by the user at run-time. This context information and its context values have a direct or indirect impact on the system's energy-efficiency.

A domain expert can label all the possible context information with suitable energy-aware context labels. This framework extends the relationships introduced in the basic feature models to add more value to the context-aware labels.

Definition 5.4 (Energy-aware Context Relationship (ECR))**.** An Energy-aware Context Relationship (ECR) is a relationship between the context root, context information, and context value of the energy-aware context model. The following semantic level modifications have been applied to the relationships proposed in FODA (Kang et al., 1990):

- **Mandatory** relationships must be established with the *primary-influencing context* information and *context root*. In addition, the *secondary-influencing context* information also can be considered for mandatory relationships if its presence is essential for better energy-savings.

- **Optional** relationships can be established by default between the *context root* and the *secondary-influencing context*.
- **Alternative** relationships must be established for all the allowed context values.
- **OR** relationship cannot be established between any context values as only one context value can be sensed at a time.

Besides specifying energy-aware context information, context values, and relationships, it is important to specify the context constraints between context values.

Definition 5.5. A Context Constraint (CC) is cross-tree constraints established between two different context information or it's context values to represent its inter-dependency or mutual exclusiveness. The *requires and excludes* constraints of FODA (Kang et al., 1990) has been extended to specify the context constraints.

The following semantic klevel extensions has been provided to *requires and exclude* constraints of FODA for context constraints:

- **Requires** could be established between the context values of distinct context information. Specifically, it is used between the context values of *primary-influencing* and *secondary-influencing* context information to express the inter-dependencies.
- **Exclude** could be established between the context values of conflicting context information. For instance, the exclude constraint could be established between the context values of *user-influencing* context with other context values as it has a higher preference.

Definition 5.6. Valid Triggering Situations (VTS) refers to the set of valid configurations generated from the energy-aware context model by applying the *context constraints* and *energy-aware context relationships*. These situations represent the state of the user and smartphone in any operating condition. The decision on selecting an appropriate set of features for each VTS is defined in the adaptation planning phase.

5.2.2 Energy-aware feature modeling

An *energy-aware feature modeling* is the process of specifying the reference requirements along with its energy-impact and energy-efficient alternatives. The basic feature model has been adopted to introduce energy impact and energy-efficient alternatives. In the energy-aware modeling framework, the energy-aware feature model consists of *root feature, atomic features, non-atomic features, energy-aware feature labels, energy-aware feature relationships, feature constraints, and energy-aware application configurations*. The domain analyst can assess each feature's energy impact requested by the stakeholders to label them with energy-aware context labels. An Energy-aware Feature Model (EAFM) can be defined as follows:

$$\mathtt{EAFM} = \{\mathtt{RF}, \mathtt{NF}, \mathtt{AF}, \mathtt{EFL}, \mathtt{EFR}, \mathtt{FC}, \mathtt{EAC}\} \tag{5.4}$$

Where,

- $\mathtt{RF}$ represents the Root Feature
- $\mathtt{NF}$ represents the Non-atomic Feature
- $\mathtt{AF}$ represents the Atomic Feature
- $\mathtt{EFL}$ represents the Energy-aware Feature Labels
- $\mathtt{EFR}$ represents the Energy-aware Feature Relationships
- $\mathtt{FC}$ represents the Feature Constraints
- $\mathtt{EAC}$ represents the Energy-aware Application Configurations

Definition 5.7 (Root Feature (RF))**.** *Root Feature (RF)* refers to an abstract feature that describes the whole energy-aware feature model.

Definition 5.8 (Non-atomic Features (NF))**.** Non-atomic Features (NF) refers to the abstract features of the energy-aware feature model, which are considered to be a parent or grouping feature to represent the energy consumption. The identified non-atomic features can be specified, as shown below:

$$\mathbb{F} = \{\mathbf{F}_1,\ \mathbf{F}_2,\ \mathbf{F}_3,\ ...,\ \mathbf{F}_n\} \tag{5.5}$$

Where,

- $\mathbb{F}$ refers to Root Feature
- $\mathbf{F}_{1...n}$, refers to set of all identified non-atomic features and its sub-features

Definition 5.9 (Atomic Features (AF))**.** Atomic Features (AF) refers to the concrete child or leaf feature of the *energy-aware feature model.* The identified atomic features can be specified as follows:

$$\mathbf{F}_i = \{f_{i\cdot 1},\ f_{i\cdot 2},\ f_{i\cdot 3},\ \cdots f_{n\cdot m}\} \tag{5.6}$$

Where,

- i is an integer and ranges from $1...n$
- m is an integer and may contain any finite value based on the number of sub features

The domain analysts are suggested to assess the energy impact of each *atomic and non-atomic* features to tag them with the suitable energy-aware labels proposed in this framework.

Definition 5.10. *Energy-aware Feature Labels (EFL)* refers to the tag that specifies each identified feature's energy impact. For this purpose, the following energy-aware feature labels are provided in this framework:

- **Energy-friendly Feature** (EF) refers to the features that consume less energy for its operation under any operating conditions. The features under this label must be considered as default features while generating the energy-aware feature model configurations. This label can be applied to *non-atomic features* if all its *atomic features* are energy-friendly.

- **Energy-hungry Feature** (EH) refers to the features that consume relatively more energy for its operation under different operating conditions. By default, the inclusion of these energy-hungry features into the energy-aware feature model configurations can be optional for energy-savings. If this feature's presence is inevitable, a domain analyst can consider to use its energy-efficient alternatives for better energy-savings.

- **Energy-efficient Alternatives** (EA) refers to the atomic sub-features of the *energy-hungry features*. Each sub-feature will be decided in such a way that it consumes less or more energy than its alternatives. Only one alternative feature can be selected at the same time for better energy-savings.

Definition 5.11. *Energy-aware Feature Relationships (EFR)* refers to the relationships between the root feature, non-atomic features, and atomic features of the energy-aware feature model. In this framework, the following semantic level modifications to the existing feature model relationships are provided to support energy-awareness:

- **Mandatory** relationship could be established by default between *root feature* and *non-atomic feature* if it is labeled with *Energy-friendly Features*. Also, a mandatory relationship can be established between *root feature* and energy-hungry *non-atomic feature*, if its presence is inevitable. For each mandatory energy-hungry *non-atomic features*, the *energy-efficient alternatives* must be specified to introduce energy-awareness. Otherwise, these features will not be considered for mandatory relationships.

- **Optional** relationship could be established by default between the energy-hungry *non-atomic feature* and *root feature* in order to save energy in the limited resource situations. Specifying energy-efficient alternatives for the features under this relationship is also optional.

- **Alternative** relationship could be established between the child features of the *energy-hungry non-atomic features*. As per the basic definition of *alternatives*, a maximum of one feature can be selected among the energy-efficient alternatives.

- **OR** relationship could be established between the *optional energy-friendly features* and its sub-features. Strictly, the OR relationship cannot be established between the *energy-hungry features* and its alternatives.

A domain analyst can apply the relationships mentioned above to the labeled energy-aware feature model to ensure that the feature model generates multiple energy-aware configurations of the system.

Definition 5.12 (Feature Constraints (FC))**.** *Feature Constraints (FC)* refers to the constraints established between the features in the energy-aware feature model to express the cross-tree constraints. Here, the constraints *Excludes* and *Requires* are used in basic feature models.

Finally, the energy-aware labels, relationships, and constraints can be applied to generate more suitable application variants that can be used after the occurrence of any valid triggering situations.

Definition 5.13 (Energy-aware Application Configurations (EAC))**.** *Energy-aware Application Configurations (EAC)* refers to the valid configurations generated from the energy-aware feature model. A list of energy-aware application configurations could be derived by applying energy-aware labels, relationships, and constraints. An EAC is valid if it conforms to the EFR and FC. An EAC is complete if it is atomic and represents a valid run-time application configuration that can be executed for the system's energy-savings.

The *energy-aware application configurations* generated from *energy-aware feature model* represents the application variants that consume different energy for its operation. In other words, few configurations consume more energy while the other configurations consume less or moderate energy. Meanwhile, the *valid triggering situations* generated from the *energy-aware context model* can be mapped with the appropriate *energy-aware application configuration* to reduce energy consumption.

5.2.3 Energy-saving adaptation planning

After modeling energy-aware context information and features, the next step is to plan the energy-saving adaptation strategies. Adaptation policies decide when and how to change the software behavior. The basic idea here is to make the adaptation plans towards energy-savings.

Therefore, the following terms are defined:

Definition 5.14. An Energy-saving Adaptation (ESA) refers to the adaptations that intend to reduce the software's energy consumption by selecting the appropriate application configurations at run-time.

The energy-saving adaptation plans follow the Event Condition Action (ECA) rules pattern. Here, *event* is the dynamic contextual changes (related to battery-critical situations), *condition* is to check the occurrence of valid entry in the energy-saving adaptation plan, *action* is the suggested energy-saving adaptation. This adaptation may be expressed in general as follows:

$$\textbf{On } VTS_{k...r} \textbf{ Execute } EAC_{l...s} \tag{5.7}$$

Where,

- k is an integer and ranges from $1...r$
- r is a total number of valid triggering situations
- l is an integer ranges from $1...s$
- s is a total number of energy-aware application configurations

The energy-saving adaptation plans specified by the domain analyst might give significant clarity to the developers about energy-saving adaptations at run-time. In summary, the energy-aware modeling framework presented in this section is targeted for the early stages of software engineering. The presented modeling framework is a generic modeling framework that can be applied to any energy-aware self-adaptive software, irrespective of the platform. However, in this thesis, the aim is to apply the energy-aware modeling framework to the energy-aware self-adaptive smartphone applications.

5.3 Validation with a Case Study

The energy-aware modeling framework is validated based on the guidelines by Easterbrook et al. (2008), Runeson and Höst (2009). Here, *family of map navigations* applications are selected as a subject application to show the efficacy of the presented modeling framework. A case study based empirical method has been selected to validate the modeling framework as applying the energy-aware modeling framework on the map navigation application is an exploratory problem. Therefore, the case study has been designed to answer the following research questions (RQs):

- **RQ1**: How to introduce the energy-related requirements at design-time for the family of map navigation applications?
- **RQ2**: How to design an energy-saving self-adaptation plan for the family of map navigation applications?
- **RQ3**: What is the impact of an energy-aware modeling framework on designing a family of map navigation applications?

A qualitative validation of the energy-aware modeling framework is presented by answering these research questions at the end of the case study. This section presents feature models for the family of map navigation application, energy-aware context model, feature model, adaptation model for dynamic energy-savings.

5.3.1 An example scenario

The map navigation application aims to assist the users in finding directions for *walking, cycling, and driving*. Additionally, the map also shows on the go directions to the destination from any source (most preferably the user's current location). Key features of the example application are described below:

- **Location Sensing**: The application must automatically fetch the user's current location. High accuracy location sources such as GPS must be used as a navigation application requires a more accurate user location.

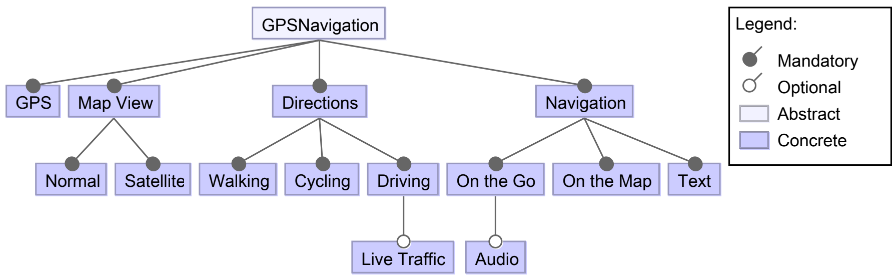


FIGURE 5.3: A basic feature model of map navigation application

- **Map Display**: The default business functionality of the navigation application is to display a map on the user's device. The user interface must include the normal view and satellite view of the map.
- **Directions**: The navigation application must show the directions to reach the destination from the user's current location. The directions must be separately displayed for *walking, cycling, and driving*. Sometimes, the live traffic of the route must be displayed to the user for driving directions.
- **Navigation**: The important requirement of this navigation application is to show the turn-by-turn directions to the user to reach the destination. These instructions can be displayed while the user is moving with audio as an optional feature. However, the application should show the directions on the map or as text-based instructions by default.
- **Energy Saving**: One of the essential features mentioned explicitly is Energy Saving. This application involves many energy-hungry operations such as GPS, audio instructions, and satellite view; it is important to consider energy-saving as an essential requirement. Especially on mobile devices, unnecessary usage of energy-hungry operations may drain the device battery quickly. Therefore, there should be a mechanism to extend the battery life as and when required.
- **User preference**: The application must provide users with the ability to specify their preferences, such as energy-saving or more accurate location, etc.

With the given requirements, a basic feature diagram of the family of map navigation application is illustrated in Figure 5.3. The feature diagram of the subject application scenario consists of the features *GPS, Map View, Directions, and Navigation*. The feature diagram of the map navigation application can be formally represented, as shown in Equation 5.8. All features have *Mandatory* relationship; therefore, these features will be added into all application configurations by default.

$$\mathbf{F}_{\mathbf{GPSNavigation}} = \{\mathtt{F}_{\mathtt{GPS}}, \mathtt{F}_{\mathtt{MapView}}, \mathtt{F}_{\mathtt{Directions}}, \mathtt{F}_{\mathtt{Navigation}}\} \tag{5.8}$$

The map navigation application's sub-features can be represented, as shown in Equation 5.9. The *Map View* feature can be further divided into *Normal and Satellite* views, which show different viewing options to the user. These two features will be added by default to the application configurations as it has *Mandatory* relationship. As specified in the requirements, the *Directions* feature can be further divided into *Walking, Cycling, Driving* to provide different route options for the users. As these features belong to the application's core business functionality, they are designated as *Mandatory* features of the application. In addition, the application also shows live traffic as an *optional* feature for driving directions.

$$\begin{aligned}
\mathtt{F}_{\mathtt{MapView}} &= \{\mathtt{Normal}, \mathtt{Satellite}\}, \\
\mathtt{F}_{\mathtt{Directions}} &= \{\mathtt{Walking}, \mathtt{Cycling}, \mathtt{Driving}\}, \\
\mathtt{F}_{\mathtt{Driving}} &= \{\mathtt{LiveTraffic}\}, \\
\mathtt{F}_{\mathtt{Navigation}} &= \{\mathtt{OntheGo}, \mathtt{OntheMap}, \mathtt{Text}\}, \\
\mathtt{F}_{\mathtt{OntheGo}} &= \{\mathtt{Audio}\}
\end{aligned} \tag{5.9}$$

The next critical feature of the subject application is *Navigation*. It has *Mandatory* relationship with immediate sub-features *On the Go, On the Map, Text*. Further, *On the Go* feature has a *Optional* relationship with its sub-feature *Audio* to provide the voice instructions. The feature model relationship rules and cross-tree constraints are applied on the feature model to derive Application Configurations (AC). Table 5.1 shows *four* different application configurations that have been derived from the basic feature model of the map navigation application (refer Figure 5.3).

TABLE 5.1: Application Configurations (AC) for deriving family of map navigation applications from basic feature diagram

Features	AC1	AC2	AC3	AC4
GPS	×	×	×	×
Map View	×	×	×	×
Normal	×	×	×	×
Satellite	×	×	×	×
Walking	×	×	×	×
Cycling	×	×	×	×
Driving	×	×	×	×
Live Traffic		×		×
Navigation	×	×	×	×
On the Go	×	×	×	×
Audio			×	×
On the Map	×	×	×	×
Text	×	×	×	×

The next step is to analyze the impact of derived ACs on the applications' overall energy consumption. As shown in Table 5.1, *AC1* does not contain energy-hungry operations *Live Traffic and Audio*. Compared to other AC configurations *AC1* may consume less energy. The *AC2* configuration uses *Live Traffic* and *AC3* uses *Audio* as extra features and will result in more energy consumption when compared to *AC1*. Finally, the *AC4* will result in high energy consumption as it includes all energy-hungry features. Overall, *AC4* might consume more energy and *AC1* might consume comparatively less energy. Unfortunately, all the generated ACs uses *GPS and Satellite view* as a default feature and no option of disabling it at run-time. Therefore, these generated ACs failed to provide significant energy-savings as they do not provide much flexibility to bind energy-efficient version at the run-time when the mobile is running with low battery. In order to provide more flexibility for better energy-savings, it is important to introduce context-awareness and the corresponding self-adaptivity in the feature diagram depicted in Figure 5.3.

5.3.2 Energy-aware context model

The energy-aware modeling framework suggests the specification of context information that influences the map navigation application's energy consumption. As the targeted platform is a smartphone, battery availability becomes more crucial, and context information becomes a critical factor in deciding energy-saving adaptations. In this sub-section, the selection of suitable context information and its influence on the map navigation application is considered. The list of identified context information is given below:

- **Battery Percentage**: The amount of remaining battery plays a major role in deciding energy-saving adaptations. For example, when the device runs out of battery, the adaptation engine or application itself should disable all the energy-hungry features. This context information can influence all the features of the navigation application.
- **Battery State**: Strict energy-saving rules may not be needed when the device battery is low and is connected with a charger. The application can bind the features which can balance both accuracy and energy-related requirements. Therefore, monitoring the state of battery charging can also be considered as another essential context information.
- **Network Speed**: Like battery-related context, the network speed can also help the application be more energy-efficient. For example, when the network runs on the slow *2G* network, running features like *satellite view or audio* may lead to more processing time, which in-turn drain the battery at a faster pace. Therefore, capturing network speed can also be considered as important context information.
- **User Preferences**: Most of the applications might include user preferences to give more user satisfying services. Sometimes, user preferences may lead to abnormal energy consumption when the battery level is low. The adaptation engine must identify such information at run-time, and the preferences should be changed for energy-savings.

With this preliminary analysis, the energy-aware context model has been modelled as depicted in Figure 5.4. The energy-aware context model contains context information

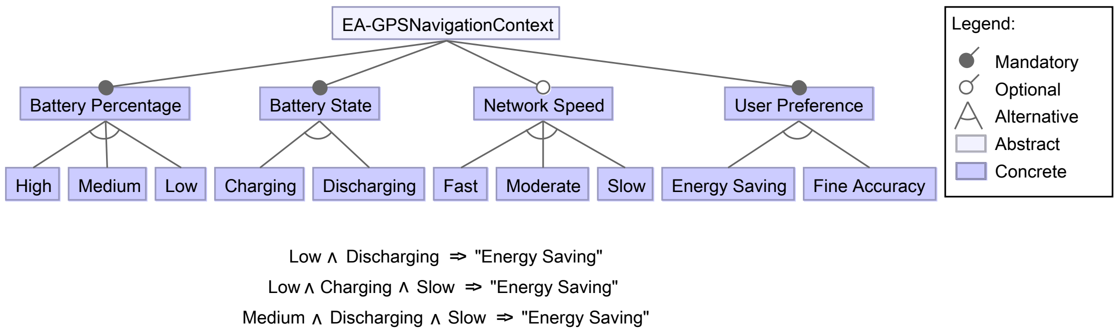


FIGURE 5.4: Energy-aware context model of map navigation application

Battery Percentage, Battery State, Network Speed, and User Preference. The energy-aware context model can be formally represented, as shown in Equation 5.10:

$$\mathbf{C_{GPSNContext}} = \{\mathtt{C_{BatteryPercentage}}, \mathtt{C_{BatteryState}}, \mathtt{C_{NetworkSpeed}}, \mathtt{C_{UserPreference}}\} \tag{5.10}$$

The context values of the different context information is represented formally in Equation 5.11.

$$\begin{aligned} \mathtt{C_{BatteryPercentage}} &= \{\mathtt{High}, \mathtt{Medium}, \mathtt{Low}\}, \\ \mathtt{C_{BatteryState}} &= \{\mathtt{Charging}, \mathtt{Discharging}\}, \\ \mathtt{C_{NetworkSpeed}} &= \{\mathtt{Fast}, \mathtt{Moderate}, \mathtt{Slow}\}, \\ \mathtt{C_{UserPreference}} &= \{\mathtt{EnergySaving}, \mathtt{FineAccuracy}\} \end{aligned} \tag{5.11}$$

As suggested by the energy-aware modeling framework, the identified context information and constraints can be categorized into *Primary Influencing Context, Secondary Influencing Context, and User Influencing Context.*

5.3.2.1 Primary influencing context

From the map navigation application, *Battery Percentage* and *Battery State* can be considered as a primary influencing context. Therefore, these two context information have *Mandatory* relationship and must be used in all the situations as it decides energy-saving adaptations. In addition, some set of values of these two context information may affect the network speed and user preferences. The *Battery Percentage* has context values of *High, Medium, and Low* to change application behavior corresponding to the mobile's battery life. These context values have an *Alternative* relationship with its parent, and therefore, only one value can be read at a time. Similarly, the *Battery State* has context values of *Charging or Discharging* with *Alternative* relationship.

5.3.2.2 Secondary influencing context

The context information, *Network Speed* can be considered for this category as it has an indirect impact on the overall energy consumption. In addition, the selection of some context values depends on the values of primary influencing contexts. It has *Optional* relationship, therefore adding this context information at run-time is not mandatory. However, the network speed further has the context values of *Fast, Moderate, and Slow* with an *Alternative* relationship.

5.3.2.3 User influencing context

Finally, the *User Preference* can be considered under this category. This context information has *Mandatory* relationship with the values *Energy Saving, and Fine Accuracy*. This context information is mainly used to reset the user's preferences at run-time, leading to abnormal battery drain. Therefore, these context values have an *Alternative* relationship between its values to set the contrasting dynamic requirements. The context values can be changed by the application at run-time in response to the change in values of *Battery Percentage, Battery State, and Network Speed* if the specified user preference is not energy-friendly.

5.3.2.4 Context constraints

As illustrated in Figure 5.4, context constraints are established between some set of context information for energy-savings. The context constraints are represented in the

TABLE 5.2: Sample valid triggering situations of map navigation application

Context	VTS1	VTS2	VTS3	VTS4	VTS5	VTS6
Battery Percentage	×	×	×	×	×	×
High						×
Medium		×	×	×	×	
Low	×					
Battery State	×	×	×	×	×	×
Charging			×	×	×	
Discharging	×	×				×
Network Speed		×	×	×	×	×
Fast					×	×
Moderate				×		
Slow		×	×			
User Preference	×	×	×	×	×	×
Energy Saving	×	×	×			
Fine Accuracy				×	×	×

Equation 5.12:

$$\begin{aligned} Low \wedge Discharging &\Rightarrow \text{"}EnergySavings\text{"}, \\ Low \wedge Charging \wedge Slow &\Rightarrow \text{"}EnergySavings\text{"}, \\ Medium \wedge Discharging \wedge Slow &\Rightarrow \text{"}EnergySavings\text{"} \end{aligned} \tag{5.12}$$

The first constraint of the Equation 5.12, signifies the smartphone is running out of battery through *Low* battery percentage and the *Discharging* battery state. At this situation, the *User Preferences* value *Fine Accuracy* cannot be set as it is a energy-hungry requirement. Similarly, the second constraint signifies the same critical situation with *Low* battery percentage, *Charging* battery state, and *Slow* network speed. Though the battery is in a charging situation, running an energy-hungry operation at slow network speed may increase the discharging rate compared to the charging rate. Therefore, in this situation, strict *Energy Savings* must be enabled. The third constraint is also used for a similar purpose.

5.3.2.5 Valid triggering situations

The context values, context constraints that are illustrated in Figure 5.4 are applied along with modified rules. The valid triggering situations are to be identified in such a way that they trigger energy-saving adaptations. The valid triggering situations derived from the context-model are listed in Table 5.2. For simplicity, only the minimal set of representative context situations have been listed. From Table 5.2, the following valid triggering situations can be derived:

- **VTS1**: (*Battery Percentage, Low, Battery State, Discharging, User Preference, Energy Saving*)
- **VTS2**: (*Battery Percentage, Medium, Battery State, Discharging, Network Speed, Slow, User Preference, Energy Saving*)
- **VTS3**: (*Battery Percentage, Medium, Battery State, Charging, Network Speed, Slow, User Preference, Energy Saving*)
- **VTS4**: (*Battery Percentage, Medium, Battery State, Charging, Network Speed, Moderate, User Preference, Fine Accuracy*)
- **VTS5**: (*Battery Percentage, Medium, Battery State, Charging, Network Speed, Fast, User Preference, Fine Accuracy*)
- **VTS6**: (*Battery Percentage, High, Battery State, Discharging, Network Speed, Fast, User Preference, Fine Accuracy*)

With the basic analysis of energy-aware context, constraints, and valid triggering situations, the energy-aware context model can be applied to the basic feature model illustrated in Figure 5.3. This activity aims to find out more energy-saving opportunities on the basic feature model. At the end of applying the context model, the energy-aware feature model can be generated with more flexibility to choose the run-time binding of energy-efficient ACs. Table 5.3 presents the summary of finding energy-efficient alternatives for energy-hungry features. The information presented in this table will be useful to modify the basic feature model illustrated in Figure 5.3 towards supporting more flexible energy-saving adaptations at run-time.

TABLE 5.3: Summary of converting basic feature model to energy-aware feature model

Category	**Context Values**	**Suitable Features**
Battery Level	High	On sensing this context value, the application can bind energy-hungry operations such as *GPS, Satellite View, Live Traffic, On the Go, Audio*.
	Medium	The application cannot run *GPS* continuously as it may drain the battery very fast on sensing this context value. There should be a mechanism to periodically activate GPS to find out the user's current exact location. Unfortunately, the basic feature model does not contain any such mechanism. Therefore, for better energy-saving, some complementary methods should be in place to improve the efficiency of the GPS.
	Low	On sensing this context, the application must avoid using energy-hungry operations. In the existing basic feature model, there is an option for disabling *On the Go, Audio*. Unfortunately, there is no option for disabling *GPS, Satellite View, and Live Traffic*. Therefore, the energy-aware feature model must be able to provide an energy-efficient alternative for those features.
Battery Sate	Charging	On sensing this context value, battery level must be checked for the value *Low* battery percentage and *Slow* network speed to enforce strict energy-saving adaptations. Then energy-hungry operations are not allowed to bind. If the battery percentage is *Medium or High* and network speed is *Moderate or Fast* the application can be set as *Fine Accuracy*. That is *GPS* can be used without any restrictions.
	Discharging	This context value can be combined with the context values of *Battery Percentage* = *Low* and *Network Speed* = *Slow* to add more flexibility to choose a energy-efficient AC.

Continued on next page

Table 5.3 – continued from previous page

Category	**Context Values**	**Suitable Features**
Network Speed	Fast	On sensing this context information, energy hungry-features like *Satellite View, Live Traffic, On the Go, Audio* can be built without any degradation provided the battery context supports *Fine accuracy* preferences.
	Moderate	This context value requires an energy-efficient way of using energy-hungry operations without much degradation in the accuracy requirements. Unfortunately, there is no provision for improving location sensing for this context value. However, features like *Live traffic, On the Map* can be built if there is sufficient battery level.
	Slow	On sensing this context value, there should be strict restrictions applied on *Satellite View, On the Go, On the Map*. Executing these energy-hungry features on a slow network connection may take more time and drain the battery faster. Therefore, on finding slow network connectivity, the application must switch to energy-saving mode.
User Preference	Energy Saving	This context value directly implies that only energy-friendly features can be selected for the operations of the application.
	Fine Accuracy	This context information implies that energy-hungry operations like *GPS* can be used if there are enough battery and a fair network connection.

5.3.3 Energy-aware feature model

This subsection presents the process of including energy-aware features into basic feature models (Figure 5.3). As an initial step, the inputs from Table are analyzed, and the following decisions are taken:

- There is no energy-efficient alternative for *GPS*. Therefore, *WiFi and Cell-ID* have been added to the feature model as energy-efficient alternatives of *GPS*.
- There is no complimentary method to update location information periodically. Therefore, two features, namely *Sensing Interval and Accelerometer*, have been added. The purpose of the *Accelerometer* is to find out if the user moved too far from the last known location, which will be used for deciding *Sensing Interval* value. More the sensing interval, the lesser the energy consumption.
- *Satellite View and Normal View* can be considered an energy-efficient alternative, and therefore an *Alternative* relationship has been added.
- *On the Go, On the Map, Text* features are considered as energy-efficient-alternatives. Therefore, an *Alternative* relationship has been added.
- Appropriate energy-aware constraints are added.

The extension to the basic feature model presented in Subsection 5.2.2 has been applied on the basic feature model, and the modified energy-aware feature model is shown in Figure 5.5. The features of an energy-aware map navigation application can be formally represented, as shown in Equation 5.13.

$$\mathbf{F}_{\mathbf{EA-GPSNavigation}} = \{\mathtt{F}_{\mathtt{LocationSensing}}, \mathtt{F}_{\mathtt{SensingInterval}}, \mathtt{F}_{\mathtt{Accelerometer}}, \mathtt{F}_{\mathtt{MapView}}, \mathtt{F}_{\mathtt{Directions}}, \mathtt{F}_{\mathtt{Navigation}}\} \tag{5.13}$$

The sub-features of energy-aware GPS navigation application can be represented, as shown in Equation 5.14.

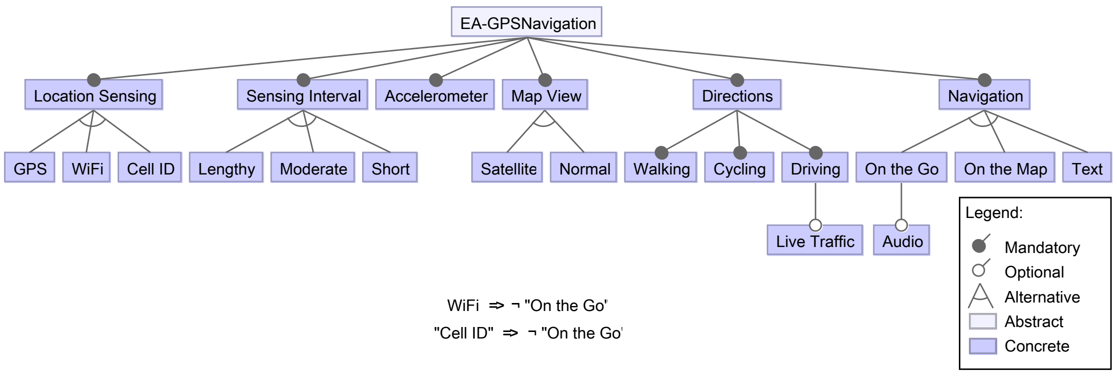


FIGURE 5.5: Energy-aware feature model of map navigation application

$$
\begin{aligned}
\mathtt{F}_{\mathtt{LocationSensing}} &= \{\mathtt{GPS}, \mathtt{WiFi}, \mathtt{CellID}\},\\
\mathtt{F}_{\mathtt{SensingInterval}} &= \{\mathtt{Lengthy}, \mathtt{Moderate}, \mathtt{Short}\},\\
\mathtt{F}_{\mathtt{MapView}} &= \{\mathtt{Satellite}, \mathtt{Normal}\},\\
\mathtt{F}_{\mathtt{Directions}} &= \{\mathtt{Walking}, \mathtt{Cycling}, \mathtt{Driving}\},\\
\mathtt{F}_{\mathtt{Driving}} &= \{\mathtt{LiveTraffic}\},\\
\mathtt{F}_{\mathtt{Navigation}} &= \{\mathtt{OntheGo}, \mathtt{OntheMap}, \mathtt{Text}\},\\
\mathtt{F}_{\mathtt{OntheGo}} &= \{\mathtt{Audio}\}
\end{aligned}
\tag{5.14}
$$

The next step in the energy-aware modeling is to categorize the identified features into the following categories: *energy-hungry, energy-friendly, energy-efficient alternatives, energy-aware constraints, and energy-aware application configurations*.

5.3.3.1 Energy-friendly features

The basic feature diagram of the map navigation application does not contain any energy-friendly features. Therefore two features, namely *Sensing Interval* and *Accelerometer* have been added to the energy-aware feature model. The newly added features can be represented as shown in Equation 5.15:

$$
\begin{aligned}
\mathrm{EF} &= \{\mathtt{SensingInterval}, \mathtt{Accelerometer}\},\\
\mathrm{EF}_{\mathtt{SensingInterval}} &= \{\mathtt{Lengthy}, \mathtt{Moderate}, \mathtt{Short}\}
\end{aligned}
\tag{5.15}
$$

Mandatory relationship has been established for *Sensing Interval* and *Accelerometer* with it's parent, hence it can be added to the application configuration by default. Further, the *Alternative* relationship has been added for the sub-features of *Sensing Interval* to give more flexibility for run-time energy-saving. Therefore, only one sub-feature can be added at a time.

5.3.3.2 Energy-hungry features

The identified energy-hungry features are *Location Sensing, Map View, Directions, Navigation* as they consume more battery for its operation. Specifically, *Location Sensing, Map View, Directions, Navigation* has energy-hungry operations such as *GPS, Satellite, Driving (Live Traffic), On the Go (Audio)* respectively. The different energy-hungry features are represented in the Equation 5.16:

$$\begin{aligned}
\mathrm{EH} &= \{\mathtt{LocationSensing}, \mathtt{MapView}, \mathtt{Directions}, \mathtt{Navigation}\}, \\
\mathrm{EH}_{\mathtt{LocationSensing}} &= \{\mathtt{GPS}\}, \\
\mathrm{EH}_{\mathtt{MapView}} &= \{\mathtt{Satellite}\}, \\
\mathrm{EH}_{\mathtt{Directions}} &= \{\mathtt{Walking}, \mathtt{Cycling}, \mathtt{Driving}\}, \\
\mathrm{EH}_{\mathtt{Driving}} &= \{\mathtt{LiveTraffic}\}, \\
\mathrm{EH}_{\mathtt{Navigation}} &= \{\mathtt{OntheGo}\}, \\
\mathrm{EH}_{\mathtt{OntheGo}} &= \{\mathtt{Audio}\}
\end{aligned} \tag{5.16}$$

As shown in Figure 5.5, all the identified energy-hungry features of *EH* (refer Equation 5.16) have a *Mandatory* relationship with its parent. Therefore, these features must be added to all application configurations by default. As per the energy-aware modeling framework rules, the domain analyst has to specify energy-efficient alternatives if any energy-hungry operation has *Mandatory* relationship. Therefore, energy-efficient alternatives are added for *GPS, Satellite, On the Go* as represented in Equation 5.17. Further, the energy-hungry features *Live Traffic* and *Audio* are *Optional* features with its parent, and hence adding energy-efficient alternatives is not required.

5.3.3.3 Energy-efficient alternatives

The next step is to add energy-efficient alternatives to the feature model. As shown in Equation 5.17, *GPS* has *WiFi and Cell ID*; *Satellite* has *Normal*; and *On the Go* has *On the Map and Text* as an energy-efficient alternatives.

$$\begin{aligned} \mathtt{EA}_{\mathtt{GPS}} &= \{\mathtt{WiFi}, \mathtt{CellID}\}, \\ \mathtt{EA}_{\mathtt{Satellite}} &= \{\mathtt{Normal}\}, \\ \mathtt{EA}_{\mathtt{OntheGo}} &= \{\mathtt{OntheMap}, \mathtt{Text}\} \end{aligned} \tag{5.17}$$

The *Alternative* relationship has been added to energy-efficient alternatives and its corresponding energy-hungry features as suggested in the energy-aware modeling framework. For instance, *Location Sensing* can use any one of *GPS or WiFi or Cell ID* at any given time. Similarly, the *Alternative* relationship has been established between the choice of *Satellite* and *On the Go*.

5.3.3.4 Energy-aware constraints

As shown in Figure 5.5, energy-aware constraints have been added to the energy-aware feature model. The list of energy-aware constraints are represented in Equation 5.18:

$$\begin{aligned} WiFi &\Rightarrow \neg\text{"}OntheGo\text{"}, \\ CellID &\Rightarrow \neg\text{"}OntheGo\text{"}, \end{aligned} \tag{5.18}$$

These energy-aware context-constraints have *Exclude* relationship for *On the Go*. This signifies that *On the Go* cannot be used in the application configuration when the *Location Sensing* is equal to *WiFi or Cell ID*. *On the Go* navigation requires accurate location information from *GPS*. As *WiFi and Cell ID* failed to provide accurate location information, using energy-hungry operation like *On the Go* may increase the overall energy consumption unnecessarily. Therefore, it is advisable not to use *On the Go* when there is no accurate location information.

TABLE 5.4: Energy-aware application configurations of map navigation application

Context	EAC1	EAC2	EAC3	EAC4	EAC5	EAC6
Location Sensing	×	×	×	×	×	×
GPS					×	×
WiFi				×		
Cell ID	×	×	×			
Sensing Interval	×	×	×	×	×	×
Lengthy	×					
Moderate		×		×	×	
Short			×			×
Accelerometer	×	×	×	×	×	×
Map View	×	×	×	×	×	×
Satellite				×	×	×
Normal	×	×	×			
Directions	×	×	×	×	×	×
Walking	×	×	×	×	×	×
Cycling	×	×	×	×	×	×
Driving	×	×	×	×	×	×
Live Traffic					×	×
Navigation	×	×	×	×	×	×
On the Go						×
Audio						×
On the Map				×	×	
Text	×	×	×			

5.3.3.5 Energy-aware application configurations

Energy-aware application configurations are selected in such a way that it corresponds to any valid triggering situations. From Table 5.2, 6 valid triggering situations have been found. Therefore, corresponding six energy-aware application configurations are identified and listed in Table 5.4. Each energy-aware application configuration can be considered as a member of a family of map navigation applications for energy-savings. These identified application configurations can be executed through self-adaptation at run-time when the application encounters the corresponding triggering situations. The identified application configurations are:

- **EAC1**: (*Location Sensing, Cell ID, Sensing Interval, Lengthy, Accelerometer, Map View, Normal, Directions, Walking, Cycling, Driving, Navigation, Text*)
- **EAC2**: (*Location Sensing, Cell ID, Sensing Interval, Moderate, Accelerometer, Map View, Normal, Directions, Walking, Cycling, Driving, Navigation, Text*)
- **EAC3**: (*Location Sensing, Cell ID, Sensing Interval, Short, Accelerometer, Map View, Normal, Directions, Walking, Cycling, Driving, Navigation, Text*)
- **EAC4**: (*Location Sensing, WiFi, Sensing Interval, Moderate, Accelerometer, Map View, Satellite, Directions, Walking, Cycling, Driving, Navigation, On the Map*)
- **EAC5**: (*Location Sensing, GPS, Sensing Interval, Moderate, Accelerometer, Map View, Satellite, Directions, Walking, Cycling, Driving, Live Traffic, Navigation, On the Map*)
- **EAC6**: (*Location Sensing, GPS, Sensing Interval, Short, Accelerometer, Map View, Satellite, Directions, Walking, Cycling, Driving, Live Traffic, Navigation, On the Go, Audio*)

5.3.4 Energy-aware adaptation model

The final step of the energy-aware modeling framework is to plan the energy-saving adaptation policies by combining the energy-aware context model and energy-aware feature model results. Here, the energy-aware modeling framework adopts the ECA rules structure for the specification of energy-saving adaptation policies. Here, *event* is the change in context value, *condition* is to check the adaptation policies for matching plan, *action* is binding energy-efficient versions at run-time. The sample or representative energy-saving adaptation policies can be modeled as represented in Equation 5.19 and in Figure 5.6. Initially, if the application encounter *VTS 6*, the *EAC6* could be bounded at run-time. As per the plan, the system initially starts with energy-hungry configurations as it uses all the energy-hungry features. Now, the goal of the system is to change to energy-saving state as it encounters corresponding degradation in the context information. As shown in Figure 5.6, *Battery Percentage* drops down from *High to Medium*, it encounters the state *VTS5*. Hence, as specified in *ESA5*, the corresponding

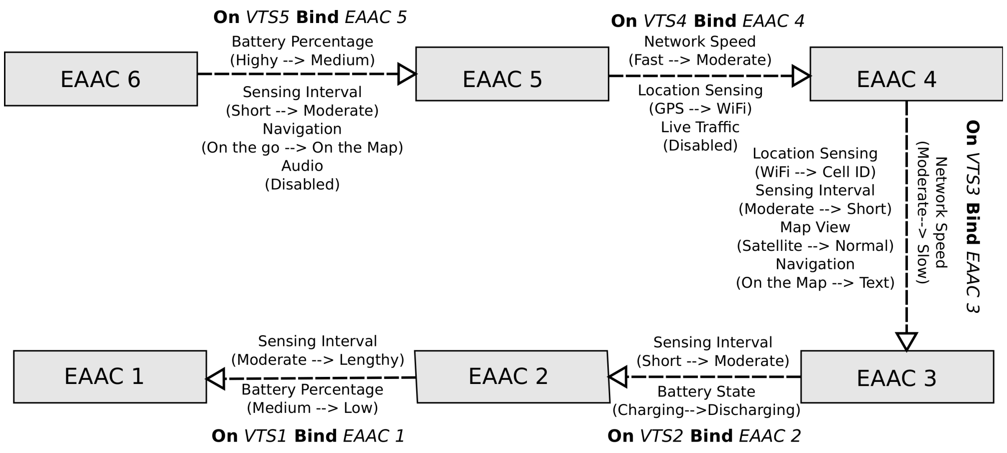

On *VTS5* **Bind** *EAAC 5*
Battery Percentage (Highy --> Medium)
Sensing Interval (Short --> Moderate)
Navigation (On the go --> On the Map)
Audio (Disabled)
On *VTS4* **Bind** *EAAC 4*
Network Speed (Fast --> Moderate)
Location Sensing (GPS --> WiFi)
Live Traffic (Disabled)
On *VTS3* **Bind** *EAAC 3*
Network Speed (Moderate--> Slow)
Location Sensing (WiFi --> Cell ID)
Sensing Interval (Moderate --> Short)
Map View (Satellite --> Normal)
Navigation (On the Map --> Text)
EAAC 6
EAAC 5
EAAC 4
EAAC 3
EAAC 2
EAAC 1
Sensing Interval (Short --> Moderate)
Battery State (Charging-->Discharging)
On *VTS2* **Bind** *EAAC 2*
Sensing Interval (Moderate --> Lengthy)
Battery Percentage (Medium --> Low)
On *VTS1* **Bind** *EAAC 1*

FIGURE 5.6: A sample energy-aware adaptation plan for valid context changes

application configuration *EAC5* can be bounded with *Sensing Interval* changed from *Short to Moderate*; *Navigation* changed from *On the go to On the Map*; *Audio* feature is *Disabled.* When the application encounters a new context change, that is *Network Speed* drops from *Fast to Moderate*, the *VTS4* can be realized and as per *ESA4*, the corresponding *EAC4* will be bounded by changing *Location Sensing* from *GPS to WiFi* and *Live Traffic* being removed or disabled from the binding. Similarly, the application can be dynamically bounded with appropriate configurations based on *ESA3, ESA2* when it encounters *VTS3, VTS2* respectively. Finally, when the *Battery Percentage* drops to *Low*, the application must be in an energy-saving state. At this situation , the *ESA1* will instruct the adaptation engine to bind the *EAC1* which is most energy-efficient configuration of the given navigation application example.

$$\begin{aligned}
ESA_1 &= On\ VTS_1\ Bind\ EAC_1,\\
ESA_2 &= On\ VTS_2\ Bind\ EAC_2,\\
ESA_3 &= On\ VTS_3\ Bind\ EAC_3,\\
ESA_4 &= On\ VTS_4\ Bind\ EAC_4,\\
ESA_5 &= On\ VTS_5\ Bind\ EAC_5,\\
ESA_6 &= On\ VTS_6\ Bind\ EAC_6
\end{aligned} \tag{5.19}$$

In summary, the energy-aware context, feature, and adaptation modeling can be efficiently applied to analyze energy-saving opportunities at run-time. As illustrated in the map navigation application, the input from the context model can be used to change the basic feature models to suit energy-savings and self-adaptation. Finally, the energy-aware adaptation can be modeled at design-time itself before developing the application. This facilitates the decision-making about the run-time energy-saving adaptation at design-time, reducing the developers' development efforts.

5.3.5 Discussions

The case study results show that the energy-aware modeling framework can serve as an effective way to help domain analysts plan energy-saving adaptations in the early stages of software development. Based on guidelines from Easterbrook et al. (2008), a case study based empirical method has been used to validate the energy-aware modeling framework. The case study aimed at answering the following research questions:

- **RQ1: How to introduce the energy-related requirements at design-time for the family of map navigation applications?** The case study scenario presented in the previous sub-sections shows how to introduce energy-aware context information, features of the energy-aware self-adaptive navigation application using an energy-aware modeling framework. The energy-aware context information can be further classified into *primary, secondary, alternative and user* influencing context to prioritize context information for triggering energy-saving adaptations. Furthermore, the energy-aware context model helps designers identify energy-saving opportunities on the existing basic feature model. Similarly, the energy-aware features can be classified into *energy-friendly, energy-hungry, energy-efficient alternative* to decide the most suitable version of features for energy-savings at run-time.

- **RQ2: How to design an energy-saving self-adaptation plan for the family of map navigation applications?** The case study results show how energy-awareness can be introduced during adaptation modeling. The valid triggering situations derived from the energy-aware context model and energy-aware application configurations derived from the energy-aware feature model are well utilized in the case study to model energy-saving adaptation policies. The case study results show six sets of VTS configurations and corresponding EACs for energy-aware self-adaptation. The sample adaptation plan illustrated in Figure 5.6 is evidence for modeling energy-savings opportunities of map navigation applications.

- **RQ3: What is the impact of an energy-aware modeling framework on designing a family of map navigation applications?** The proposed modeling framework produces energy-aware design-time models of navigation applications in the early stages of software development. Introducing dynamic requirements early in the life cycle of the software project will give clear information to developers. The developers can efficiently use energy-aware models and adaptation plans to develop a dynamically adaptive application. Design-time models can be mapped easily to compile-time or deployment-time, or run-time models for dynamic binding or reconfigurations of the application.

To the best of our knowledge, sufficient conceptual contribution has been added to the model of energy-aware context, features, and adaptation requirements. To summarize, the energy-aware modeling framework has made the following contributions:

1. Adopted feature models to specify energy-aware context information
2. Adopted feature models to specify energy-aware feature information
3. Extended rules and energy-aware categories for specifying the context and feature based on its impact on energy consumption
4. A way to derive Valid Triggering Situations (VTS) and Energy-aware Application Configurations (EAC) through energy-aware context model and energy-aware feature model
5. ECA based method for modeling energy-saving adaptations through VTS and EAC

The presented modeling framework uses feature models to specify energy-aware context information, features, and self-adaptation adaptation plans. The modified rules introduced energy-awareness into the existing basic feature model of the map navigation application. The case study results show that the context categories and application categories can be easily derived from energy consumption analysis. The valid triggering situation and energy-aware application configurations provide clear information on modeling energy-saving adaptations. Through the case study, it has been shown that how energy-saving adaptations could be planned.

5.4 Threats to Validity

In this section, the potential threats to the validity of the presented case study are discussed. The guidelines given by Runeson and Höst (2009) are followed to categorize and discuss the threats.

Construct Validity

Construct validity refers to the correctness of interpretation and measurement of the theoretical constructs. The modeling language with high construct validity would reduce the confusion while interpreting the models produced using the language. In the energy-aware modeling framework, the primary threat to validity is the usage of FeatureIDE notations for representing energy-aware context and feature categories. As the default notations of FeatureIDE have a different meaning compared to categories presented in this modeling framework, there is a high possibility of confusion while interpreting the model. The second threat to construct validity is the manual derivation of energy-saving adaptation plans, and there is no structured format to represent the adaptation plans. This might make the readers interpret the energy-saving adaptation models differently. This threat can be addressed by providing a dedicated notation for energy-aware modeling, as presented in Chapter 6. In addition, the introduction of structured XML notations for energy-saving adaptation plans in Chapter 6 would mitigate the threat of misinterpretation of models by different stakeholders. However, this framework's significant shortcoming is the manual derivation of valid triggering situations and energy-aware application configurations. This issue is addressed by providing tool support for the presented conceptual modeling framework.

Internal Validity

Internal validity refers to the causal relationships between the energy-aware context model, energy-aware feature models, and energy-saving adaptation plans. There is a possibility of validity threats in deciding the factors that affect smartphone applications' battery consumption. Therefore, in the energy-aware modeling framework, the different categories of context information like primary, secondary, and user-influencing context have been added. Thus, the model would serve the purpose of analyzing the relationship between factors and battery consumption while using this modeling framework. Similarly, suitable categories such as energy-hungry and energy-friendly were added to the feature model to include the ability to analyze the relationship between features and battery consumption. The other threat to internal validity is related to analyzing the relationship between the energy-aware context model and the energy-aware feature model. This threat was mitigated by adding energy-saving adaptation plans to the energy-aware modeling framework. Overall, the models are designed so that it considers the factors affecting battery consumption and other influencing factors. From the case study, it is evident that there is less possibility of internal threat while using this model, as it covers most of the aspects of causal relationships in the domain of energy-aware self-adaptive software.

External Validity

The external validity refers to the generalizability of the models and case study results presented in this chapter. There is a possibility of validity threats in terms of the application domain considered in this case study. The location-based Smartphone applications considered in this case study can only be generalized to the family of location-based applications. On the other hand, it would not be possible to generalize the case study results to different application domains. However, the results presented for *battery percentage* and *battery state* can be generalized to other applications that can be used on smartphones. Further, the concepts such as categories about context model, feature model, and adaptation model have kept application domain-independent to increase the generalizability. Hence, the presented energy-aware modeling framework is generic for all energy-aware self-adaptive applications. Thus, the energy-aware modeling framework can be re-used for any other domain to introduce energy-awareness and self-adaptivity. Hence, the method and results presented in this chapter are conceptual starting points for further research in modeling energy-saving adaptation policies.

Reliability

Reliability refers to the extent to which the presented case study is repeatable by other researchers. A possible threat to validity under this category is the unavailability of dedicated notations and tool support for the energy-aware context model and energy-aware feature model of the proposed modeling framework. As the proposed modeling framework uses the FeatureIDE interface, there is a high possibility of error while the researchers repeat the case study after few years. This threat is addressed by providing a separate set of notations and tool support developed as part of this research work presented in Chapter 6.

5.5 Summary

This chapter presents the energy-aware modeling framework to explicitly consider energy-efficiency-related requirements in the early stages of software development. A basic feature model provided by Kang et al. (1990) has been adopted to specify energy-efficiency related information. The contributions of this objective is to provide an energy-aware modeling framework for domain analysts to introduce energy-aware requirements in the early software development phases. The energy-aware modeling framework provides a systematic way of analyzing energy-related requirements and situations to enforce energy-saving self-adaptations. This framework adopts feature diagrams to model energy-aware requirements. It primarily provides extended notations and rules for using feature models to introduce energy-related requirements. In addition, it provides a separate classification mechanism for the features and context of the software under development. For instance, the context classification includes *Primary-influencing Context. Secondary-influencing Context, and User-influencing Context*. Likewise, the feature classification includes *Energy-friendly Feature, Energy-hungry Feature, and Energy-efficient Alternatives*. Here, *context information* refers to the device operating conditions or user preferences that affect software's energy-efficiency. On the other hand, *features* refer to the functional requirements of the software. The outcome of the energy-aware modeling framework is a set of Valid Triggering Situations (VTS) and Energy-efficient Application Configurations (EAC). Further, the modeling framework provides Event Condition Action (ECA) based adaptation rules for specifying the energy-saving adaptation plan. As proof of concept, the energy-aware modeling framework has been applied to a family of map navigation applications. The case study results show that the energy-aware modeling framework can be effectively used to specify energy-aware requirements of the family of software systems.

Chapter 6

ENERGY-SAVING ADAPTATION PLANNER

The energy-aware modeling framework presented in Chapter 5 (Research Objective II) provides conceptual extensions to feature models. The domain analyst has to use the existing feature modeling tools such as FeatureIDE to model the energy-aware requirements. However, the current feature modeling tools do not have specific notations for differentiating context categories and feature categories. Therefore, this objective aims to develop dedicated notations and tool support for energy-aware context modeling, energy-aware feature-modeling, and energy-saving adaptation planning. This chapter presents a tool named "energy-Saving Adaptation Planner (eSAP)" that has been developed by extending a popular feature modeling tool FeatureIDE. The developed tool would help the domain analyst to model, label, and validate the energy-saving adaptation plans in the early stages of software development. The notations are described in Section 6.2 along with tool description. In Section 6.3, the tool's efficacy is presented and validated through a case study focusing on map navigation application requirements.

6.1 Extending FeatureIDE

A popular feature modeling tool FeatureIDE (Thüm et al., 2014) has been extended to develop tool support for the energy-aware modeling framework presented in the previous chapter. FeatureIDE is an Eclipse-based open source tool for feature-oriented software development. FeatureIDE supports the following feature-oriented software development techniques: *feature-oriented programming, aspect-oriented programming, delta-oriented programming, and preprocessors*. It is widely used in the development of Software Product Lines (SPL) (Clements and Northrop, 2001). The FeatureIDE supports the development of such re-usable SPLs in an Integrated Development Environment (IDE). It supports the following phases of FOSD for developing SPL: *domain analysis, domain design, domain implementation, requirements analysis, software generation, and quality assurance*. In this thesis, the domain analysis and requirement analysis phases of FeatureIDE are extended to provide tool support for an energy-aware modeling framework.

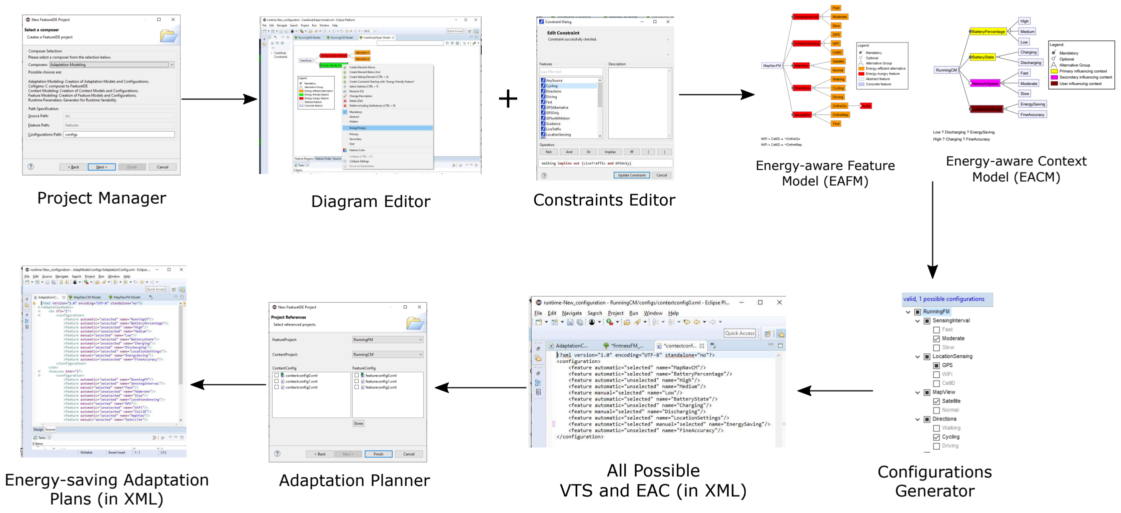
The domain analysis phase of FeatureIDE produces the feature models to identify the commonalities and variabilities among the family of software systems. FeatureIDE provides a diagram editor with graphical notations to create feature models. The graphical notations cover the relationships mentioned in FODA such as `and`, `or`, `xor` in the form of *mandatory, optional, alternative* features. In addition, it supports the *abstract, concrete, and dead* feature notations in the feature diagram. The crosstree constraints like *includes and excludes* are supported by the propositional formula notations in the featureIDE. Finally, FeatureIDE applies the relationships and cross-tree constraints on the features mentioned in the feature diagram to generate a set of valid configurations that are a subset of all features. It also allows the user to change the configurations to check if the specified configurations are valid or invalid. Each valid configuration would be considered as a member of a family of software products. FeatureIDE supports both visual and textual representation of the feature diagram, valid and invalid configurations. The graphical representation is in the form of a feature diagram with colored notations to differentiate feature groups. The textual notations are in the form of XML for features, relationships, cross-tree constraints, and generated configurations.

FeatureIDE is selected to develop energy-Saving Adaptation Planner (*e*SAP) as it is open-source software. The development team has provided several extension points for building our ideas as an Eclipse plugin. Specifically, the extension points of *feature diagram editor and configuration editor* have been used to develop *e*SAP. As presented in the energy-aware modeling framework, context information specification is essential in modeling energy-aware self-adaptive software. The energy-aware modeling framework highlights the importance of energy-saving adaptation plans to take decisions at the design phase. Hence, there is a distinct lack of support to produce adaptation models combining feature and context models. Moreover, the FeatureIDE cannot explicitly introduce energy-related requirements to the feature and context model. Therefore this objective aims at providing the following support:

1. Ability to introduce energy impact of features and energy-efficient alternatives with suitable energy-aware labels
2. Ability to introduce suitable context information and label them with its energy impact
3. Ability to plan energy-saving adaptations

The existing implementation of FeatureIDE supports only the feature model and does not support the specification of context information. Therefore, it has been decided to extend the FeatureIDE project types to have three models, namely, *energy-aware context model, energy-aware feature model, and adaptation model*. The feature diagram editor of FeatureIDE is extended to introduce the new notations. The energy-aware context model and energy-aware feature models are generated using the diagram editor. On the other hand, the adaptation models are generated using the configuration editor of the FeatureIDE. In *e*SAP, the configuration editor of FeatureIDE is extended as Configuration generator to allow the user to save the customized configurations of their choice using the energy-aware context and energy-aware feature models. Finally, the XML files of the configurations are used to generate the adaptation model specified by the domain analyst. The eSAP source code is hosted on GitHub[1] and is freely available for usage and contribution. The remaining part of this chapter presents the tool description, notations, and case study related to *e*SAP.

[1] https://github.com/marimuthuc/esap

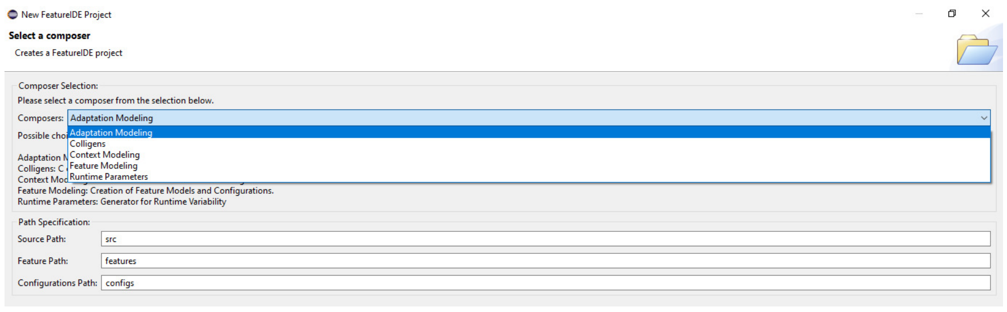

Project Manager
Diagram Editor
Constraints Editor
Energy-aware Feature Model (EAFM)
Energy-aware Context Model (EACM)
Configurations Generator
All Possible VTS and EAC (in XML)
Adaptation Planner
Energy-saving Adaptation Plans (in XML)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration>
    <feature automatic="selected" name="MapNavCM"/>
    <feature automatic="selected" name="BatteryPercentage"/>
    <feature automatic="unselected" name="High"/>
    <feature automatic="unselected" name="Medium"/>
    <feature manual="selected" name="Low"/>
    <feature automatic="selected" name="BatteryState"/>
    <feature automatic="unselected" name="Charging"/>
    <feature manual="selected" name="Discharging"/>
    <feature automatic="selected" name="LocationSettings"/>
    <feature automatic="selected" manual="selected" name="EnergySaving"/>
    <feature automatic="unselected" name="FineAccuracy"/>
</configuration>
```

FIGURE 6.1: Primary components and generated artifacts of *e*SAP

6.2 *e*SAP Description

*e*SAP aids domain analysts to enumerate all the possible features and context values that are associated with a particular domain. This tool also allows them to assign labels that represent the energy impact of these features and context information on the overall energy consumption of the system. For this purpose, popular feature modeling tool FeatureIDE has been extended using the available extension points. The following extensions have been made to develop *e*SAP: *project manager, diagram editor, constraint editor, configuration generator, and adaptation planner*. Using these extensions, the *e*SAP is able to produce three artifacts in the requirement collection phase: (1) Graphical energy-aware feature model, (2) Graphical energy-aware context model, (2) Textual energy-saving adaptation plans. The overall workflow of *e*SAP is depicted in Figure 6.1 with its components and produced artifacts. As shown Figure 6.1, the components *diagram editor and constraints editor* produce either *energy-aware context model or energy-aware feature model*. Similarly, the *configuration generator* produces XML-based configurations called as *valid triggering situations (VTS) and energy-aware application configurations (EAC)*. Finally, the *energy-saving adaptation planner* combines the VTS and EAC to provide the XML-based self-adaptation plans that represent the run-time reconfiguration for particular battery-related contextual change. This section presents the details about the components of *e*SAP and descriptions of graphical


New FeatureIDE Project

Select a composer

Creates a FeatureIDE project

Composer Selection:

Please select a composer from the selection below.

Composers: Adaptation Modeling

Possible choi

Adaptation M

Colligens: C

Context Mod

Feature Modeling: Creation of Feature Models and Configurations.

Runtime Parameters: Generator for Runtime Variability

Path Specification:

Source Path: src

Feature Path: features

Configurations Path: configs

FIGURE 6.2: A screenshot of project manager that shows different types of projects

notations.

Project manager: The first extension has been introduced to the project manager to create appropriate project types for three different artifacts. As shown in Figure 6.2, the following project types are added in *e*SAP: *feature modeling, context modeling, and adaptation modeling*. Using the extended project manager, the domain analyst can select the appropriate model type during the new project creation wizard for the domain under investigation. Here, the project type *feature modeling* refers to the *energy-aware feature model* of the framework presented in Section 5.2.2. Similarly, the *context modeling* refers to the *energy-aware context modeling* and *adaptation modeling* refers to the *energy-saving adaptation planning* of the energy-aware modeling framework presented in Section 5.2.1 and Section 5.2.3.

Diagram editor: The second extension has been made to the feature diagram editor of FeatureIDE. As explained in Section 5.2, the energy-aware modeling framework contains two different feature diagrams. For this purpose, *e*SAP allows the domain analyst to create separate feature diagram types for specifying features and context information. The FeatureIDE's feature model editor has been extended to add energy-aware labels, as mentioned in Section 5.2. FeatureIDE uses colored notation to differentiate the *abstract feature, concrete feature, dead feature, etc* of the basic feature model. Similarly, suitable colored notations (See Table 6.1 and Table 6.2) have been added to *e*SAP to differentiate different labels of energy-aware feature model and energy-aware context model.

As shown in Table 6.1, the *e*SAP consists of three labels for specifying the energy impact of each feature identified from the requirements. A domain analyst can assign

TABLE 6.1: Notations of energy-aware feature model

Notation and Label	**Allowed Relationship(s)**	**Description**
🟥 Energy-hungry feature	Optional	By default
	Mandatory	Must specify Energy-efficient alternatives
🟧 Energy-efficient alternative	Alternative	Child of Energy-hungry features
🟩 Energy-friendly feature	Mandatory	By default

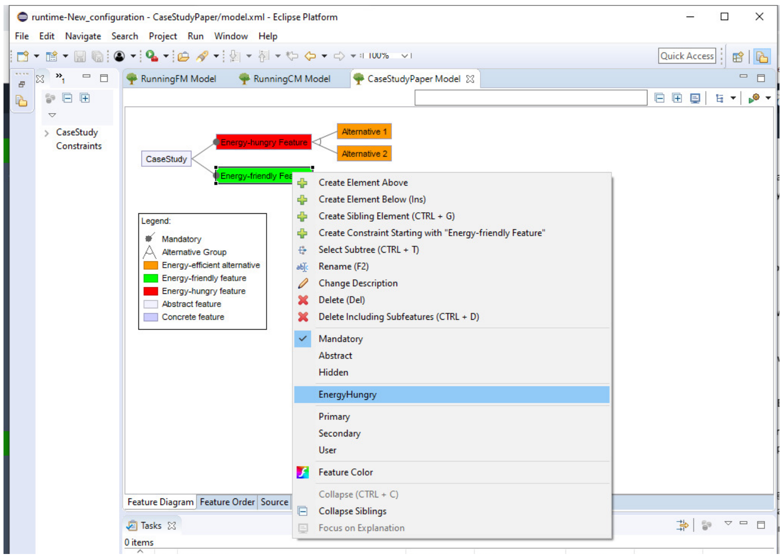


FIGURE 6.3: A snapshot of assigning energy-aware labels

a suitable label to the feature using the extensions made to the feature diagram editor as shown in Figure 6.3. In addition to specifying energy-aware labels, the diagram editor allows the domain analysts to redefine relationships such as *mandatory, optional, and alternative* among the feature and context models. Here, *mandatory* relationship is assigned to *energy-friendly features by default*. The *optional* relationship can be established between *root feature* and *energy-hungry feature* by default. However, the domain analyst can change it to *mandatory* relationship by specifying atomic *energy-efficient alternatives*. The *alternative* relationship is established between the *energy-efficient alternatives* of the energy-aware feature model.

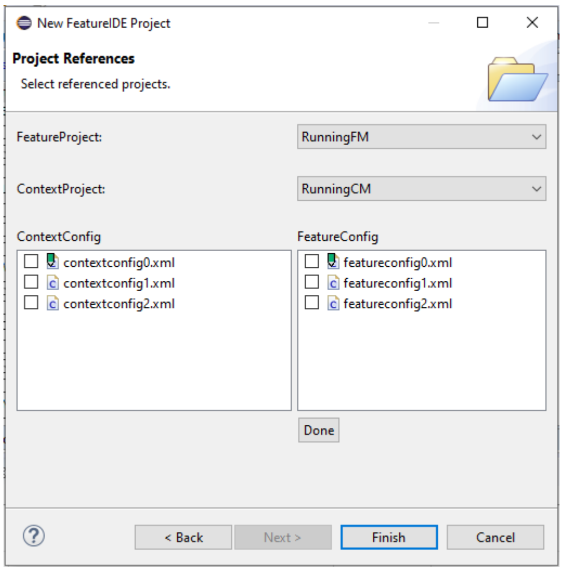
As shown in Table 6.2, *e*SAP tool consists of colored notations for specifying the energy

TABLE 6.2: Notations of energy-aware context model

Notation and Label	**Allowed Relationship(s)**	**Description**
🟨 Primary influencing context	Mandatory	Must specify its context values
🟪 Secondary influencing context	Optional, Mandatory	By default it is optional, and can be modified to mandatory if it is directly associated with primary-influencing context
🟥 User influencing context	Mandatory	By default

impact of possible context information of application under development. The leaf nodes represent the context values, and non-root and non-atomic nodes in the energy-aware context model refer to the context information. The appropriate label can be assigned to context information added using the feature diagram editor. In addition, the relationships such as *mandatory, optional, alternative* can be added between *root context, context information, and context value*. The *mandatory* relationship is allowed between the *root context* and the *primary-influencing context* information by default. On the other hand, *secondary-influencing context* information will be considered for default *optional* relationship with context root. The *context values* will be assigned with *alternative* relationship by default with it's parent. The *user-influencing context* can be considered for *mandatory* relationship by default as it represents the user preferences.

Constraint Editor: The constraint editor primarily uses the *requires* and *exclude* constraints of the basic feature model along with the Boolean expressions provided in the FeatureIDE. The domain analyst can establish constraints between the feature, context values, which has a conflicting impact on energy consumption. Similarly, constraint editor can be used to establish constraints between a group of features or context values using the (,), `AND`, `OR`, `NOT` boolean expressions. The constraint editor also ensures the validation of cross-tree constraints along with a check for mismatching brackets, redundant constraints, disallowed feature and context constraints. The feature diagram editor with the constraints added using the constraint editor produces the graphical feature diagram that represents the energy-aware models. The produced models are *energy-aware feature model* and *energy-aware context model*.


New FeatureIDE Project

Project References

Select referenced projects.

FeatureProject: RunningFM

ContextProject: RunningCM

ContextConfig
- ☐ contextconfig0.xml
- ☐ contextconfig1.xml
- ☐ contextconfig2.xml

FeatureConfig
- ☐ featureconfig0.xml
- ☐ featureconfig1.xml
- ☐ featureconfig2.xml

FIGURE 6.4: A screenshot of energy-saving adaptation plan editor

Configuration Generator: The configuration generator has been created by extending the configuration editor interface of FeatureIDE. This allows the domain analyst to select or deselect the context and feature added in the energy-aware feature diagrams. This also checks that the selection of context and feature follows the rules assigned to relationships and cross-tree constraints. Using this interface, the domain analyst can generate valid triggering situations and energy-aware applications, which later will be used for planning self-adaptation. This interface allows the domain analyst to generate more than one configurations as an XML file that can be stored inside the *configurations* folder of the project workspace. Finally, the generated configurations can be validated against the energy-aware relationships and constraints.

Energy-aware adaptation model planner: The energy-aware adaptation model editor is one of the additional project types added to the basic FeatureIDE tool. The domain analyst can select the *adaptation modeling* type in the project manager while creating a new project. The adaptation model editor requires configurations generated from energy-aware feature model and an energy-aware context model to create the adaptation plans. As shown in Figure 6.4, on creating the *adaptation modeling* project, the project manager shows an interface to import the *context model* and *feature model* through an drop-down interface. The project explorer browses through all the active projects and populates them under *feature project* and *context project* field of adaptation plan editor.

The domain-expert can select the feature models under analysis for creating the adaptation plan. The other pre-requisite to create adaptation plans is to create VTS and EAC from the appropriate models. After selecting the *feature project* and *context project*, the adaptation model editor browse through the configuration folder and populates all the generated VTS and EAC for the selected feature models. The left side pane shown in Figure 6.4 lists out the available valid triggering situation as context configurations. Similarly, the right side pane shown in Figure 6.4 lists out the energy-aware application configurations. The checkbox added in front of all listed configurations enables the domain analyst to select the needed configuration to create the adaptation plan. One configuration from the left side pane and right side pane can be selected at a time, which refers to mapping the VTS to EAC. The domain analyst can select such pairs for all the available configurations. Finally, on clicking *Done* button, the adaptation plans will be generated as an XML file in the project folder.

6.3 Validation with Case Study

A case study has been conducted for specifying energy-saving self-adaptive requirements for map navigation application to show the efficacy of *e*SAP. The guidelines suggested by Easterbrook et al. (2008) have been followed to conduct this case study. Therefore, the case study has been designed to answer the following research questions (RQs):

- **RQ1**: How to introduce energy-related requirements for the family of map navigation applications?
- **RQ2**: How to create an energy-saving adaptation plan for the family of map navigation applications?
- **RQ3**: What is the impact of *e*SAP on modeling energy-aware requirements of map navigation applications?

The case study aims at answering the research questions mentioned-above by applying *e*SAP tool on a subject application domain. The case study has been conducted for

TABLE 6.3: MapNav's functional requirements

Name	**Description**
`GetCurrentLocation`	Fetches the current location of the user using GPS
`DisplayMap`	Displays map using Satellite, Terrain and Street Views
`ShowDirections`	Shows the walking, cycling, and driving directions
`PerformNavigation`	Helps the user to navigate with turn-by-turn directions
`PerformReroute`	Reroutes the user if he/she deviated from the originally suggested direction
`UpdateDistance`	Calculates and updates the estimated distance and estimated time of arrival
`ShowTrafficinfo`	Shows live traffic information during cycling and driving
`ShowPublicTransport`	Shows available public transports from a source to destination for any given day and time
`VoiceAssistance`	Gives voice commands during turn-by-turn navigation

the domain of map navigation applications by analyzing the features of famous map navigation applications.

6.3.1 An example scenario: Reference requirements

Consider the example of Map Navigation application (MapNav) for Android. The application requirements are inspired by popular map navigation applications such as Google Maps, OpenStreetMap, and Maps.me. MapNav assists the users in finding *walking, cycling, and driving* directions from any source to destination. It also shows turn-by-turn directions to the destination from user's current location. Let us assume that *MapNav* application will be developed to satisfy the requirements listed in Table 6.3. The energy impact analysis of given functional requirements are presented in this sub section. `GetCurrentLocation` demands high accuracy as it sense the user's current location using GPS. However, user might not need high accuracy all the time, and this requirement must be extended with GPS-alternative options also. `DisplayMap` displays the map and it may consume more energy for downloading the data from map provider and to render the images. Therefore, this requirement is energy-hungry and must be light-weight on slow internet connection as it might take longer to get downloaded. `ShowDirections` needs data connection to download the information from map provider. Also, it involves adding extra lines on the map to show the directions. `PerformNavigation` depends on the following requirements: `DisplayMap`, `PerformReroute`, `UpdateDistance`, and `VoiceAssistance`. `PerformReroute` requires computation and mobile data to reroute the user, if he/she deviates from the originally suggested route. As this requirement needs continuous location monitoring, it may consume too much of energy by polling the GPS continuously. `UpdateDistance` calculates the distance between the users current location and the destination during the turn-by-turn navigation. This requirements also needs continuous location monitoring and computation to calculate the distance. `VoiceAssistance` provides the voice navigation to the user while driving a car. This is an additional feature to the on-map turn-by-turn navigation which requires the access to audio resources of the smartphone device. It can be considered as energy-hungry requirements as it performs the text-to-speech conversation. `ShowTrafficinfo` needs the connection to map provider as it has to download the live traffic data over internet. `ShowPublicTransport` also needs the connection to map provider to provide recent public transport available from the source to destination.

Overall, the given requirements require *continuous GPS sensing, monitoring user movement, connection to the internet, audio, and displaying map*. All these activities are considered to be energy-hungry activities when the battery level is low. Specifically, during the battery discharge, these energy-hungry activities must be dealt with it's energy-efficient alternatives. Therefore, switching from the energy-hungry components to it's alternatives becomes essential to reduce battery consumption.

6.3.2 Identifying energy-aware application configurations

As mentioned earlier, the majority of the reference requirements are energy-hungry as they require continuous location-sensing. However, few of the requirements could be made energy-efficient and delivered to the end-user to satisfy the given requirements. Therefore, the domain analyst has identified the energy-efficient alternative for each energy-hungry reference requirement and produced the energy-aware feature model, as shown in Figure 6.5 using *e*SAP tool. As shown in Figure 6.5, the following features (Equations 6.1 and 6.2) are added to the reference requirements to find out various energy-efficient versions suitable for contextual changes:

$$\mathbf{F} = \{\mathrm{F}_{\texttt{LocationSensing}}, \mathrm{F}_{\texttt{SensingInterval}}, \mathrm{F}_{\texttt{Directions}}, \mathrm{F}_{\texttt{TravelType}}, \mathrm{F}_{\texttt{Guidance}}, \mathrm{F}_{\texttt{LiveTraffic}}, \mathrm{F}_{\texttt{PublicTransport}}, \mathrm{F}_{\texttt{Reroute}}\} \tag{6.1}$$

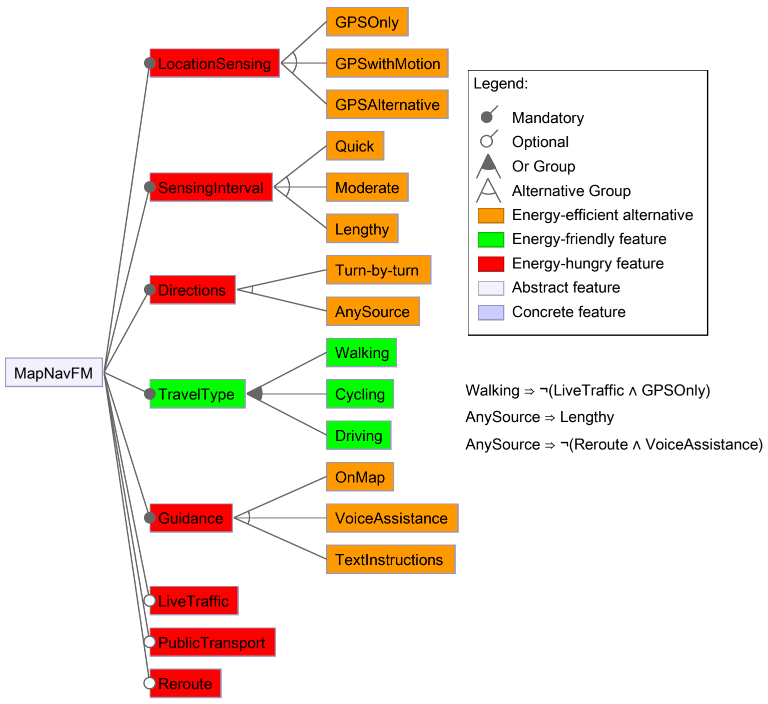


FIGURE 6.5: Energy-aware feature model of MapNav application

$$\begin{aligned}
\mathtt{F_{LocationSensing}} &= \{\mathtt{GPSOnly}, \mathtt{GPSwithMotion}, \mathtt{GPSAlternative}\}, \\
\mathtt{F_{SensingInterval}} &= \{\mathtt{Quick}, \mathtt{Moderate}, \mathtt{Lengthy}\}, \\
\mathtt{F_{Directions}} &= \{\mathtt{Turn-by-turn}, \mathtt{AnySource}\}, \\
\mathtt{F_{TravelType}} &= \{\mathtt{Walking}, \mathtt{Cycling}, \mathtt{Driving}\}, \\
\mathtt{F_{Guidance}} &= \{\mathtt{OnMap}, \mathtt{VoiceAssistance}, \mathtt{TextInstructions}\}
\end{aligned} \tag{6.2}$$

The energy impact of each feature has been analyzed and the suitable energy-aware feature labels and relationships have been assigned using *e*SAP tool. As shown in Figure 6.5, `LocationSensing` has been assigned as *energy-hungry feature* as it uses GPS for high accuracy location-sensing. Since it has been a *mandatory* feature, the following energy-efficient alternatives have been added to the energy-aware feature model: `GPSwithMotion`, `GPSAlternative`. Here, `GPSWithMotion` uses motion sensors like an accelerometer to reduce unwanted energy consumption when the user is not moving. On the other hand, `GPSAlternative` refers to location sensing using non-GPS mechanisms such as CellID, WiFi which provides less accuracy with less energy consumption. `SensingInterval` is newly introduced in the energy-aware feature model as it decides the duration of GPS requests. This feature has been considered with mandatory relationship as `Quick` sensing interval would consume abnormal energy because of frequent GPS request. Hence, its energy-efficient alternatives `Moderate`, `Lengthy` have been added to the energy-aware feature model. Here, `Lengthy` GPS sensing interval refers to GPS location requests every 5 minutes while `Moderate` GPS sensing interval could be considered every minute. These sensing intervals can be dynamically switched when there is a critical battery situation which is sensed from context information. `Directions` is considered as *energy-hungry* as it involves continuous monitoring of the user's current location along with calculating the route from the most recent location during the navigation. However, sometimes users might prefer to just look at the route without actual turn-by-turn navigation. Therefore, the energy-efficient alternative `AnySource` has been added in the energy-aware feature model. Therefore, based on the user preference, the two ways of navigation could be made adaptive to avoid unwanted location monitoring and route calculation. `TravelType` contains three sub-features, namely `Walking`, `Cycling`, `Driving` which refers to the travel type of the user. This feature is considered for *mandatory energy-friendly* as it is the core functional requirement of the *MapNav* application. As it has the *OR* relationship among the sub-features, more than one instance could exist in the application configuration at the same time. `Guidance` refers to the assistance given by the map navigation applications to the user during turn-by-turn navigation. This feature is considered as *energy-hungry* as it involves in graphics rendering (`OnMap`)and audio instructions (`VoiceAssistance`). In addition, this feature is considered for *mandatory* relationship as it is the core business requirement. In order to save energy consumption, the energy-efficient alternative, namely `TextInstructions` which does not require graphics or audio is used to guide the users. Therefore, these features can be switched at run-time whenever there is a need to save energy. `LiveTraffic`, `PublicTransport`, `Reroute` are labeled as *energy-hungry* features in the energy-aware feature model as they require continuous location sensing, route calculation, and downloading data from the internet. In addition, they add extra graphical elements on the map application to show the live traffic. Unlike other energy-hungry features, these features have been considered for *optional* relationships as they do not have energy-efficient alternatives. Therefore, the domain analyst has decided to discard these features from configurations during the battery critical situations.

As shown in Figure 6.5, few constraints are also added to illustrate the energy-saving opportunities. For instance, the constraint $\texttt{Walking} \Rightarrow \neg(\texttt{LiveTraffic} \wedge \texttt{GPSOnly})$ refers to the constraint where it states that `LiveTraffic` is invalid as the user is looking for `Walking` direction. With the help of this constraint, the data downloaded from the internet for showing live traffic could be saved. In addition, the walking takes longer time to reach the destination compared to `Driving`, and `Cycling`. Therefore, in this situation, `GPSOnly` location-sensing may drain the battery quickly as it is active for a longer duration. Therefore, `GPSOnly` can be replaced with `GPSwithMotion`, which is its energy-efficient alternative. Similarly, the constraint $\texttt{AnySource} \Rightarrow \texttt{Lengthy}$ refers to a situation where the source is manually entered by the user, and it does not require frequent sensing interval to avoid continuous GPS polling. Further, the constraint $\texttt{AnySource} \Rightarrow \neg(\texttt{Reroute} \wedge \texttt{VoiceAssistance})$ refers to the situation when the reroute and voice assistance becomes invalid if the user is only looking for directions instead of turn-by-turn navigation instructions. With the updated energy-aware feature model and constraints, a domain analyst can generate energy-aware application configurations with the help of a configuration generator. The *e*SAP tool produced a total of 1748 valid configurations for the given energy-aware feature model. For the demonstration purpose, few representative energy-aware application configurations have been picked and listed below:

- **EAC0**: $\texttt{LocationSensing}, \texttt{GPSOnly}, \texttt{SensingInterval}, \texttt{Quick}, \texttt{Directions}, \texttt{Turn}-\texttt{by}-\texttt{turn}, \texttt{TravelType}, \texttt{Cycling}, \texttt{Driving}, \texttt{Guidance}, \texttt{OnMap}, \texttt{LiveTraffic}, \texttt{PublicTransport}, \texttt{Reroute}$
- **EAC1**: $\texttt{LocationSensing}, \texttt{GPSOnly}, \texttt{SensingInterval}, \texttt{Quick}, \texttt{Directions}, \texttt{Turn}-\texttt{by}-\texttt{turn}, \texttt{TravelType}, \texttt{Cycling}, \texttt{Driving}, \texttt{Guidance}, \texttt{VoiceAssistance}, \texttt{Reroute}$
- **EAC2**: $\texttt{LocationSensing}, \texttt{GPSOnly}, \texttt{SensingInterval}, \texttt{Quick}, \texttt{Directions}, \texttt{Turn}-\texttt{by}-\texttt{turn}, \texttt{TravelType}, \texttt{Cycling}, \texttt{Driving}, \texttt{Guidance}, \texttt{OnMap}$

6.3.3 Identifying valid triggering situations

With the given functional requirements as shown in Table 6.3, the domain analyst has to identify the situations when the application requirements show negative energy impact. Let us assume that with existing domain knowledge the domain analyst has identified the context information and context values as shown in Equations 6.3 and 6.4.

$$\mathbf{C} = \{\mathtt{C}_{\mathtt{BatteryLevel}}, \mathtt{C}_{\mathtt{BatteryState}}, \mathtt{C}_{\mathtt{NetworkSpeed}}, \mathtt{C}_{\mathtt{AppState}}, \mathtt{C}_{\mathtt{SourceLoc}}\} \tag{6.3}$$

$$\begin{aligned}
\mathtt{C}_{\mathtt{BatteryLevel}} &= \{\mathtt{High}, \mathtt{Medium}, \mathtt{Low}\}, \\
\mathtt{C}_{\mathtt{BatteryState}} &= \{\mathtt{Charging}, \mathtt{Discharging}\}, \\
\mathtt{C}_{\mathtt{NetworkSpeed}} &= \{\mathtt{Fast}, \mathtt{Moderate}, \mathtt{Slow}\}, \\
\mathtt{C}_{\mathtt{AppState}} &= \{\mathtt{ForeGround}, \mathtt{BackGround}\}, \\
\mathtt{C}_{\mathtt{SourceLoc}} &= \{\mathtt{CurrentLoc}, \mathtt{AnyLoc}\}
\end{aligned} \tag{6.4}$$

To create the energy-aware context model, the energy impact of each context information and context values must be analyzed and energy-aware labels must be assigned appropriately. The context information and it's influence on the *MapNav*'s functional requirements are discussed in this sub section. `BatteryLevel` and `BatteryState` refers to the remaining available battery percentage and status of charging respectively. This information is primary source for deciding run-time energy-saving adaptation. For example, a `BatteryLevel` with *discharging* `BatteryState` would refer to battery critical situation which needs a strict energy saving plan to be enforced. `NetworkSpeed` refers to speed at which the data can be downloaded from the map provider over internet. This context information can be used to deactivate the data-intensive download activities when the device has slow internet connection. For instance, downloading data to find out `ShowTrafficinfo` and `ShowPublicTransport` would take longer time when it is executed on the slow internet connection which in-turn will consume more energy than downloading in faster internet connection. Therefore, this context information can be used to make few energy-hungry features optional at run-time. `AppState` refers to the status of the running application. It can be either foreground or background. This information can be used to select between alternatives of the energy-hungry components. For instance, if the application in the *background* is using maps for turn-by-turn instructions, it may consume energy unwantedly. Thus, instead `VoiceAssistance` could be used to reduce the energy consumed by the `DisplayMap` activity. `SourceLoc` refers to the starting location to perform the actions in *MapNav* application. The source location of the user decides the accuracy required for proving the *MapNav's* functionality. For instance, the functionality `PerformNavigation` requires high accuracy (user's current location using GPS) while `ShowDirections` requires coarse-grained accuracy (manually entered location by the user). Here, fetching current location using GPS consumes battery compared to user-entered location, which does not require GPS. Therefore, user's starting location and associated accuracy requirement can be made adaptive to avoid GPS usage when it is not required.

With this preliminary energy impact analysis, the domain analyst can assign the suitable energy-aware label suggested in the framework. The Figure 6.6 shows the energy-aware context model created by *e*SAP tool. It shows the list of context information, its values and assigned energy-aware label. As shown in Figure 6.6, the context information `BatteryLevel` and `BatteryState` are labeled as *Primary-influencing Context* as it directly associated with the energy consumption. These two context information would serve as a primary factor to trigger the run-time adaptation. Similarly, `NetworkSpeed` is labelled as *Secondary-influencing Context* as different energy-hungry features shows various energy consumption behaviour under these conditions. An `AppState` is also considered as *Secondary-influencing Context* as the switching between energy-hungry components can be triggered inline with the change in its values. Finally, the `SourceLoc` is considered as the *User-influencing Context*, as this information will be selected by the user during the application startup.

After modeling the context information, a domain analyst could use the *configuration generator* to generate and validate the *valid triggering situations*. For the given energy-aware context model, the *e*SAP produced 144 valid triggering situations. For demonstration purpose, a few significant valid triggering situations have been listed below:

- **VTS0**: `BatteryLevel`, `High`, `BatteryState`, `Charging`, `NetworkSpeed`, `Fast`, `AppState`, `ForeGround`, `SourceLoc`, `CurrentLoc`

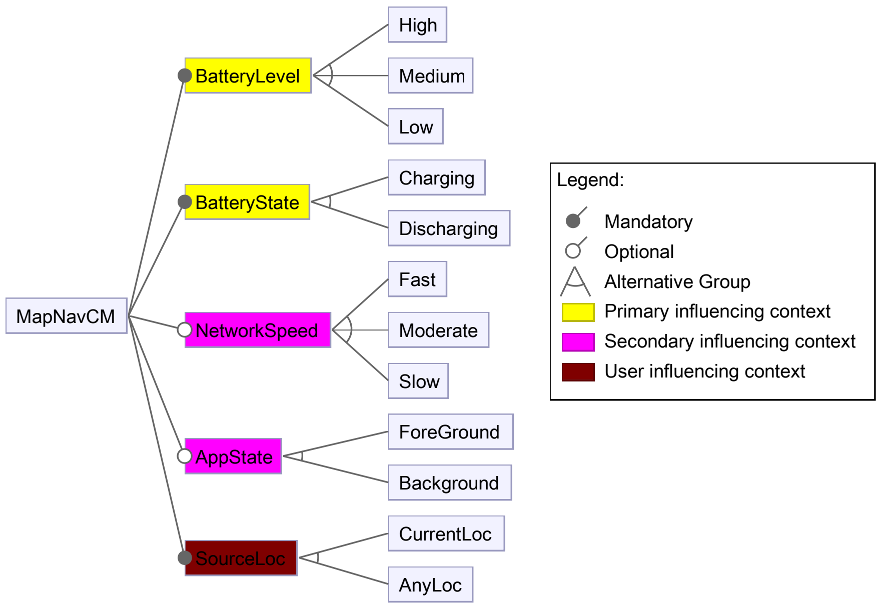


FIGURE 6.6: Energy-aware context model of MapNav application

- **VTS1**: `BatteryLevel`, `High`, `BatteryState`, `Discharging`, `NetworkSpeed`, `Fast`, `AppState`, `BackGround`, `SourceLoc`, `CurrentLoc`
- **VTS2**: `BatteryLevel`, `Medium`, `BatteryState`, `Discharging`, `NetworkSpeed`, `Moderate`, `AppState`, `ForeGround`, `SourceLoc`, `CurrentLoc`

6.3.4 Planning energy-saving adaptations

The representative energy-aware application configurations (EAC) presented in Section 6.3.2 and valid triggering situations (VTS) presented in Section 6.3.3 are considered to plan the energy-saving adaptation. By analyzing the representative EAC and representative VTS, the following energy-saving adaptation (ESA) plans are derived:

$$\begin{aligned}\mathtt{ESA}_0 &= On\ \mathtt{VTS}_0\ Execute\ \mathtt{EAC}_0,\\ \mathtt{ESA}_1 &= On\ \mathtt{VTS}_1\ Execute\ \mathtt{EAC}_1,\\ \mathtt{ESA}_2 &= On\ \mathtt{VTS}_2\ Execute\ \mathtt{EAC}_2,\end{aligned} \tag{6.5}$$

The *adaptation planner* component of the *e*SAP tool helps the domain analyst to map the corresponding VTS and EAC derived from the energy-aware context and feature model, respectively. The energy-saving adaptation plans are returned as the XML representation, as shown in Listing 6.1.

```xml
<AdaptationModel>
    <On VTS="1">
        <configuration>
            <feature automatic="selected" name="MapNavCM"/>
            <feature automatic="selected" name="BatteryLevel"/>
            <feature manual="selected" name="High"/>
            <feature automatic="unselected" name="Medium"/>
            <feature automatic="unselected" name="Low"/>
            <feature automatic="selected" name="BatteryState"/>
            <feature automatic="unselected" name="Charging"/>
            <feature manual="selected" name="Discharging"/>
            <feature automatic="selected" name="NetworkSpeed"/>
            <feature manual="selected" name="Fast"/>
            <feature automatic="unselected" name="Moderate"/>
            <feature automatic="unselected" name="Slow"/>
            <feature automatic="selected" name="AppState"/>
            <feature automatic="unselected" name="ForeGround"/>
            <feature manual="selected" name="Background"/>
            <feature automatic="selected" name="SourceLoc"/>
            <feature manual="selected" name="CurrentLoc"/>
            <feature automatic="unselected" name="AnyLoc"/>
        </configuration>
    </On>
    <Execute EAAC="1">
        <configuration>
            <feature automatic="selected" name="MapNavFM"/>
            <feature automatic="selected" name="LocationSensing"/>
            <feature manual="selected" name="GPSOnly"/>
            <feature automatic="unselected" name="GPSwithMotion"/>
            <feature automatic="unselected" name="GPSAlternative"/>
            <feature automatic="selected" name="SensingInterval"/>
            <feature manual="selected" name="Quick"/>
            <feature automatic="unselected" name="Moderate"/>
            <feature automatic="unselected" name="Lengthy"/>
            <feature automatic="selected" name="Directions"/>
            <feature automatic="selected" name="Turn-by-turn"/>
            <feature automatic="unselected" name="AnySource"/>
            <feature automatic="selected" name="TravelType"/>
            <feature manual="selected" name="Cycling"/>
            <feature manual="selected" name="Driving"/>
            <feature automatic="selected" name="Guidance"/>
            <feature automatic="unselected" name="OnMap"/>
            <feature manual="selected" name="VoiceAssistance"/>
            <feature automatic="unselected" name="TextInstructions"/>
            <feature manual="selected" name="Reroute"/>
        </configuration>
    </Execute>
</AdaptationModel>
```

LISTING 6.1: An energy-saving adaptation plan generated by *e*SAP representing ESA1

As shown in Listing 6.1, the `VTS1` represents the situation when the application is in the background, during the turn-by-turn direction while the battery state is changed to `Discharging`. As the application uses `CurrentLoc` and `BatteryLevel` is `High`, it is suitable situation to execute energy-hungry components. However, `OnMap` produces unwanted energy consumption as the application is running in the background. Therefore, it makes sense if the application is reconfigured to disable graphics rendering associated with `OnMap` to avoid energy waste. However, to guide the user, as per `ESA1`, the `VoiceAssistance` could be enabled as an alternative to satisfy the application requirements. Similar to the energy-saving adaptation plan mentioned in Listing 6.1, multiple critical battery situations can be identified in the domain analysis phase itself to provide more clarity to developers during the programming of the energy-saving self-adaptive behavior.

6.3.5 Discussions

As mentioned before, based on the guidelines given by Easterbrook et al. (2008) has been used to conduct the case study to validate the abilities of *e*SAP. The case study aimed at answering the following research questions:

- **RQ1: How to introduce energy-related requirements for the family of map navigation applications?** The energy-aware context model, energy-aware feature model, and energy-saving adaptation plan have been introduced for this purpose. The energy-aware context model classifies the context information into *primary-influencing, secondary-influencing, and user-influencing* context to show the energy impact of context information. Furthermore, the energy-aware context model helps domain analysts to identify critical battery situations (Valid Triggering Situations [VTS]) to trigger the energy-saving adaptation. Similarly, the energy-aware feature model classifies the features into *energy-friendly, energy-hungry, energy-efficient alternative* based on its impact on energy consumption. The energy-aware feature model generates multiple versions of application configurations (Energy-aware Application Configurations [EAC]) to decide the most suitable version of features for energy savings at run-time. The *project manager, diagram editor, constraints editor, configuration generator* components of *e*SAP tool helps the domain analysts to introduce energy-related requirements to the existing functional requirements.

- **RQ2: How to create an energy-saving adaptation plan for the family of map navigation applications?** The case study results show the generation of VTS and EAC from the corresponding energy-aware feature diagrams. The valid triggering situations and energy-aware application configurations are well utilized in the case study to plan energy-saving adaptation policies. The *adaptation planner* components of the *e*SAP tool facilitate the domain analyst to create the energy-saving adaptation plans in the XML format. The case study results show five sets of VTS configurations and corresponding EACs for energy-saving self-adaptation. The sample adaptation plan illustrated in Listing 6.1 is evidence for planning energy-saving adaptation of map navigation applications.

- **RQ3: What is the impact of *e*SAP on modeling energy-aware requirements of map navigation applications?** The *e*SAP produces energy-saving adaptation plans for map navigation applications, which will be used at the early stages of software development. Introducing dynamic requirements early in the life cycle of the software project will give clear information to developers. The developers can efficiently use energy-aware models and adaptation plans to develop an energy-friendly, dynamically adaptive application. Further, the artifacts produced by the *e*SAP tool might reduce the efforts spent at the development phase.

To the best of our knowledge, a *sufficient* contribution has been made to the modeling and analyzing energy-aware context, features, and adaptation. The tool presented in this chapter might aid domain analysts to apply an energy-aware modeling framework on any other application domain, which has energy as one of the important software quality. To summarize, the following contributions have been made through the development of *e*SAP:

1. Introduction of graphical notations to group the context and feature based on their energy impact;

2. A tool support to aid the domain analyst to model and validate energy-aware requirements and energy-saving self-adaptation.

The *e*SAP uses feature diagrams for modeling energy-aware context information and energy-aware features. The feature diagrams are widely used modeling methods to identify the commonality and variability in the software systems. Therefore, it is believed that using feature models would be a suitable technique for identifying various energy-aware application configurations. The energy-saving adaptation planning presented in this chapter would help understand the uncertainty and battery critical situations before developing the application. The case study shows that the energy-aware labels can be easily derived from energy impact analysis. The valid triggering situation and energy-aware application configurations provide clear information on planning energy-saving adaptations. The artifacts produced by *e*SAP tool are platform-independent and are generic for all application domain. Thus, the energy-aware modeling framework and *e*SAP can be re-used for other applications domain as well to introduce energy-awareness and self-adaptivity.

6.4 Summary

This chapter introduces the tool support for specifying energy-related requirements in the early stages of software development. The contribution of this objective is to provide a tool support, namely, Energy-saving Adaptation Planner (*e*SAP) for the concepts presented in the energy-aware modeling framework. The popular feature modeling tool *FeatureIDE* (Thüm et al., 2014) has been modified with the following extensions: *project manager, diagram editor, configuration editor, and energy-saving adaptation planner* to develop *e*SAP. The *e*SAP tool produces the following artifacts, namely *energy-aware context model, energy-aware feature model, and energy-saving adaptation plans*. Further, *e*SAP aids the domain analyst to generate *valid triggering situations* and *energy-aware application configurations* in order to produce XML-based energy-saving adaptation plans. The efficacy of *e*SAP is shown with the help of a case study focusing on the map navigation application. The presented approach and tool support may help the domain analyst plan energy-saving adaptation before implementing the application. In addition, the case study results show that this approach would serve as a starting point in considering energy-efficiency and self-adaptivity as critical factors during software development.

Chapter 7

ENERGY-SAVING CODE GENERATOR

The tool developed in Research Objective III (eSAP) aids the domain analyst create domain-specific models that are platform-independent. The models generated by eSAP gives the developers conceptual clarity for writing code that is self-adaptive for energy-saving. However, eSAP cannot generate source code from the generated models to aid the developers in the development phase. Therefore, this objective aims to develop a domain-specific language and corresponding code generator, namely eGEN, to aid the developer in the development phase. In this objective, textual domain-specific language and automatic code generator concepts have been adopted to generate domain-dependent models and platform-specific source code. In Section 7.1, the domain analysis of location-based applications is presented to identify the features to be included in eGEN. This section also presents the way of identifying reusable components that can be added to the code generator. The actual tool description and language grammar are explained in Section 7.2. Finally, the validation of eGEN with the case study is presented in Section 7.3.

Previously, the developers' community used automatic code generation tools to help new developers who face commonly occurring problems. As the adaptive location-sensing is common for all location-based applications, there is a need to assist the developers in introducing self-adaptive behaviors for energy savings. Therefore, in this objective, the aim is to develope a tool to help developers by automatically generating and suggesting the Android Java code, which can be added to the existing Android repositories.

7.1 Domain Analysis

Domain analysis about the location-based smartphone applications has been conducted and possible ways of introducing model-driven development solutions have been researched. The goal of introducing self-adaptive behavior for reducing battery consumption has been our primary importance during the domain analysis. Primarily, the focus is on reducing GPS usage for reducing battery consumption while maintaining accuracy-related requirements. The domain analysis has been conducted from two perspectives. The first perspective is about identifying a suitable methodology for balancing energy-accuracy requirements of location-based applications. Therefore, literature and developer documents have been explored to get domain knowledge about reducing location-based applications' energy consumption. The second perspective is about considering the location-based applications as self-adaptive software to find out the common features among all location-based applications. Here, the findings would be beneficial while designing the domain-specific language and code generator to have included the artifacts for all common features.

7.1.1 Identifying self-adaptive location strategies

There are several energy-efficient location strategies available in the literature for location-based applications at development time. This sub-section presents some significant research works towards reducing the energy consumption of location-sensing.

Alternatives to GPS

Under this category, the approaches completely replace the GPS-based positioning with other energy-efficient alternative techniques such as WiFi, Cell ID, etc. One such approach uses Cell-ID sequencing matching (Paek et al., 2011) for energy-efficient positioning on the smartphones. The authors have proposed CAPS, a Cell-ID Aided Positioning System that aims to improve the accuracy of the cell tower-based approach while keeping the energy overhead low. CAPS identifies the user's current position cell-ID information and the past GPS coordinates. The authors have claimed that the proposed approach provides accuracy comparable to GPS positioning with minimum energy consumption. GSM positioning system, CellSense (Ibrahim and Youssef, 2012) is proposed for GSM-based cell phones with a probabilistic fingerprinting. Cellsense also provides a hybrid approach that uses probabilistic and deterministic techniques to achieve high accuracy with less computation overhead. Another system, Placemap (Yadav et al., 2014) is proposed to discover different places and routes with the help of GSM information. This approach uses the initial WiFi-based positioning for improving location accuracy. WiFi-based positioning system consumes comparatively more energy than the GSM or cell ID based techniques. The research work by Choi et al. (2017) aims at reducing the energy consumption of WiFi scanning by predicting the optimal number of scanned access points. WiFi-based positioning system requires the dense deployment of access points to provide location accuracy, which is an extra overhead. To address this issue, SAIL was proposed by Mariakakis et al. (2014), which uses a single WiFi access point with the help of precise dead-reckoning using smartphone motion sensors. Overall, these approaches promising energy savings compared to GPs techniques; on the other hand, failed to give high accuracy.

Combining movement detection

The continuous GPS based positioning systems failed because of the unwanted GPS scans when the user is not moving. The recent research works focus on reducing the number of unwanted GPS scans by identifying the user's movement with the built-in smartphone sensors. Kjærgaard et al. (2009) proposed EnTracked, a system for energy-efficient and robust position tracking. This system estimates the system condition and mobility to schedule the location updates effectively. Another approach, rate-adaptive positioning system (RAPS) (Paek et al., 2010) aims to turn off the GPS when it is not available. The RAPS system uses a duty-cycled accelerometer, Bluetooth communication to estimate the user movement. In addition, RAPS uses the cell tower-RSS blacklisting to detect if the GPS signal is available or not. To support the continuous location determination, Höpfner and Schirmer (2012) uses the mobility profile of the user with the help of an accelerometer. Similar to postponing the GPS positioning, PlaceWalker (Cho et al., 2015) postpones the WiFi positioning by monitoring the user activity with the accelerometer's help. In general, compass, gyroscope, barometer, and other inertial sensors are also used for movement detection. An important disadvantage with these techniques are not all smartphones are equipped with inertial sensors.

Collaborative strategies

Fetching location from other location-based applications or neighboring devices can be used to reduce the number of GPS requests found in several research works. One such framework (Man and Ngai, 2014) aims to coordinate with multiple location-based applications to reduce energy consumption on GPS location updates further. In this approach, the GPS location received by the application can be reused among the other applications in a short period. In another approach, the aggregator and collector groups are used for location-sensing (Xi et al., 2015). Here, the collector group uses aggregator groups' location, which turns on their GPS periodically to determine the location.

Adaptive strategies

The adaptive strategies dynamically select a suitable location strategy based on dynamic accuracy and energy requirements. Zhuang et al. (2010), proposed an adaptive location-sensing framework which uses substitution, suppression, piggybacking, and adaptation. Here, *substitution* uses the alternatives to GPS; *suppression* low-power intensive sensors such as accelerometer; *piggybacking* uses the collaborative strategies; and *adaptation* adjust the sensing interval to save energy at run-time. In this approach, all the other previous approaches are used in an adaptive way to improve the energy and accuracy requirements. A-Loc (Lin et al., 2010), automatically manages location sensor availability, accuracy, and energy by selecting the most suitable location sensing mechanism at run-time. Another approach (Morillo et al., 2012), dynamically adapts between GPS, WiFi-based localization, and accelerometer at run-time based on the user's outdoor exit detection to improve the GPS efficiency. In Virtual GPS (Thokala et al., 2014), a middleware chooses a suitable location strategy (GPS or WiFi or Cell ID) for a giver accuracy requirement with minimum energy consumption. Kim et al. (2016b) use the context information such as a category of applications executed, the user's movement pattern, and the battery level to select the suitable location detection scheme (GPS or Cell-tower). In recent research work, Capurso et al. (2017) proposed indoor-outdoor detection techniques to switch between GPS and other indoor-localization methods to improve the energy efficiency of the location-based applications. In addition to academic research works, the official Google Location APIs has continuously evolved to improve location-sensing energy-efficiency. For instance, Google location APIs such as Fused Location Provider API[1], FusedLocationProviderClient[2] are evolving in every release of API with new energy-saving methods.

In summary, the *alternative strategy, sensor-based strategy, and collaborative strategy* relies on a static model that does not balance energy-accuracy requirements at run-time. Since users move from indoor to outdoor, a static model may fail to serve both environments. In such a situation, adaptive strategies provide better results by switching between suitable location strategies based on the user context such as indoor-outdoor, dynamic accuracy-energy requirements. The previously discussed self-adaptive location-sensing strategies are development time solutions to tackle GPS-based applications' energy consumption issues. In general, the solutions attempt to change the application behavior dynamically in response to the change in context and user preferences. These solutions suggest ways to improve coding practice directly at the source code level. The software development cost might increase if the self-adaptive location-sensing decisions are taken during development. Therefore, it is essential to abstract the energy-saving decisions to the design-time before making decisions on lower-level code development. To the best of our knowledge, there is a lack of tool support for specifying location-based

[1]https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderApi

[2]https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient

Android applications' self-adaptive location-sensing strategies. Therefore, the aim is to develop a tool, *e*GEN, to consider energy-related requirements at the design phase.

7.1.2 Identifying commonalities and variabilities

As a second perspective, considered location-based applications have been considered as Software Product-lines (SPL) to find out the common and variable features. A Software Product-line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission developed from a common set of core assets in a prescribed way (Clements and Northrop, 2002). Software product lines are widely used in the industries to provide a set of reusable assets for the software developers to reduce the development efforts (Van Gurp et al., 2001). The commonality among similar products is captured to develop reusable assets in software product lines. The software developers can focus more on product-specific aspects rather than common aspects. The map navigation application also can be realized as a software product line as it has lots of commonality among the family of location-based applications. This sub-section presents an initial exploration of product families of general map navigation application with a running example to show the SPL-based solution's suitability. Here, common features will be considered for designing the domain-specific language and code generator. The variable features will be considered as business requirements and will not be included in the domain-specific language and code generator.

Scoping of map navigation applications

In general, all navigation applications may consist of *identifying the users location, accessibility of the application, viewing the places and directions on the map, directions instructions, additional features such as saving, printing, and sharing*. Here, reusable assets or features of the map navigation application are analyzed.

Location sensing

Location sensing is the process of identifying the user's current location with the smartphone's capabilities.The user's current position can be identified by *GPS, WiFi, Cellular ID, Collaborative methods*. GPS is most suitable in outdoor situations and for more accuracy. WiFi is mostly ideal for indoor scenarios and sometimes for Urban areas where there are many WiFi access points. The Cellular ID is also preferred sometimes as an alternative to WiFi-based positioning. The collaborative location sharing between the location-based application of a smartphone is also used to reduce the unnecessary usage of location sources. The smartphone inertial sensors such as accelerometer, compass, gyroscopes, barometer, proximity sensors, microphone, and camera sensors are also used in combination with GPS to improve the location sensing efficiency. For any map navigation application, location sensing is the common mandatory feature added by default. On the other hand, many location-sensing mechanisms are available; each has variable capabilities with respect to Energy consumption, accuracy, etc. In addition, instead of using the current location, the user can select any location manually to see the directions or navigation options. Therefore, location sensing is the most common feature which can be reused in different navigation applications.

Accessibility

Providing accessibility to the user is an essential aspect of location-based applications. In general, most location-based applications provide accessibility to the user with or without an internet connection. The online maps require internet connectivity, whereas offline maps do not require internet connectivity.

Map views

Viewing the maps on the user's smartphone is an essential feature of the map navigation applications. Many navigation applications provide different types of views to users. The most commonly used views are normal view, satellite view, terrain view. And recently, street views of the urban areas have also been displayed in the maps. Similar to location sensing, the map view also a mandatory feature that must be available in all the map navigation applications. Therefore, the different map views can be reused in the various versions of the map navigation applications.

Directions

The important feature of map navigation applications is providing directions between two locations. There are two different aspects of the directions feature. The first one is the type of directions provided; the second one is the display type. The different categories under direction types are: walking directions for pedestrians, cycling directions for cyclers, driving directions for cars, and public transport information for everyone. Displaying the direction is the default feature of the map navigation application. The next important aspect is displaying the directions from source to destination. The directions can be displayed as text, or it can be displayed on the maps. Further, the map application can show the go navigation information (turn-by-turn) based on the user's current location with the help of GPS.

Extra features

In addition to the mandatory features mentioned before, some maps also provide some extra features like saving the maps or directions for offline use, printing the map or directions, and sharing the location or directions through messages, emails, or social networks. This capability can be considered an optional feature as it is not the essential feature of a map navigation application.

Initial exploration of possible product categories

Product Configuration 1 (Street View) This product configuration can produce a set of applications that are targeted for people to find out the street view of the new place to be visited. This product contains the *GPS, and Manual* location sensing techniques that provide only online access to the user. This is a fundamental application derived from the Table 7.1.

TABLE 7.1: Initial exploration of possible product categories (incomplete list)

Category	Features	PL1	PL2	PL3	PL4	PL5
Location Sensing	GPS (GP)	×	×	×	×	×
	WiFi (WF)					
	Cell ID (CL)					×
	Manual (MN)	×	×	×	×	×
Access	Online (ON)	×	×	×	×	×
	Offline (OF)					
Maps Views	Normal View (NV)		×	×	×	×
	Satellite View (SV)		×	×	×	
	Terrain View (TV)					
	Street View (SrV)	×				
Directions Type	Walking (WK)			×	×	
	Cycling (CY)			×	×	
	Driving (DR)		×		×	
	Live Traffic (LT)		×			
	Public Trans. (PT)					×
	Schedule (SL)					×
Directions Display	Text (TX)			×	×	×
	On Map (OM)		×	×	×	×
	On the Go (OG)				×	
Extra	Save (SE)					×
	Print (PR)					×
	Share (SH)					

Product Configuration 2 (Live Traffic) This product configuration can produce a set of applications which provides the live traffic on the roads and are used by the car drivers. Similar to *product configuration 1*, this configuration also uses *GPS, and Manual* for location sensing and provides online access. Whereas it does not provide a street view, instead it provides the normal and satellite view on the maps. It also provides the driving directions with live traffic data that will be marked on the map. This configuration has the commonality in location sensing technique and variants in map view compared to previous application configurations. It has extra features in showing directions. The applications that are developed using this configuration will be useful for car drivers.

Product Configuration 3 (Maps with Directions) This product configurations can produce a set of applications which provide the directions from source to destination. This product configuration has the commonality in location sensing and accessibility as it uses *GPS, Manual and Online*. Compared to product configuration 2, this product configuration has the commonality with map views as it provides *Normal and Satellite view*. With respect to directions, this product configuration varies in terms of *direction type and direction display* as it provides *walking, cycling directions* with *text and on the map* instructions. The applications derived from these product configurations will be useful to pedestrians and cyclers who use GPS enables devices.

Product Configuration 4 (GPS Navigation) This product configuration can produce a set of applications that has on the go directions for walking, cycling, and driving. This configuration has commonality with *product configuration 2 and 3* in terms of *location sensing, map views*. Similar to *product configuration 3*, this configuration also provides walking, cycling directions, and ability to provide driving directions. In addition, it has the additional feature of showing the directions by fetching the user's current location on the map. The applications from these product configurations can be useful for pedestrians, cyclers, and car drivers.

Product Configuration 5 (Public transport) This product configuration can produce a set of applications that produces the schedule for public transport. In terms of location sensing, this application can get the user's location from either GPS or Cellular ID or Manually. This product configuration provides online connectivity to the user. As a new feature, it gives the ability to save, share, and print the location information. This application can satisfy the needs of users looking for public transport options.

From the family of products derived from the running example, the following can be sketched:

- **Commonality**: The map navigation application contains some common features like location sensing, map views, direction type, etc. Suitable artifacts can be developed to use different map navigation applications in the family when it has the common requirements. Finally, the implementation of the software artifacts will result in common reusable components.

- **Reusable variation**: As shown in the running example, by using the reusable components of the navigation application, many different applications can be developed for targetting different application requirements.

- **Product specifics**: The product specific features need not be included in the product lines. For instance, more variable features can be added to list nearby locations for travelers or find out the hotels or petrol pumps as product-specific features separately. Therefore, variable applications can be developed using the basic set of reusable product line components.

With this brief analysis, it can be concluded that the map navigation application can be considered as an SPL as it exhibits several reusable features and product-specific features. From the domain-analysis, it can be inferred that location-sensing is a common feature that exists in all types of location-based applications. Therefore, the model and the generated location-sensing code using *e*GEN can be used in all kinds of location-based applications to reduce the development efforts. Therefore, the aim is to provide domain-specific language and code generator for common reusable features in this approach.

7.1.3 *e*GEN domain model

As shown in Listing 7.1, the domain model contains the entities `AdaptationPolicy`, `Context`, `Feature`. The `AdaptationPolicy` refers to the self-adaptive energy-saving location-sensing strategies. It can be defined with the combination of `Context` and `Feature`. The entity `Context` refers to the contextual situation suitable for enabling energy-saving self-adaptation. The context information primarily includes `BatteryState`, `BatteryLevel`, and `ApplicationState`. The `BatteryLevel` refers to the remaining battery percentage of the smartphone. The `BatteryState` refers to the charging or discharging status of the smartphone. The `ApplicationState` refers to the background or foreground execution of the application.

```
entity AdaptationPolicy {
        PolicyID
        Context
        Adaptation
        }
entity Context {
        BatteryState = (Charging | Discharging)
        BatteryLevel = (Low | Medium | High)
        ApplicationState (ForeGround | Background)
        }
entity Feature {
        SensingInterval
        DecreasingFactor
        BatteryAwareFunction
        }
```

LISTING 7.1: Domain model for modeling location-sensing

The entity `Feature` includes the following: `SensingInterval`, `DecreasingFactor`, and `BatteryAwareFunction`. The `SensingInterval` refers to the time difference between two subsequent location-sensing requests. The `DecreasingFactor` refers to the numerical value that will be used to calculate the sensing interval for each battery drop in the exponential battery-aware function. The `BatteryAwareFunction` refers to the type of change (exponential or linear) in fixing the sensing interval. This domain model is used as a basis for defining the grammar of the DSML, which is part of *e*GEN.

7.2 *e*GEN Description

The basic idea of energy-saving location-sensing is to enforce energy-saving policies in the following situations: (1) when the battery is discharging, and the battery level is critical, (2) when the app is in the background. Therefore, *e*GEN is designed to give domain analysts options to assign values for *critical battery level, and sensing-interval*

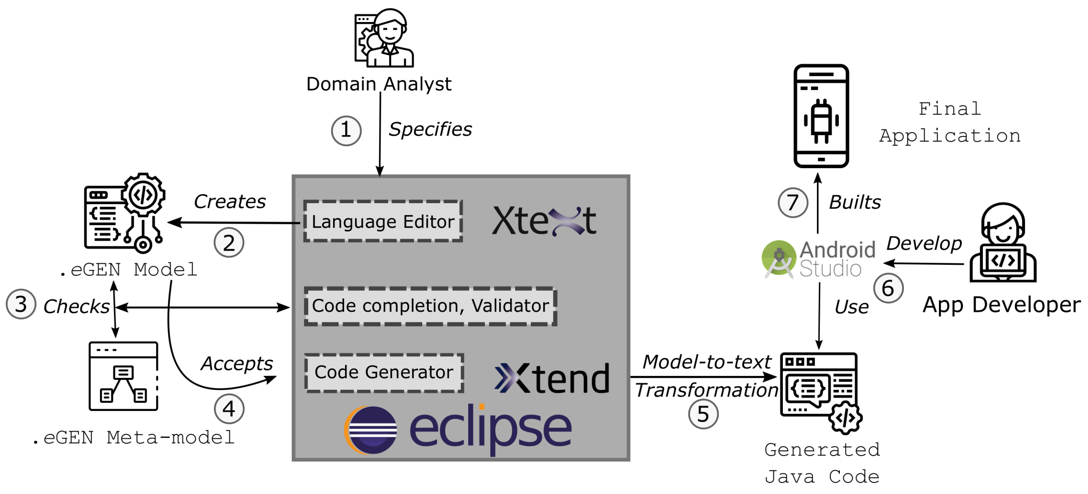


FIGURE 7.1: *e*GEN eco-system

based on the application requirements. A domain-specific modeling language, and automatic code generation tool, namely *e*GEN has been developed, with the help of *Xtext and Xtend* (Behrens et al., 2008), an Eclipse-based language development framework. As shown in Figure 7.1, the uses of *e*GEN consists of seven steps. In the first step, the *domain analyst* use the Eclipse editor to specify the energy-saving location-sensing using the textual domain-specific modeling language. In the second step, the editor creates the .egen model. In the third step, the *validator* module of *Xtext* checks the .egen model whether it is following the *e*GEN meta-model defined in DSML. In the fourth step, the code generator takes the validated .egen model as the input and generates the Java code using model-to-text transformation in the fifth step. Here, the code generator converts the textual .egen model to a native Android code. The generated code contains the self-adaptive location-sensing combined with battery-aware code. In the sixth step, the developer could add the generated code to existing Android applications to make it energy-aware. Finally, in the seventh step, the updated Android project can be built and installed on the Android device by the user. The DSML enables the definition of context and run-time re-configurations. The adaptation policies written in *e*GEN are more concise, easier to maintain, can be written quickly.

7.2.1 Technical background

A domain-specific language (DSL) and automatic code generation tool, namely *e*GEN has been developed with the help of Xtext and Xtend, an Eclipse-based language development framework. Xtext covers all aspects of language infrastructure including a parser, linker, compiler, interpreter, and IDE support. Especially, Xtext with Eclipse provides syntax coloring and code completion features to make the job easier for the developers. Additionally, Xtext offers validation, code generation, customization, and many more features. This makes Xtext a suitable language development framework for developing domain-specific languages. The developed DSL specification is defined in grammar like EBNF form with the help of Xtext. The self-adaptation policies can be created with the file extension *.egen* by using the *e*GEN tool. The DSL enables the definition of context and run-time re-configurations. The adaptation policies written in *e*GEN are more concise, easier to maintain, can be written quickly. The *e*GEN tool uses the following Android APIs to achieve self-adaptation: *Battery Manager API, Fused Location API, and Android Activity Life Cycle*. Here, *Battery Manager API* provides a method for querying battery and charging properties; Fused Location API provides the ability to control location-sensing; and *Android Activity Life Cycle* helps to identify the status of the Android activity. The source code of *e*GEN is available in GitHub[3] for use.

7.2.2 *e*GEN grammar

This subsection describes the *e*GEN grammar along with the structure of the language elements such as *features, context, and adaptation policy*. The defined grammar is easy to understand, and it allows even a non-programmer to model the textual adaptation policy for energy-aware self-adaptive location-sensing. In addition, the DSL grammar is platform-independent, and it can be applied to other platforms as well.

[3] https://github.com/marimuthuc/egen

Allowed features

The *feature* element of *e*GEN is used to define the application requirements that affect the battery consumption of smartphone devices. Especially in location-based applications, the features like *location-sensing interval, type of change in sensing interval* play a significant role in deciding the self-adaptive location-sensing strategies. Therefore, in *e*GEN, the values for `Features` that are allowed during the definition of features have been fixed, as shown in 7.2.

```
Features:
    SensingInterval | Decreasing_Factor | BatteryAwareFunction;
```

LISTING 7.2: Structure of the features values

As shown in Listing 7.2, *e*GEN allows following `Features` for specifying energy-aware requirements:

- `SensingInterval` refers to the time difference between two subsequent location-sensing requests.
- `DecreasingFactor` refers to the numerical value that will be used to calculate the sensing interval for each battery drop in the exponential battery-aware function.
- `BatteryAwareFunction` refers to the type of change (exponential or linear) in fixing the sensing interval.

Feature definition

Each `Feature` can have their own rules for defining the corresponding values as shown in Listing 7.3. The rules for defining feature values are given below:

- The definition of location-sensing interval starts with the keyword "`SensingInterval`" and can be assigned with an `integer` value (refer $lines1 - 2$ in listing 7.3). Here, the "`SensingInterval`" must be assigned in milliseconds.

- The definition of decreasing factor starts with the keyword "`DecreasingFactor`" and can be assigned with the `integer` value (refer *lines3* − *4* in listing 7.3) as decided by the domain analyst.
- The definition of a type of battery-aware function starts with defining the value for keyword "`BatteryAwareFunction`" and can be assigned with one of the following fixed values: `linear`, `exponential` (refer *lines5* − *6* in listing 7.3).

```
1 SensingInterval:
2     'SensingInterval' '=' ivalue = MYINT_T;
3 DecreasingFactor:
4     'DecreasingFactor' '=' ivalue=MYINT_T;
5 BatteryAwareFunction:
6     'BatteryAwareFunction' '=' value=('Exponential' | 'Linear')
     ;
```

LISTING 7.3: Structure of feature definition

Allowed context

The `Context` element of *e*GEN is used to define the valid situations to enforce self-adaptive energy-saving policies of smartphone applications. For location-based applications, the following context is considered in *e*GEN: *remaining battery percentage, charging state of the device, and state of the application.*

```
Context:
    BatteryState | BatteryLevel | AppState | Threshold_Medium |
    Threshold_High
```

LISTING 7.4: Structure of the context values

As shown in Listing 7.4, *e*GEN allows following `Context` for specifying energy-saving situations:

- `BatteryState` refers to the charging or discharging status of the smartphone.

- `BatteryLevel` refers to the remaining battery percentage of the smartphone. Further, the context `Threshold_High` and `Threshold_Medium` is used to define the maximum and medium battery percentage for triggering self-adaptive behavior.

- `ApplicationState` refers to the background or foreground execution of the application.

Context definition

According to *e*GEN grammar, the definition of each allowed context can have its predefined values, and domain analyst defined values.

```
1  BatteryState:
2      'BatteryState' '=' value=('Charging' | 'Discharging');
3  BatteryLevel:
4      'BatteryLevel' '=' value=('High' | 'Medium' | 'Low');
5  Threshold_High:
6      'Threshold_High' '=' ivalue=MYINT_T;
7  Threshold_Medium:
8      'Threshold_Medium' '=' ivalue=MYINT_T;
9  AppState:
10     'AppState' '=' value=('Foreground' | 'Background')
```

LISTING 7.5: Structure of the context constraints

As shown in Listing 7.5, the rules for specifying context values are given below:

- The definition of charging state of the device starts with the keyword "`BatteryState`" and can have one of the following values: `Charging`, `Discharging` (refer $lines1 - 2$ in listing 7.5).

- The definition of remaining battery level starts with the keyword "`BatteryLevel`" and can have any one of the following values: `High`, `Medium`, `Low` (refer $lines3 - 4$ in listing 7.5). Domain analyst can assign the values for `High` and `Medium` as an integer value based on the application requirements (refer $lines5 - 8$ in listing

7.5). The value `Low` will be inferred by the code generator script based on the range given for `High` and `Medium`.

- The definition of application execution state starts with the keyword "`AppState`" and can have any one of the following values: `Foreground`, `Background` (refer $lines9 - 10$ in listing 7.5).

Adaptation policy

According to *e*GEN grammar, the definition of adaptation policy consists of assigning *five* contexts and *three* features. The specification of a self-adaptive location-sensing policy can have one or more entries differentiated with a unique ID.

```
1  Model:
2      eGEN += AdaptationPolicy*;
3  AdaptationPolicy:
4      'AdaptationPolicy' ivalue=MYINT_T '{' 'Condition' '{'
     Situation1 = Context 'AND' value=(Situation2);
5  Situation2:
6      Block = Situation3 '}' 'then' 'Adaptation' '{'
     FeatureBlock1 = Features 'AND' value=(FeatureBlock2)  '}' '}
     ';
7  Situation3:
8      Context = Context 'AND' value=(Situation4);
9  Situation4:
10     Context = Context 'AND' value=(Situation5);
11 Situation5:
12     Context = Context 'AND' value=(Context);
13 FeatureBlock2:
14     Feature2 = Features 'AND' value=(Features);
```

LISTING 7.6: Structure of the adaptation policy

As shown in Listing 7.6, a single adaptation policy definition contains following parts:

- starts with the keyword `AdaptationPolicy` to describe an adaptation policy followed by the unique ID of type `integer` (refer $lines3 - 4$ in listing 7.6).

- an opening brace for adaptation policy definition
- a keyword `Condition` to describe the allowed context changes that trigger the self-adaptation
- an opening brace for context block definition
- five condition definition, each consists of context assigned with allowed values. The description of context values is given in Listing 7.5. Here, the context definition can be in any order. However, the repetition of context information is not allowed inside the same context block.
- multiple valid contexts can be separated by the keyword `Condition`.
- a closing brace for context block
- a keyword `Adaptation` to describe the corresponding set of features to be executed at run-time for the contextual changes described with the keyword `Condition`.
- an opening brace for a feature block
- three feature definition, each consists of features assigned with allowed values. The description of feature definition is given in Listing 7.3. Here, the feature definition can be in any order. However, the repetition of feature information is not allowed inside the same feature block.
- multiple adaptations can be separated by the keyword `Condition`.
- a closing brace for a feature block
- Finally, a closing brace for adaptation policy

7.2.3 Energy-aware code generator

The code generator is defined by mapping each element in the *e*GEN DSML to a corresponding Android library. The $BatteryManager$ class is used to fetch the $BatteryState$, $BatteryLevel$ of the Android device. The application's status is identified using the Android activity lifecycle $onStart()$, $onResume()$, $onPause()$. The $SensingInterval$,

$DecreasingFactor$, $BatteryAwareFunction$ elements are mapped to the location request block of *FusedLocationProviderClient* API. Especially, the $BatteryAwareFunction$ uses the *battery percentage* fetched from battery manager and *decreasing factor* mentioned by domain analyst in the script to calculate the dynamic sensing interval. The code generator creates following Java files that contains the artifacts for the self-adaptive location-sensing: $MainActivity.Java$, $LocationUtility.Java$, $BatteryAware.Java$, and $AdaptationUtility.java$. $BatteryAware.java$ is the file that does the adaptive location-sensing activity. MainActivity extends the $BatteryAware$ activity and fetches the location coordinates from the function $onLocationUpdate()$ defined in the batteryAware class. The application developers can modify $MainActivity.java$ to write their business logic. The file $AdaptationUtility.java$ contains the code that alters the sensing-interval based on the context provided. The battery state defined in the *e*GEN model is verified against the charging status obtained from the $BatteryManager$ API. $LocationUtility.java$ contains the code that does the location fetching activity as per the sensing-interval captured in the $AdaptationUtility.java$. The generated code can be appended to the existing Android projects to make their app self-adaptive for location-sensing.

7.3 Validation

The code generated by the *e*GEN tool is validated through controlled experiments. The variables such as *percentage of battery drop* and *location accuracy* is used for validating the tool. The code generator has been redefined till it produces the source code that reduces battery consumption. In addition, the code has been defined to make sure that it balances accuracy requirements with acceptable degradation. Two different subject applications with the same functional requirements have been developed to validate the code generated by *e*GEN. The subject application functionalities are given below:

- **Location fetching**: This functionality fetches the location coordinates of the user device from GPS. The Fused-location provider-client API is used to fetch location in the subject applications.

- **Location logging**: This functionality sends the location coordinates to the Firebase database periodically. The time difference between the subsequent location-sensing will be decided by the sensing-interval defined by the developer.
- **Calculating Distance**: This function calculates the distance covered by the user with the help of GPS logs from the FireBase database. In addition, the application would have a option to stop the location logging.

The two versions of subject applications have been developed as follows: (1) without code from *e*GEN and (2) with code generated from *e*GEN. The first version of the subject application has static sensing-interval and is considered too energy-hungry. The sensing interval doesn't change even if the device is running on a low battery. On the other hand, the second subject application has been developed by applying code generated from *e*GEN, which will work based on dynamically adaptive location-sensing strategies. An excerpt of the adaptation policy is shown in Listing 7.7. The generated Java code has been added to the first subject application to make it self-adaptive. As a result, it produced the second subject application.

```
AdaptationPolicy 01 {
    Condition {
        BatteryState = Discharging  AND
        BatteryLevel = Low AND
        Threshold_High = 63 AND
        Threshold_Medium = 46 AND
        AppState = Background
    } then
    Adaptation {
        SensingInterval = 250 AND
        Decreasing_Factor = 20 AND
        BatteryAwareFunction = Linear
    }
}
```

LISTING 7.7: An Excerpt from the egen model of subject application 2

As shown in Listing 7.7, one of the self-adaptive location-sensing policies would be to execute an energy-efficient version when the device is running with a low-battery and

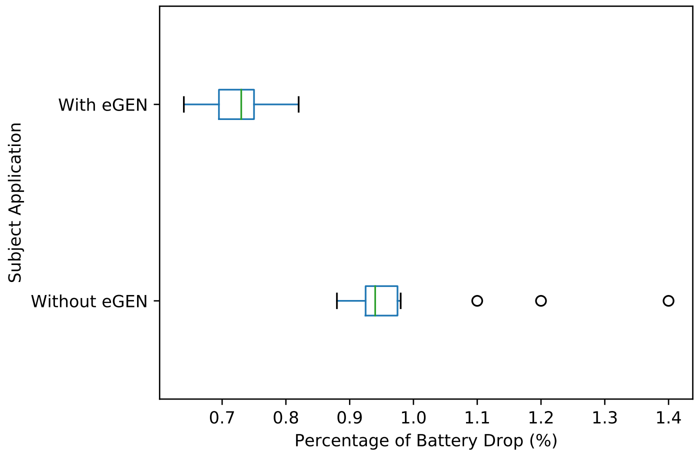


FIGURE 7.2: Improvement in battery saving

in a discharging state. Similar to this adaptation policy, several other policies have been defined for the second subject application.

The controlled experiments have been performed with fifteen trials for each subject application on Redmi Note 8 running Android 10 to see the improvement in *battery consumption* and *accuracy*. The `BatteryConsumption` has been calculated using the *Google Battery Historian* tool in terms of `PercentageofBatteryDrop`. The `Accuracy` has been calculated as the `DistanceCovered` using the GPS logs in meters. The output of the subject application *without the generated code* and *with the generated code* have been analyzed. The results of Battery consumption are shown in Figure 7.2, and the accuracy (distance covered) is shown in Figure 7.3. As observed from Figure 7.2, the inter-quartile range of *percentage of battery drop* produced by the first subject application (without *e*GEN) is between 0.9% to 1.0%. On the other hand, the inter-quartile range of *percentage of battery drop* produced by the second subject application (with *e*GEN) is between 0.7% to 0.8%. The results show that the code generated by *e*GEN produce battery-aware code that consumes comparatively less battery for its operation.

With respect to *Accuracy* parameter, as shown in Figure 7.3, the inter-quartile range of *distance covered* produced by the first subject application (without *e*GEN) is between 505 meters to 513 meters. In contrast, the inter-quartile range of *distance covered* produced by the second subject application (with *e*GEN) is between 492 meters to 497 meters. As a result, the second application shows an inaccurate location with 13 meters

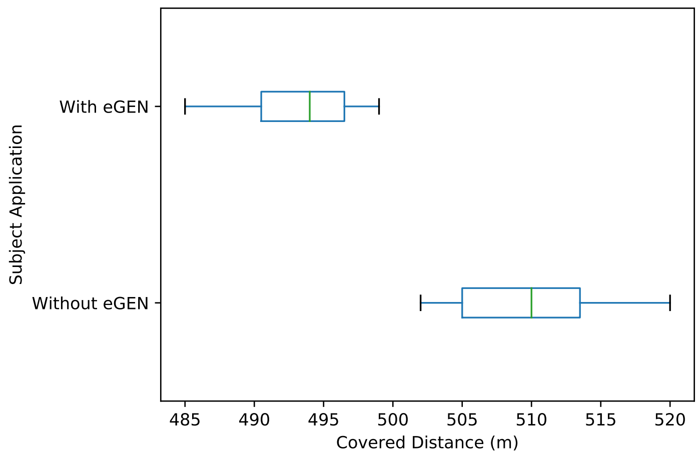


FIGURE 7.3: Degrade in accuracy

to 16 meters difference which is negligible when the second subject application reduces the percentage of battery drop. Therefore, from the results, it is empirically evident that the code generated by *e*GEN can balance between battery and accuracy requirements with acceptable level of degradation.

7.4 Summary

This chapter presents a description of *e*GEN tool for modeling energy-aware self-adaptive behaviors of location-based mobile applications. The primary contribution of this objective is *e*GEN tool which aids the domain analyst and developers in designing and developing battery-aware code for energy-saving location-sensing. *e*GEN is developed using *Xtext and Xtend* as an Eclipse plugin. *e*GEN currently supports the Android platform and covers the *battery manager API, fused location provider client, and Android activity life cycle*. The case study shows that *e*GEN provides the ability to specify self-adaptive location-sensing strategies before development. The subject application developed using the generated code shows less battery consumption with negligible degradation in accuracy. The primary target audience of *e*GEN tool are the domain analysts and developers. The domain analyst would use the tool for specifying the energy-saving adaptation plans, and the developer would use the generated code in the existing Android applications for handling location-based services.

Chapter 8

CONCLUSIONS AND FUTURE WORKS

This chapter of the thesis concludes our work by giving a summary of the preceding chapters and describing some of the future work directions.

8.1 Thesis Summary

The thesis starts with introducing the motivation, running example, and research objectives of this research work. Then, in Chapter 2, it presents the following background information relevant to this thesis: *smartphone energy inefficiencies, energy-aware self-adaptive software, existing modeling frameworks for self-adaptive software, feature-oriented software development, and model-driven development*. In Chapter 3, a detailed summary of relevant research has been presented under *empirical studies, post-development approaches, and pre-development approaches*. The research gaps highlighted at the end of Chapter 3 positions this research in the early stages of software development. The remaining chapters of this thesis present the respective contributions for each research objective. The research questions introduced in the motivation section of Chapter 1 considered throughout this thesis to find out appropriate answers. Here, the research questions listed and identified answers are discussed to recapitulate the thesis contribution:

- **RQ1**: What solutions do developers apply to reduce the energy consumption of commonly used energy-hungry components?
  The empirical studies in software engineering have been used to find out the energy-saving solutions, and the answer is presented in Chapter 4. Three types of empirical studies have been conducted, which include *controlled experiments, qualitative data analysis of Stack Overflow data, and quantitative data analysis of GitHub data*. The controlled experiments results show that the popular location-APIs use inertial sensors to reduce Android apps' battery consumption provided the sensors are used only when needed. The qualitative analysis of Stack Overflow discussions about location-based Android applications' energy consumption helps to identify the energy-saving solutions suggested by the developers. Overall, the expert developers recommend using self-adaptive location-sensing along with the battery manager. Analyzing GitHub commits shows that developers have applied energy-saving solutions after deployment and it resulted in more development efforts. Hence, introducing energy-saving solutions early would help developers get more clarity on energy-saving before development to reduce development efforts.

- **RQ2**: Is it possible to consider energy-efficiency as an essential non-functional requirement in the early stages of software development?
  Yes, it is possible to introduce energy-efficiency as an essential non-functional requirement in the early stages, and the proof of concept is presented in Chapter 5. In literature, there exist several modeling frameworks to model self-adaptive behavior as a non-functional requirement. But none of the existing modeling frameworks can explicitly specify the energy-related requirements. Hence, in Chapter 5, a conceptual modeling framework for energy-aware modeling with self-adaptation planning is discussed. The energy-aware modeling framework answer this research question by categorizing the requirements into the following models: *energy-aware context model, energy-aware feature model, and energy-saving adaptation plan*. The popular feature-oriented domain analysis framework has been adopted with suitable extensions to introduce formalism in energy-aware

modeling. The *energy-aware context model* classifies the requirements into the following based on its impact on energy consumption: *primary-influencing context, secondary-influencing context, and user-influencing context*. The *energy-aware feature model* classifies the requirements into the following based on its energy consumption: *energy-hungry features, energy-efficient alternatives, and energy-friendly features*. The configurations generated from *energy-aware context model and energy-aware feature model* could be combined to plan energy-saving adaptation plans. The case study presented in Chapter 5 shows the efficacy of the energy-aware modeling framework. The case study results show that it is possible to introduce energy-related requirements along with self-adaptive requirements in the early stages with the help of an energy-aware modeling framework.

- **RQ3**: What is necessary to support domain analysts with suitable tool support to consider energy-efficiency in the early stages?
  The answer to this research question is presented in Chapter 6. The conceptual framework presented in Chapter 5 is a manual feature modeling process and has no tool support. Therefore, it has been decided to find available options to provide tool support for the energy-aware modeling framework. After a thorough search in the literature, the popular feature modeling tool feature IDE is selected to develop tool support as it is an open-source tool. As a result, the extension points of FeatureIDE have been studied and analyzed to introduce concepts presented in the energy-aware modeling framework. Finally, three project types have been added to FeatureIDE's project manager. It includes *energy-aware context modeling, energy-aware feature modeling, and energy-saving adaptation modeling*. A colored notations representing the energy-aware labels have been added to the FeatureIDE code base under each model type. Finally, the XML adaptation model generator has been added to the FeatureIDE code base to help model energy-saving adaptation plans. The developed tool is named as energy-saving Adaptation Planner (*e*SAP) and is freely available for use and contribution in GitHub[1]. The case study shows that *e*SAP contains all necessary components to model the energy-aware self-adaptive behavior.

[1]https://github.com/marimuthuc/esap

- **RQ4**: Is it possible to support developers by transforming design-time models to source code for reusable components to reduce the development efforts?
  Yes, it is possible to generate source code for battery-aware location-sensing from the design-time artifacts. In Chapter 7, the energy-saving Code Generator (*e*GEN) is discussed with a textual domain-specific modeling language and code generator. The textual domain-specific language has been designed after conducting the domain analysis on popular location-based applications to ensure it covers all the modeling elements. Domain-specific modeling has been developed using the Xtext tool. The model elements are concise and easy to understand to the domain analyst which help introduce better-awareness in the early stages. Further, a corresponding code generator has been developed to generate native Java code, including battery manager API and Fused Location Provider Client API. The code generator tool *e*GEN is developed as an Eclipse plugin and freely available for use in GitHub[2]. The experimental results show that the code generation of the battery-aware source code is possible for location-sensing. The subject applications show that the generated code could be added to existing Android applications to make them battery-aware. The experimental results show that the generated code balances between energy and accuracy requirements of location-based Android applications.

In summary, this thesis contributes the following to the research community:

- Energy-saving usage patterns of location-based Android applications
- Research efforts needed for fixing energy-related issues after deployment in open source applications
- An energy-aware modeling framework for energy-aware self-adaptive software
- An energy-saving adaptation planner for energy-aware self-adaptive software
- An energy-saving code generator for location-based applications

[2]https://github.com/marimuthuc/egen

8.2 Conclusion

The results of empirical studies (Objective 1) show that the expert developers suggest self-adaptive location-sensing strategies to reduce battery consumption. However, analyzing GitHub commits implies that the introduction of energy-aware self-adaptive behavior after deployment would increase the development efforts. Therefore, the decision to contribute tool support to the early stages of software development has been taken to carry out the remaining research activities. The case study conducted to show the efficacy of the energy-aware modeling framework (Objective 2) shows that self-adaptive behavior would be a potential solution for solving the smartphone's energy consumption issues. It also highlights the importance of classifying the context and features based on their impact on battery consumption. However, the conceptual framework lacks tool support and providing tool would help the domain analyst. Therefore, to consider the self-adaptive energy-savings in the early stages, tool support *e*SAP has been developed as part of research objective 3, which will be used during the domain analysis phase. This tool facilitates the domain analyst to analyze the possibility of introducing self-adaptive energy-saving strategies to the application under development. The tool produces the following artifacts: *energy-aware feature model, energy-aware context model, and energy-aware adaptation model*. This tool has been developed on top of the popular feature modeling tool, FeatureIDE. The case study and results suggest that the energy-aware modeling framework and the *e*SAP tool would serve as the starting point in the research area of specifying and analyzing the energy-saving self-adaptive requirements in the early stage of software development. Finally, for research objective 4, a code generator, namely *eGEN* has been developed that automatically generates the battery-aware code for energy-saving adaptive location-sensing. The generated code could be added to the existing location-based Android applications to make it energy-aware self-adaptive software. The results show that the subject application with generated code can balance battery consumption and accuracy-related requirements. Overall, the presented methodology would aid the domain analyst to enumerate energy-saving opportunities in the early stages. The artifact produced by *e*SAP would help the developers to make energy-saving decisions at the code level. Further, the code generated from *e*GEN would reduce the development efforts to introduce energy-aware self-adaptive behavior for the developers.

8.3 Future Work

The energy-aware modeling framework, *e*SAP and *e*GEN would serve as the starting point in the research area of specifying and analyzing the energy-saving self-adaptive requirements in the early stage of software development. However, there are several improvements to the developed tools that can be carried out in the future, as listed below:

- The first potential future work would be the addition of quantitative aspects of energy-related requirements to the models produced by *e*SAP. The energy-aware modeling framework focuses only on the qualitative aspects of the energy-related requirements. Adding quantitative aspects could make the models more realistic.
- The negative impact of energy-savings is not explicitly considered in the current version of *e*SAP tool. In the future, a module could be added to consider the trade-off between other affected software quality factors caused by strict energy-saving adaptations.
- The preliminary version of *e*GEN supports only Android platform, and in the future, support to iOS platform could also be considered. In addition, location-based features such as activity recognition and inertial sensors like accelerometer and magnetometer could also be added in the next version. Furthermore, the developed tool could be evaluated with Mobile app developers for its usability and completeness by conducting industrial case studies.
- The *e*GEN tool is restricted only to the Android application's location-sensing. In future, the *e*GEN can be extended to cover other energy-hungry components such as screen, CPU, GPU, and Network.

Appendix A

PUBLICATIONS

Journal Publications

Published

1. **Marimuthu C**, K. Chandrasekaran, Sridhar Chimalakonda. (2020). "Energy Diagnosis of Android Applications: A Thematic Taxonomy and Survey." *ACM Computing Surveys*, ACM, 53 (6), Article 117, 36 Pages, DOI: https://doi.org/10.1145/3417986

2. **Marimuthu C**, Sridhar Chimalakonda, K. Chandrasekaran. (2020). "How do open source app developers perceive API changes related to Android battery optimization? An empirical study." *Journal of Software: Practice and Experience*, Wiley, p. 1-20, DOI: https://doi.org/10.1002/spe.2928

3. **Marimuthu Chinnakali**, Sanjana Palisetti, K. Chandrasekaran. (2020). "Organizing the knowledge from Stack Overflow about location-sensing of Android applications." *IET Software*. IET Digital Library, 14 (3), p. 221-233, DOI: https://doi.org/10.1049/iet-sen.2019.0284

Under Review

1. **Marimuthu C**, Sridhar Chimalakonda, and K. Chandrasekaran., "An Energy-aware Modeling Framework for Self-adaptive Smartphone Applications." *Journal of Systems and Software*. Elsevier.

Conference Publications

Published

1. **Marimuthu C**, Sanjana Palisetti, K. Chandrasekaran. (2019). "An empirical study on managing energy and accuracy requirements of location based android applications". In Proceedings of *the 31st International Conference on Software Engineering and Knowledge Engineering (SEKE 2019)*, Lisbon, Portugal (pages. 553-556). DOI: https://doi.org/10.18293/SEKE2019-179

2. **Marimuthu C**, K. Chandrasekaran. (2017). "Software engineering aspects of green and sustainable software: A systematic mapping study". In Proceedings of *the 10th Innovations in Software Engineering Conference (ISEC 2017)*, ACM, Jaipur, India (Pages 34-44). DOI: https://doi.org/10.1145/3021460.3021464

3. **Marimuthu C**, K. Chandrasekaran. (2016). "Feature-Oriented Domain Analysis Framework for Energy-Aware Self-Adaptive Software". In Proceedings of *the IEEE Green Computing and Communications (GreenCom)*, IEEE, Chengdu, China (Pages 773-776). DOI: https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData.2016.163

Under Review

1. **Marimuthu C**, Kowndinya Boyalakuntla, Sridhar Chimalakonda, and K. Chandrasekaran., "*e*GEN: An Energy-saving Modeling Language and Code Generator for Location-sensing of Mobile Apps." Submitted to *36th IEEE/ACM International Conference on Automated Software Engineering (ASE 2021)*.

Appendix B

BIO-DATA

Name	:	Marimuthu C
Email Id	:	muthucwc.seopro@gmail.com
Date of Birth	:	May 12, 1990
Address	:	56A, Kottamedu, Mettur Road, Mecheri, Salem, Tamilnadu, PIN: 636 453.

Educational Qualifications:

Degree	Year of Passing	University
Ph.D., (CSE)	2015 - Pursuing	National Institute of Technology Karnataka, Surathkal.
M.Tech., (CSE)	2015	National Institute of Technology Karnataka, Surathkal.
B.E., (CSE)	2007	Sona College of Technology, Salem.

Short Bio and Research Interest:

Marimuthu C is a Ph.D. student in the Department of Computer Science and Engineering at National Institute of Technology Karnataka, Surathkal . He is working under the supervision of Prof. K. Chandrasekaran for his Ph.D. degree in the area of "Model-driven development of Energy-aware Self-adaptive Software". He also works in the field of Empirical Software Engineering and Mining Software Repositories. Specifically, he works with data available on StackOverflow and GitHub to summarize the existing knowledge of developers and their perspectives on Non-functional requirements.

His research interest includes Software Engineering, particularly Energy-aware Software, Empirical Software Engineering, Mining Software Repositories, Machine Learning for Software Engineering, Modeling of Self-adaptive Software, Domain-specific Languages.

Bibliography

Abbasi, A. M., Al-Tekreeti, M., Naik, K., Nayak, A., Srivastava, P., and Zaman, M. (2018). "Characterization and detection of tail energy bugs in smartphones." *IEEE Access*, 6, 65098–65108.

Acher, M., Collet, P., Lahire, P., and France, R. B. (2013). "Familiar: A domain-specific language for large scale management of feature models." *Science of Computer Programming*, 78(6), 657–681.

Ahmad, M., Bruel, J.-M., Laleau, R., and Gnaho, C. (2012). "Using relax, sysml and kaos for ambient systems requirements modeling." *Procedia Computer Science*, 10, 474–481.

Ahmad, R. W., Gani, A., Hamid, S. H. A., Xia, F., and Shiraz, M. (2015). "A review on mobile application energy profiling: Taxonomy, state-of-the-art, and open research issues." *Journal of Network and Computer Applications*, 58, 42–59.

Ahmed, S. and Bagherzadeh, M. (2018). "What do concurrency developers ask about?: a large-scale study using stack overflow." *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ACM, 30.

Antkiewicz, M. and Czarnecki, K. (2004). "Featureplugin: feature modeling plug-in for eclipse." *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, ACM, 67–72.

Apel, S. and Kästner, C. (2009). "An overview of feature-oriented software development.." *J. Object Technol.*, 8(5), 49–84.

Arnold, M., Vechev, M., and Yahav, E. (2011). “Qvm: An efficient runtime for detecting defects in deployed systems.” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 21(1), 2.

Asadi, M., Soltani, S., Gasevic, D., Hatala, M., and Bagheri, E. (2014). “Toward automated feature model configuration with optimizing non-functional requirements.” *Information and Software Technology*, 56(9), 1144–1165.

Atkinson, C. and Kuhne, T. (2003). “Model-driven development: a metamodeling foundation.” *IEEE software*, 20(5), 36–41.

Baddour, A. M., Sang, J., Hu, H., Akbar, M. A., Loulou, H., Ali, A., and Gulzar, K. (2019). “Cim-css: A formal modeling approach to context identification and management for intelligent context-sensitive systems.” *IEEE Access*, 7, 116056–116077.

Banerjee, A., Chong, L. K., Ballabriga, C., and Roychoudhury, A. (2018). “Energypatch: Repairing resource leaks to improve energy-efficiency of android apps.” *IEEE Transactions on Software Engineering*, 44(5), 470–490.

Banerjee, A., Guo, H.-F., and Roychoudhury, A. (2016). “Debugging energy-efficiency related field failures in mobile apps.” *Proceedings of the International Conference on Mobile Software Engineering and Systems*, ACM, 127–138.

Banerjee, A. and Roychoudhury, A. (2016). “Automated re-factoring of android apps to enhance energy-efficiency.” *Mobile Software Engineering and Systems (MOBILESoft), 2016 IEEE/ACM International Conference on*, IEEE, 139–150.

Bao, L., Lo, D., Xia, X., Wang, X., and Tian, C. (2016). “How android app developers manage power consumption?: An empirical study by mining power management commits.” *Proceedings of the 13th International Conference on Mining Software Repositories*, ACM, 37–48.

Bao, T., Zheng, Y., and Zhang, X. (2012). “White box sampling in uncertain data processing enabled by program analysis.” *ACM SIGPLAN Notices*, Vol. 47, ACM, 897–914.

Barnett, S., Avazpour, I., Vasa, R., and Grundy, J. (2019). “Supporting multi-view development for mobile applications.” *Journal of Computer Languages*, 51, 88–96.

Batory, D. (2005). "Feature models, grammars, and propositional formulas." *Proceedings of the 9th International Conference on Software Product Lines*, SPLC'05, Berlin, Heidelberg, Springer-Verlag, 7–20.

Behrens, H., Clay, M., Efftinge, S., Eysholdt, M., Friese, P., Köhnlein, J., Wannheden, K., and Zarnekow, S. (2008). "Xtext user guide." *Dostupné z WWW: http://www. eclipse. org/Xtext/documentation/1_0_1/xtext. html*, 7.

Bellas, N., Hajj, I., Polychronopoulos, C., and Stamoulis, G. (2000). "Architectural and compiler techniques for energy reduction in high-performance microprocessors." *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(3), 317–326.

Benavides, D., Segura, S., and Ruiz-Cortés, A. (2010). "Automated analysis of feature models 20 years later: A literature review." *Information systems*, 35(6), 615–636.

Benavides, D., Segura, S., Trinidad, P., and Cortés, A. R. (2007). "Fama: Tooling a framework for the automated analysis of feature models.." *VaMoS*, 2007, 01.

Beverungen, D., Breidbach, C. F., Poeppelbuss, J., and Tuunainen, V. K. (2019). "Smart service systems: An interdisciplinary perspective.." *Inf. Syst. J.*, 29(6), 1201–1206.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). "Latent dirichlet allocation." *Journal of machine Learning research*, 3(Jan), 993–1022.

Brambilla, M., Cabot, J., and Wimmer, M. (2017). "Model-driven software engineering in practice." *Synthesis lectures on software engineering*, 3(1), 1–207.

Braun, V. and Clarke, V. (2013). *Successful qualitative research: A practical guide for beginners*. sage.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2004). "Tropos: An agent-oriented software development methodology." *Autonomous Agents and Multi-Agent Systems*, 8(3), 203–236.

Brouwers, N., Zuniga, M., and Langendoen, K. (2014). "Neat: a novel energy analysis toolkit for free-roaming smartphones." *Proceedings of the 12th ACM conference on embedded network sensor systems*, ACM, 16–30.

Brown, G., Cheng, B. H., Goldsby, H., and Zhang, J. (2006). “Goal-oriented specification of adaptation requirements engineering in adaptive systems.” *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, ACM, 23–29.

Calefato, F., Lanubile, F., and Novielli, N. (2018). “How to ask for technical help? evidence-based guidelines for writing questions on stack overflow.” *Information and Software Technology*, 94, 186–207.

Capurso, N., Mei, B., Song, T., Cheng, X., and Yu, J. (2018). “A survey on key fields of context awareness for mobile devices.” *Journal of Network and Computer Applications*, 118, 44–60.

Capurso, N., Song, T., Cheng, W., Yu, J., and Cheng, X. (2017). “An android-based mechanism for energy efficient localization depending on indoor/outdoor context.” *IEEE Internet of Things Journal*, 4(2), 299–307.

Carette, A., Younes, M. A. A., Hecht, G., Moha, N., and Rouvoy, R. (2017). “Investigating the energy impact of android smells.” *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, 115–126.

Carroll, A. and Heiser, G. (2010). “An analysis of power consumption in a smartphone.” *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC’10, Berkeley, CA, USA, USENIX Association, 21–21, <http://dl.acm.org/citation.cfm?id=1855840.1855861>.

Cañete, A., Horcas, J.-M., Ayala, I., and Fuentes, L. (2020). “Energy efficient adaptation engines for android applications.” *Information and Software Technology*, 118, 106220.

Ceri, S., Daniel, F., Matera, M., and Facca, F. M. (2007). “Model-driven development of context-aware web applications.” *ACM Transactions on Internet Technology (TOIT)*, 7(1), 2–es.

Chen, G. and Kotz, D. (2000). “A survey of context-aware mobile computing research.” *Report No. 867843*, Hanover, NH, USA.

Cheng, B. H., De Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., et al. (2009a). "Software engineering for self-adaptive systems: A research roadmap." *Software engineering for self-adaptive systems*, Springer, 1–26.

Cheng, B. H., Sawyer, P., Bencomo, N., and Whittle, J. (2009b). "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty." *International Conference on Model Driven Engineering Languages and Systems*, Springer, 468–483.

Cho, D.-K., Lee, U., Noh, Y., Park, T., and Song, J. (2015). "Placewalker: An energy-efficient place logging method that considers kinematics of normal human walking." *Pervasive and Mobile Computing*, 19, 24–36.

Choi, T., Chon, Y., and Cha, H. (2017). "Energy-efficient wifi scanning for localization." *Pervasive and Mobile Computing*, 37, 124–138.

Chowdhury, S., Di Nardo, S., Hindle, A., and Jiang, Z. M. J. (2018). "An exploratory study on assessing the energy impact of logging on android applications." *Empirical Software Engineering*, 23(3), 1422–1456.

Classen, A., Boucher, Q., and Heymans, P. (2011). "A text-based approach to feature modelling: Syntax and semantics of tvl." *Science of Computer Programming*, 76(12), 1130–1143.

Clements, P. and Northrop, L. (2002). *Software product lines*. Addison-Wesley,.

Clements, P. C. and Northrop, L. (2001). *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley.

Cruz, L. and Abreu, R. (2018). "Using Automatic Refactoring to Improve Energy Efficiency of Android Apps." *Proceedings of the XXI Iberoamerican Conference on Software Engineering*, 163–176.

Cruz, L., Abreu, R., and Rouvignac, J.-N. (2017). "Leafactor: Improving energy efficiency of android apps via automatic refactoring." *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, IEEE, 205–206.

Czarnecki, K. and Eisenecker, U. W. (2000). *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

Dao, T. A., Singh, I., Madhyastha, H. V., Krishnamurthy, S. V., Cao, G., and Mohapatra, P. (2017). "Tide: A user-centric tool for identifying energy hungry applications on smartphones." *IEEE/ACM Transactions on Networking*, 25(3), 1459–1474.

Dardenne, A., Van Lamsweerde, A., and Fickas, S. (1993). "Goal-directed requirements acquisition." *Science of computer programming*, 20(1-2), 3–50.

Datta, S. K., Bonnet, C., and Nikaein, N. (2014). "Self-adaptive battery and context aware mobile application development." *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, IEEE, 761–766.

De Lemos, R., Giese, H., Müller, H. A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N. M., Vogel, T., et al. (2013). "Software engineering for self-adaptive systems: A second research roadmap." *Software Engineering for Self-Adaptive Systems II*, Springer, 1–32.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). "Indexing by latent semantic analysis." *Journal of the American society for information science*, 41(6), 391–407.

Delaluz, V., Kandemir, M., Vijaykrishnan, N., and Irwin, M. J. (2000). "Energy-oriented compiler optimizations for partitioned memory architectures." *Proceedings of the 2000 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '00, New York, NY, USA, ACM, 138–147, <http://doi.acm.org/10.1145/354880.354900>.

Desmet, B., Vallejos, J., Costanza, P., De Meuter, W., and D'Hondt, T. (2007). "Context-oriented domain analysis." *International and Interdisciplinary Conference on Modeling and Using Context*, Springer, 178–191.

Dey, A. K. (2001). "Understanding and using context." *Personal Ubiquitous Comput.*, 5(1), 4–7.

Dong, M. and Zhong, L. (2011). “Self-constructive high-rate system energy modeling for battery-powered mobile systems.” *Proceedings of the 9th international conference on Mobile systems, applications, and services*, ACM, 335–348.

dos Santos, E. B., Andrade, R. M., and de Sousa Santos, I. (2021). “Runtime testing of context-aware variability in adaptive systems.” *Information and Software Technology*, 131, 106482.

Duhoux, B., Dumas, B., Leung, H. S., and Mens, K. (2019). “Dynamic visualisation of features and contexts for context-oriented programmers.” *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 1–6.

Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D. (2008). “Selecting empirical methods for software engineering research.” *Guide to advanced empirical software engineering*, 285–311.

Eric, S. Y. (2009). “Social modeling and i.” *Conceptual modeling: Foundations and applications*, Springer, 99–121.

France, R. B., Ghosh, S., Dinh-Trong, T., and Solberg, A. (2006). “Model-driven development using uml 2.0: promises and pitfalls.” *Computer*, 39(2), 59–66.

Franch, X., López, L., Cares, C., and Colomer, D. (2016). “The i* framework for goal-oriented modeling.” *Domain-specific conceptual modeling*, Springer, 485–506.

Frank, U. (2013). “Domain-specific modeling languages: requirements analysis and design guidelines.” *Domain Engineering*, Springer, 133–157.

Georgiou, S., Rizou, S., and Spinellis, D. (2019). “Software development lifecycle for energy efficiency: Techniques and tools.” *ACM Comput. Surv.*, 52(4).

Gottschalk, M., Jelschen, J., and Winter, A. (2014). “Saving energy on mobile devices by refactoring..” *EnviroInfo*, 437–444.

Gottschalk, M., Jelschen, J., and Winter, A. (2016). “Refactorings for energy-efficiency.” *Advances and New Trends in Environmental and Energy Informatics*, Springer, 77–96.

Gottschalk, M., Josefiok, M., Jelschen, J., and Winter, A. (2012). "Removing energy code smells with reengineering services." *INFORMATIK 2012*.

Griss, M. L., Favaro, J., and d'Alessandro, M. (1998). "Integrating feature modeling with the rseb." *Proceedings. Fifth International Conference on Software Reuse (Cat. No. 98TB100203)*, IEEE, 76–85.

Hailpern, B. and Tarr, P. (2006). "Model-driven development: The good, the bad, and the ugly." *IBM systems journal*, 45(3), 451–461.

Hao, S., Li, D., Halfond, W. G., and Govindan, R. (2013). "Estimating mobile application energy consumption using program analysis." *Proceedings of the 2013 International Conference on Software Engineering*, IEEE Press, 92–101.

Heitkötter, H., Kuchen, H., and Majchrzak, T. A. (2015). "Extending a model-driven cross-platform development approach for business apps." *Science of Computer Programming*, 97, 31–36.

Heitkötter, H., Majchrzak, T. A., and Kuchen, H. (2013). "Cross-platform model-driven development of mobile applications with md2." *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 526–533.

Hinchey, M. G. and Sterritt, R. (2006). "Self-managing software." *Computer*, 39(2), 107–109.

Höpfner, H. and Schirmer, M. (2012). "Energy efficient continuous location determination for pedestrian information systems." *Proceedings of the Eleventh ACM International Workshop on Data Engineering for Wireless and Mobile Access*, ACM, 58–65.

Hoque, M. A., Siekkinen, M., Khan, K. N., Xiao, Y., and Tarkoma, S. (2016). "Modeling, profiling, and debugging the energy consumption of mobile devices." *ACM Computing Surveys (CSUR)*, 48(3), 39.

Ibrahim, M. and Youssef, M. (2012). "Cellsense: An accurate energy-efficient gsm positioning system." *IEEE Transactions on Vehicular Technology*, 61(1), 286–296.

Inverardi, P. and Mori, M. (2010). "Feature oriented evolutions for context-aware adaptive systems." *Proceedings of the Joint ERCIM Workshop on Software Evolution*

(EVOL) and International Workshop on Principles of Software Evolution (IWPSE), ACM, 93–97.

Inverardi, P. and Mori, M. (2011). "Requirements models at run-time to support consistent system evolutions." *2011 2nd International Workshop on Requirements@ Run. Time*, IEEE, 1–8.

Jabbarvand, R. and Malek, S. (2017). "μdroid: an energy-aware mutation testing framework for android." *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ACM, 208–219.

Jia, X. and Jones, C. (2015). "An approach for the automatic adaptation of domain-specific modeling languages for model-driven mobile application development." *IC-SOFT*, Springer, 365–379.

Jiang, H., Yang, H., Qin, S., Su, Z., Zhang, J., and Yan, J. (2017). "Detecting energy bugs in android apps using static analysis." *International Conference on Formal Engineering Methods*, Springer, 192–208.

Jung, W., Kang, C., Yoon, C., Kim, D., and Cha, H. (2012). "Devscope: a nonintrusive and online power analysis tool for smartphone hardware components." *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ACM, 353–362.

Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, A. (1990). "Feature-oriented domain analysis (foda) feasibility study." *Report No. CMU/SEI-90-TR-021*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, <http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11231>.

Kang, K. C., Kim, S., Lee, J., Kim, K., Shin, E., and Huh, M. (1998). "Form: A feature-; oriented reuse method with domain-; specific reference architectures." *Annals of Software Engineering*, 5(1), 143.

Kästner, C. and Apel, S. (2011). "Feature-oriented software development." *International Summer School on Generative and Transformational Techniques in Software Engineering*, Springer, 346–382.

Kelényi, I., Nurminen, J. K., Siekkinen, M., and Lengyel, L. (2014). *Advanced Computational Methods for Knowledge Engineering: Proceedings of the 2nd International Conference on Computer Science, Applied Mathematics and Applications (ICCSAMA 2014).* Springer International Publishing, Cham, Chapter Supporting Energy-Efficient Mobile Application Development with Model-Driven Code Generation, 143–156.

Kelly, S. and Tolvanen, J.-P. (2008). *Domain-specific modeling: enabling full code generation*. John Wiley & Sons.

Kemerlis, V. P., Portokalidis, G., Jee, K., and Keromytis, A. D. (2012). "libdft: Practical dynamic data flow tracking for commodity systems." *Acm Sigplan Notices*, Vol. 47, ACM, 121–132.

Kephart, J. O. and Chess, D. M. (2003). "The vision of autonomic computing." *Computer*, 36(1), 41–50.

Kern, E., Naumann, S., and Dick, M. (2015). "Processes for green and sustainable software engineering." *Green in Software Engineering*, Springer, 61–81.

Kim, C. H. P., Kroening, D., and Kwiatkowska, M. (2016a). "Static program analysis for identifying energy bugs in graphics-intensive mobile apps." *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016 IEEE 24th International Symposium on*, IEEE, 115–124.

Kim, D., Lee, S., and Bahn, H. (2016b). "An adaptive location detection scheme for energy-efficiency of smartphones." *Pervasive and Mobile Computing*, 31, 67–78.

Kim, K. and Cha, H. (2013). "Wakescope: runtime wakelock anomaly management scheme for android platform." *Proceedings of the Eleventh ACM International Conference on Embedded Software*, IEEE Press, 27.

Kjærgaard, M. B. and Blunck, H. (2011). "Unsupervised power profiling for mobile devices." *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, Springer, 138–149.

Kjærgaard, M. B., Langdal, J., Godsk, T., and Toftkjær, T. (2009). "Entracked: energy-efficient robust position tracking for mobile devices." *Proceedings of the 7th international conference on Mobile systems, applications, and services*, ACM, 221–234.

Kleppe, A. G., Warmer, J., Warmer, J. B., and Bast, W. (2003). *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional.

Krupitzer, C., Roth, F. M., VanSyckel, S., Schiele, G., and Becker, C. (2015). "A survey on engineering approaches for self-adaptive systems." *Pervasive and Mobile Computing*, 17, 184–206.

Le Goaer, O. and Waltham, S. (2013). "Yet another dsl for cross-platforms mobile development." *Proceedings of the First Workshop on the globalization of domain specific languages*, 28–33.

Lee, D. D. and Seung, H. S. (1999). "Learning the parts of objects by non-negative matrix factorization." *Nature*, 401(6755), 788.

Li, D., Hao, S., Gui, J., and Halfond, W. G. (2014). "An empirical study of the energy consumption of android applications." *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, IEEE, 121–130.

Li, L., Beitman, B., Zheng, M., Wang, X., and Qin, F. (2017a). "edelta: Pinpointing energy deviations in smartphone apps via comparative trace analysis." *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*, IEEE, 1–8.

Li, Q., Xu, C., Liu, Y., Cao, C., Ma, X., and Lü, J. (2017b). "Cyandroid: stable and effective energy inefficiency diagnosis for android apps." *Science China Information Sciences*, 60(1), 012104.

Li, Y., Guo, Y., Kong, J., and Chen, X. (2015). "Fixing sensor-related energy bugs through automated sensing policy instrumentation." *Low Power Electronics and Design (ISLPED), 2015 IEEE/ACM International Symposium on*, IEEE, 321–326.

Lin, K., Kansal, A., Lymberopoulos, D., and Zhao, F. (2010). "Energy-accuracy tradeoff for continuous mobile device location." *Proceedings of the 8th international conference on Mobile systems, applications, and services*, ACM, 285–298.

Liu, Y., Xu, C., Cheung, S.-C., and Lu, J. (2014). “Greendroid: Automated diagnosis of energy inefficiency for smartphone applications.” *IEEE Transactions on Software Engineering*, (1), 1–1.

Liu, Y., Xu, C., Cheung, S.-C., and Terragni, V. (2016). “Understanding and detecting wake lock misuses for android applications.” *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ACM, 396–409.

Ma, X., Huang, P., Jin, X., Wang, P., Park, S., Shen, D., Zhou, Y., Saul, L. K., and Voelker, G. M. (2013). “Edoctor: Automatically diagnosing abnormal battery drain issues on smartphones..” *NSDI*, Vol. 13, 57–70.

Mahmoud, S. S. and Ahmad, I. (2014). “A green model for sustainable software engineering.” *International Journal of Software Engineering and Its Applications*, 7(4), 55–74.

Malik, H., Zhao, P., and Godfrey, M. (2015). “Going green: An exploratory analysis of energy-related questions.” *Proceedings of the 12th Working Conference on Mining Software Repositories*, IEEE Press, 418–421.

Man, Y. and Ngai, E. C.-H. (2014). “Energy-efficient automatic location-triggered applications on smartphones.” *Computer Communications*, 50, 29–40.

Manotas, I., Bird, C., Zhang, R., Shepherd, D., Jaspan, C., Sadowski, C., Pollock, L., and Clause, J. (2016). “An empirical study of practitioners’ perspectives on green software engineering.” *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, IEEE, 237–248.

Marand, E. A., Marand, E. A., and Challenger, M. (2015). “Dsml4cp: a domain-specific modeling language for concurrent programming.” *Computer Languages, Systems & Structures*, 44, 319–341.

Mariakakis, A. T., Sen, S., Lee, J., and Kim, K.-H. (2014). “Sail: Single access point-based indoor localization.” *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, ACM, 315–328.

Mauro, J., Nieke, M., Seidl, C., and Yu, I. C. (2018). “Context-aware reconfiguration in evolving software product lines.” *Science of Computer Programming*, 163, 139–159.

Mehta, H., Owens, R. M., Irwin, M. J., Chen, R., and Ghosh, D. (1997). “Techniques for low energy software.” *Proceedings of the 1997 International Symposium on Low Power Electronics and Design*, ISLPED ’97, New York, NY, USA, ACM, 72–75, <http://doi.acm.org/10.1145/263272.263286>.

Mizouni, R., Serhani, M. A., Benharref, A., and Al-Abassi, O. (2012). “Towards battery-aware self-adaptive mobile applications.” *2012 IEEE Ninth International Conference on Services Computing*, IEEE, 439–445.

Moghimi, M., Venkatesh, J., Zappi, P., and Rosing, T. (2012). “Context-aware mobile power management using fuzzy inference as a service.” *International Conference on Mobile Computing, Applications, and Services*, Springer, 314–327.

Monsoon Solutions, I. (2018). *Mobile Device Power Monitor Manual Ver 1.19*, <https://msoon.github.io/powermonitor/PowerTool/doc/Power Monitor Manual.pdf> (Jun).

Morales, R., Saborido, R., Khomh, F., Chicano, F., and Antoniol, G. (2018). “Earmo: an energy-aware refactoring approach for mobile apps.” *IEEE Transactions on Software Engineering*, 44(12), 1176–1206.

Morillo, L. S., RamíRez, J. O., GarcíA, J. A., and Gonzalez-Abril, L. (2012). “Outdoor exit detection using combined techniques to increase gps efficiency.” *Expert Systems with Applications*, 39(15), 12260–12267.

Moura, I., Pinto, G., Ebert, F., and Castor, F. (2015). “Mining energy-aware commits.” *Proceedings of the 12th Working Conference on Mining Software Repositories*, IEEE Press, 56–67.

Murugesan, S. (2008). “Harnessing green it: Principles and practices.” *IT professional*, 10(1), 24–33.

Naik, K. (2010). “A survey of software based energy saving methodologies for handheld wireless communication devices.” *Report No. Tech. Report No. 2010-13*, Dept. of ECE, University of Waterloo. May 08, 2015.

Noorian, M., Bagheri, E., and Du, W. (2012). “Non-functional properties in software product lines: A taxonomy for classification..” *SEKE*, Vol. 12, 663–667.

Núñez, M., Bonhaure, D., González, M., and Cernuzzi, L. (2020). “A model-driven approach for the development of native mobile applications focusing on the data layer.” *Journal of Systems and Software*, 161, 110489.

Octeau, D., Jha, S., and McDaniel, P. (2012). “Retargeting android applications to java bytecode.” *Proceedings of the ACM SIGSOFT 20th international symposium on the foundations of software engineering*, ACM, 6.

Oliner, A. J., Iyer, A. P., Stoica, I., Lagerspetz, E., and Tarkoma, S. (2013). “Carat: Collaborative energy diagnosis for mobile devices.” *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, ACM, 10.

Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimhigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S., and Wolf, A. L. (1999). “An architecture-based approach to self-adaptive software.” *IEEE Intelligent Systems and Their Applications*, 14(3), 54–62.

Ortiz, G., García-de Prado, A., Berrocal, J., and Hernandez, J. (2019). “Improving resource consumption in context-aware mobile applications through alternative architectural styles.” *IEEE Access*, 7, 65228–65250.

Paek, J., Kim, J., and Govindan, R. (2010). “Energy-efficient rate-adaptive gps-based positioning for smartphones.” *Proceedings of the 8th international conference on Mobile systems, applications, and services*, ACM, 299–314.

Paek, J., Kim, K.-H., Singh, J. P., and Govindan, R. (2011). “Energy-efficient positioning for smartphones using cell-id sequence matching.” *Proceedings of the 9th international conference on Mobile systems, applications, and services*, ACM, 293–306.

Palomba, F., Di Nucci, D., Panichella, A., Zaidman, A., and De Lucia, A. (2019). “On the impact of code smells on the energy consumption of mobile applications.” *Information and Software Technology*, 105, 43–55.

Pang, C., Hindle, A., Adams, B., and Hassan, A. E. (2016). “What do programmers know about software energy consumption?.” *IEEE Software*, 33(3), 83–89.

Pascual, G. G., Lopez-Herrejon, R. E., Pinto, M., Fuentes, L., and Egyed, A. (2015a). “Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications.” *Journal of Systems and Software*, 103, 392–411.

Pascual, G. G., Pinto, M., and Fuentes, L. (2015b). “Self-adaptation of mobile systems driven by the common variability language.” *Future Generation Computer Systems*, 47, 127–144.

Pathak, A., Hu, Y. C., and Zhang, M. (2011). “Bootstrapping energy debugging on smartphones: A first look at energy bugs in mobile devices.” *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, New York, NY, USA, ACM, 5:1–5:6, <http://doi.acm.org/10.1145/2070562.2070567>.

Pathak, A., Hu, Y. C., and Zhang, M. (2012a). “Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof.” *Proceedings of the 7th ACM european conference on Computer Systems*, ACM, 29–42.

Pathak, A., Jindal, A., Hu, Y. C., and Midkiff, S. P. (2012b). “What is keeping my phone awake?: Characterizing and detecting no-sleep energy bugs in smartphone apps.” *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys ’12, New York, NY, USA, ACM, 267–280, <http://doi.acm.org/10.1145/2307636.2307661>.

Penzenstadler, B., Raturi, A., Richardson, D., and Tomlinson, B. (2014). “Safety, security, now sustainability: The nonfunctional requirement for the 21st century.” *Software, IEEE*, 31(3), 40–47.

Pinto, G. and Castor, F. (2017). “Energy efficiency: A new concern for application software developers.” *Commun. ACM*, 60(12), 68–75.

Pinto, G., Castor, F., and Liu, Y. D. (2014). “Mining questions about software energy consumption.” *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 22–31.

Prehofer, C. (1997). "Feature-oriented programming: A fresh look at objects." *European Conference on Object-Oriented Programming*, Springer, 419–443.

Rahman, A., Partho, A., Morrison, P., and Williams, L. (2018). "What questions do programmers ask about configuration as code?." *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*, ACM, 16–22.

Reiser, M.-O. (2009). *Core Concepts of the Compositional Variability Management Framework (CVM): A Practitioner's Guide*. Citeseer.

Runeson, P. and Höst, M. (2009). "Guidelines for conducting and reporting case study research in software engineering." *Empirical software engineering*, 14(2), 131.

Sahar, H., Bangash, A. A., and Beg, M. O. (2019). "Towards energy aware object-oriented development of android applications." *Sustainable Computing: Informatics and Systems*, 21, 28–46.

Sahu, M. and Mohapatra, D. P. (2017). "Computing dynamic slices of feature–oriented programs using execution trace file." *ACM SIGSOFT Software Engineering Notes*, 42(2), 1–16.

Sahu, M. and Mohapatra, D. P. (2019). "Computing dynamic slices of concurrent feature-oriented programs." *Arabian Journal for Science and Engineering*, 44(11), 9471–9497.

Sahu, M. and Mohapatra, D. P. (2020). "Computing dynamic slices of feature-oriented programs with aspect-oriented extensions." *Informatica*, 44(2).

Salehie, M. and Tahvildari, L. (2009). "Self-adaptive software: Landscape and research challenges." *ACM Trans. Auton. Adapt. Syst.*, 4(2), 14:1–14:42.

Schilit, B., Adams, N., and Want, R. (1994). "Context-aware computing applications." *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, WMCSA '94, Washington, DC, USA, IEEE Computer Society, 85–90, <http://dx.doi.org/10.1109/WMCSA.1994.16>.

Schmidt, D. C. (2006). "Guest editor's introduction: Model-driven engineering." *Computer*, 39(2), 25–31.

Schobbens, P.-Y., Heymans, P., and Trigaux, J.-C. (2006). "Feature diagrams: A survey and a formal semantics." *14th IEEE International Requirements Engineering Conference (RE'06)*, IEEE, 139–148.

Schulman, A., Schmid, T., Dutta, P., and Spring, N. (2011). "Phone power monitoring with battor." *Proc. Annu. ACM Int. Conf. Mobile Comput. Netw.*

Selic, B. (2003). "The pragmatics of model-driven development." *IEEE software*, 20(5), 19–25.

Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., and Saake, G. (2012). "Spl conqueror: Toward optimization of non-functional properties in software product lines." *Software Quality Journal*, 20(3-4), 487–517.

Sinnema, M. and Deelstra, S. (2007). "Classifying variability modeling techniques." *Information and Software Technology*, 49(7), 717–739.

Soltani, S., Asadi, M., Gašević, D., Hatala, M., and Bagheri, E. (2012). "Automated planning for feature model configuration based on functional and non-functional requirements." *Proceedings of the 16th International Software Product Line Conference-Volume 1*, ACM, 56–65.

Su, C.-L., Tsui, C.-Y., and Despain, A. (1994). "Low power architecture design and compilation techniques for high-performance processors." *Compcon Spring '94, Digest of Papers.*, 489–498 (Feb).

Suh, G. E., Lee, J. W., Zhang, D., and Devadas, S. (2004). "Secure program execution via dynamic information flow tracking." *ACM Sigplan Notices*, Vol. 39, ACM, 85–96.

Surhone, L. M., Tennoe, M. T., and Henssonow, S. F. (2010). *Goal-Oriented Requirements Language*. Betascript Publishing, Beau Bassin, MUS.

Tahir, A., Yamashita, A., Licorish, S., Dietrich, J., and Counsell, S. (2018). "Can you tell me if it smells?: A study on how developers discuss code smells and anti-patterns in stack overflow." *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, ACM, 68–78.

Thokala, S. K., Koundinyaa, P., Mishra, S., and Shi, L. (2014). "Virtual gps: A middleware for power efficient localization of smartphones using cross layer approach." *Proceedings of the Middleware Industry Track*, Industry papers, New York, NY, USA, ACM, 2:1–2:7, <http://doi.acm.org/10.1145/2676727.2676729>.

Thramboulidis, K. and Christoulakis, F. (2016). "Uml4iot—a uml-based approach to exploit iot in cyber-physical manufacturing systems." *Computers in Industry*, 82, 259–272.

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., and Leich, T. (2014). "Featureide: An extensible framework for feature-oriented software development." *Science of Computer Programming*, 79, 70–85.

Truyen, F. (2006). "The fast guide to model driven architecture the basics of model driven architecture." *Cephas Consulting Corp*.

Tufail, H., Azam, F., Anwar, M. W., and Qasim, I. (2018). "Model-driven development of mobile applications: A systematic literature review." *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, IEEE, 1165–1171.

Van Deursen, A. and Klint, P. (2002). "Domain-specific language design requires feature descriptions." *Journal of Computing and Information Technology*, 10(1), 1–17.

Van Gurp, J., Bosch, J., and Svahnberg, M. (2001). "On the notion of variability in software product lines." *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*, IEEE, 45–54.

Vaquero-Melchor, D., Palomares, J., Guerra, E., and de Lara, J. (2017). "Active domain-specific languages: Making every mobile user a modeller." *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, IEEE, 75–82.

Vaupel, S., Taentzer, G., Gerlach, R., and Guckert, M. (2018). "Model-driven development of mobile applications for android and ios supporting role-based app variability." *Software & Systems Modeling*, 17(1), 35–63.

Vogel, T. and Giese, H. (2014). "Model-driven engineering of self-adaptive software with eurema." *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 8(4), 18.

Völter, M. "Best practices for dsls and model-driven development." *Journal of Object Technology*, 8(6), 79–102.

Wan, M., Jin, Y., Li, D., Gui, J., Mahajan, S., and Halfond, W. G. (2017). "Detecting display energy hotspots in android apps." *Software Testing, Verification and Reliability*, 27(6), e1635.

Wan, M., Jin, Y., Li, D., and Halfond, W. G. (2015). "Detecting display energy hotspots in android apps." *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*, IEEE, 1–10.

Wang, J., Liu, Y., Xu, C., Ma, X., and Lu, J. (2016). "E-greendroid: effective energy inefficiency analysis for android applications." *Proceedings of the 8th Asia-Pacific Symposium on Internetware*, ACM, 71–80.

Wang, X., Li, X., and Wen, W. (2014). "Wlcleaner: Reducing energy waste caused by wakelock bugs at runtime." *Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on*, IEEE, 429–434.

Weimer, W. and Necula, G. C. (2004). "Finding and preventing run-time error handling mistakes." *ACM SIGPLAN Notices*, Vol. 39, ACM, 419–431.

Xi, T., Wang, W., Ngai, E. C.-H., Song, Z., Tian, Y., and Gong, X. (2015). "Energy-efficient collaborative localization for participatory sensing system." *Global Communications Conference (GLOBECOM), 2015 IEEE*, IEEE, 1–6.

Yadav, K., Naik, V., Kumar, A., and Jassal, P. (2014). "Placemap: Discovering human places of interest using low-energy location interfaces on mobile phones." *Proceedings of the Fifth ACM Symposium on Computing for Development*, ACM, 93–102.

Yang, Z., Li, Z., Jin, Z., and Zhang, H. (2017). "Review on requirements modeling and analysis for self-adaptive systems: A ten-year perspective." *arXiv preprint arXiv:1704.00421*.

Yoon, C., Kim, D., Jung, W., Kang, C., and Cha, H. (2012). “Appscope: Application energy metering framework for android smartphone using kernel activity monitoring..” *USENIX Annual Technical Conference*, Vol. 12, 1–14.

Yu, E. S. (1997). “Towards modelling and reasoning support for early-phase requirements engineering.” *Proceedings of ISRE’97: 3rd IEEE International Symposium on Requirements Engineering*, IEEE, 226–235.

Zhang, L., Gordon, M. S., Dick, R. P., Mao, Z. M., Dinda, P., and Yang, L. (2012). “Adel: An automatic detector of energy leaks for smartphone applications.” *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ACM, 363–372.

Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M., and Yang, L. (2010). “Accurate online power estimation and automatic battery behavior based power model generation for smartphones.” *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ACM, 105–114.

Zhu, C., Zhu, Z., Xie, Y., Jiang, W., and Zhang, G. (2019). “Evaluation of machine learning approaches for android energy bugs detection with revision commits.” *IEEE Access*, 7, 85241–85252.

Zhuang, Z., Kim, K.-H., and Singh, J. P. (2010). “Improving energy efficiency of location sensing on smartphones.” *Proceedings of the 8th international conference on Mobile systems, applications, and services*, ACM, 315–330.

Zhuo-Qun, Y. and Zhi, J. (2012). “Requirements modeling and system reconfiguration for self-adaptation of internetware.” *Proceedings of the Fourth Asia-Pacific Symposium on Internetware*, Internetware ’12, New York, NY, USA, ACM, 11:1–11:6, <http://doi.acm.org/10.1145/2430475.2430486>.

Zimmermann, T. (2016). “Card-sorting: From text to themes.” *Perspectives on Data Science for Software Engineering*, Elsevier, 137–141.