



REPUBLIC OF TURKEY

ALTINBAS UNIVERSITY

Institute of Graduate Studies

Electrical and Computer Engineering/ PHD

**FEATURE MODEL-BASED CONFIGURATION REUSE
SCHEME FOR SELF-ADAPTIVE SYSTEMS**

Alkubaisi, Sumayah

Thesis submitted for the degree of Doctor Of Philosophy

Supervisor

Asst. Prof. Dr. Oguz Ata

Prof. Dr. Said Ghoul

Istanbul, 2021

FEATURE MODEL-BASED CONFIGURATION REUSE SCHEME FOR SELF-ADAPTIVE SYSTEMS

[Al-Kubaisi, Sumayah]

Electrical and Computer Engineering

Submitted to the Graduate School of Science and Engineering
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

ALTINBAŞ UNIVERSITY

[2021]

The thesis titled Feature Model-Based Configuration Reuse Scheme for Self-Adaptive Systems prepared by Sumayah Alkubaisi and submitted on (Date) has been **accepted by majority of votes** for the degree of PhD in Electrical and Computer Engineering.

Prof. Dr. Said Ghoul

the Co-Supervisor

Asst. Prof. Dr. Oguz Ata

the Supervisor

Thesis Defense Committee Members:

Asst. Prof. Dr. Dogu Cagdas Atilla

School of Engineering and
Natural Sciences,

Altinbas University

Asst. Prof. Dr. Oguz Ata

School of Engineering and
Natural Sciences,

Altinbas University

Asst. Prof. Dr. Oguz Karan

School of Engineering and
Natural Sciences,

Altinbas University

Asst. Prof. Dr. Aytug Boyaci

Air Force Academy,

National Defense University

Assoc. Prof. Dr. Metin Zontul

Faculty of Economic,
Administrative and Social,
Science

Istanbul Ayvansaray

University

I hereby declare that this thesis/dissertation meets all format and submission requirements of a (PhD) thesis.

Accepted by Altınbaş University Institute of Graduate
Studies on the following date:

____/____/____

I hereby declare that all information/data presented in this graduation project has been obtained in full accordance with academic rules and ethical conduct. I also declare all unoriginal materials and conclusions have been cited in the text and all references mentioned in the Reference List have been cited in the text, and vice versa as required by the abovementioned rules and conduct.

Sumayah Alkubaisi

Signature

DEDICATION

I dedicate this work to my beloved parents. I appreciate everything you have done for me. Without you it was almost impossible to me to complete this thesis work.



ACKNOWLEDGEMENTS

I would like to express my thanks to my supervisor Ass. Prof. Dr. Oguz Ata and my Co-supervisor Prof. Dr. Said Ghoul for being there from beginning to end, providing me with guidance and valuable advice, believing in me, and giving me a space to seek and discover things by myself, while keeping a watchful eye upon my work.

I am grateful to my family members; my beloved parents, who have always been there for me, educated me and unconditionally supported me to reach this point; my sisters and brothers for their love, help and support through the years.

I am grateful to my teachers who gave us the best of their ability and to my friends who supported me and motivated me all the way.

Above all, I am thankful to the Lord, the Almighty for granting me opportunity, strength, persistence and lightening my way.

ABSTRACT

Feature Model-Based Configuration Reuse Scheme for Self-Adaptive Systems

Al-Kubaisi, Sumayah,

PhD, Electrical and Computer Engineering, Altınbaş University,

Supervisor: Asst. Prof. Dr. Oguz Ata

Co-Supervisor: Prof. Dr. Said Ghoul

Date: [Month Year]

Pages: (Total page number, starting from ABSTRACT page)

Large systems have become progressively more complicated over the recent years, leading to increased focus on system maintenance and configurations. Self-adaptive systems (SASs) are considered to be of great importance in the handling of complex systems. SASs are capable of reconfiguring themselves during run time to satisfy the scenarios of the requisite contexts (corresponding to changes in user requirements, systems themselves, or their environments). However, reconfiguration process need to be triggered each time the system is adapted. In this context, configuration reuse is a prospective alternative to reduce time, effort, and error-prone.

Most self-adaptive systems utilize feature models (FMs) to represent their complex architectures and benefit from the reuse of commonalities and variability information. However, reconfiguration of self-adaptive systems corresponding to each adaptation of the system requires significant computational time and resources. While current FM-based techniques represent and reuse elementary features to model SASs concepts, modeling and reusing configurations have not yet been considered. In this context, this study presents an extension to FMs by introducing and

managing configuration features and their reuse process. Evaluation results demonstrate that reusing configuration features reduces the effort and time required by reconfiguration processes during the run time to meet the required scenario according to the context.

Keywords: Self-adaptive systems, Feature models, System reuse, Configuration management, Variability modeling.



TABLE OF CONTENTS

	<u>Pages</u>
ABSTRACT.....	vii
LIST OF TABLES	xii
LIST OF FIGURES	xiii
LIST OF ABBREVIATIONS	xvi
LIST OF SYMBOLS.....	xviii
1. INTRODUCTION	1
1.1 PREFACE.....	1
1.2 RESEARCH CONTEXT.....	2
1.2.1 Variability Feature Modelling.....	2
1.2.2 Self-Adaptive Systems modelled with Feature Models.....	3
1.2.3 System Reuse.....	4
1.3 PROBLEM STATEMENT.....	4
1.4 MOTIVATION.....	5
1.5 CONTRIBUTION.....	6
1.6 THESIS OUTLINE.....	7
2. BACKGROUND.....	10
2.1. FEATURE MODELS (FMs).....	10
2.1.1. Feature oriented domain analysis (Classical Feature models).....	10
2.1.2 Context oriented domain analysis.....	13
2.2 TEXTUAL REPRESENTATION OF FMs.....	15
2.3 USER PROFILE.....	17

2.4 CONFIGURATION FEATURE.....	18
2.5 SUMMERY.....	19
3. LITERATURE WORK.....	20
3.1 INTRODUCTION.....	20
3.2 FEATURE VARIABILITY MODELLING.....	20
3.3 SELF-ADAPTIVE SYSTEMS.....	27
3.4 SELF-ADAPTIVE SYSTEMS MODELLED WITH FEATURE MODELS.....	28
3.5 SYSTEM REUSE.....	38
3.6 RESEARCH GAP.....	40
3.7 OPEN PROBLEMS AND THESIS SCOP.....	41
3.8 SUMMERY.....	42
4. CONFIGURATION FEATURE MODELLING AND REUS EXTENSION.....	43
4.1 MODELLING SASs DEVELOPED WITH REUSE PROCESS.....	43
4.2 ILLUSTRATIVE EXAMPLE.....	46
4.3 MODELLING OF CONFIGURATIN FEATURES.....	48
4.3.1 Enhanced FM with configuration – example.....	48
4.3.2 Definitions.....	48
4.4 CONFIGURATION FEATURE MANAGEMENT.....	49
4.4.1 Configuration feature framework.....	49
4.4.2 Configuration feature management process.....	50
4.4.3 Configuration management – examples.....	54

4.5 CASE STUDY.....	63
4.6 SUMMARY.....	67
5. EVALUATION AND CONCLUSION.....	69
5.1 INTRODUCTION.....	69
5.2 APPLICATION AREA.....	69
5.3 EVALUATION.....	70
5.4 CONCLUSION.....	74
5.5 SUMMARY.....	75
REFERENCES.....	76
APPENDIX A.....	88
APPENDIX B.....	97

LIST OF TABLES

	<u>Pages</u>
Table 2.1: the textual representation of a set of propositional constraint of a FM using EBNF notations [4].....	16
Table 2.2: the structure of Features of a feature diagram in propositional logic [4].....	16
Table 5.1: Comparison between various configuration management schemes modeled using FM...	74

LIST OF FIGURES

	<u>Pages</u>
Figure 2.1: A mandatory parent-child relationship.....	11
Figure 2.2: An optional parent-child relationship.....	11
Figure 2.3: An alternative-group relationship.....	12
Figure 2.4: An or-group relationship.....	12
Figure 2.5: An and-group relationship.....	13
Figure 2.6: The FM of list software after adding a configuration feature [22].....	19
Figure 3.1: FODA: Monitor Engine System [56].....	21
Figure 3.2: Manufacturer’s Options for the Entertainment System of Car, [15].....	22
Figure 3.3: Textual Design Methodology (TDM) mixing class and feature concepts, [17].....	23
Figure 3.4: Example of dynamic-priority based In consistent Feature Models (IFM) fixing. (a) Example feature model. (b) Corresponding a locally-graph-better (LGB) method graph, for the example feature model in (a), [59].....	24
Figure 3.5: The tow-layers FM, [34].....	25
Figure 3.6: The elementary feature model structure using EBNF notations [20, 21].....	26
Figure 3.7: The conceptual architecture of SAS [6].....	28
Figure 3.8: Dynamic reconfiguration approach, [82].....	30
Figure 3.9: an over view of the proposed approach [84].....	31
Figure 3.10: Partial feature model of the Baby Care System [35].....	32
Figure 3.11: The Hy-PAB conceptual architecture [55].....	32

Figure 3.12: Architecture of the proposed method [36].....	33
Figure 3.13: The concept for reconfiguration [37].....	34
Figure 3.14: An example of temporal validities of features [87].....	34
Figure 3.15: The methodology of feature-based variability modelling in software evolution [21].....	36
Figure 3.16: The device control unit (DCU) feature model extract with binding time annotations [39].....	36
Figure 3.17: shows the reusable adaptation logic (AL) model [45].....	39
Figure 3.18: The ASPL strategy and the ASPLe methodology [6].....	40
Figure 4.1: a configuration that is responsible for emergency call in Europe.....	44
Figure 4.2: a configuration that is responsible for emergency call in Russia.....	45
Figure 4.3: Feature model for a car system [4].....	47
Figure 4.4: Enhanced feature model for a car system with configuration features.....	48
Figure 2.5: Configuration feature framework.....	50
Figure 4.6: State diagram of the configuration process.....	51
Figure 4.7: Composing state framework.....	52
Figure 4.8: Saving state framework.....	53
Figure 4.9: Reusing state framework.....	54
Figure 4.10: Screenshot of the FM following the creation of the three driving styles, modeled using the FM editor of DarwinSPL.....	65
Figure 4.11: Updated FM with the configuration features, modeled using the FM editor of DarwinSPL.....	66

Figure 4.12: Contextual elements of the FM, modeled using the contextual information editor of DarwinSPL.....	66
Figure 4.13: VF of the FM, modeled using the VF editor of DarwinSPL.....	67
Figure 4.14: Constraints of the FM, modeled using the constraints editor of DarwinSPL.....	67



LIST OF ABBREVIATIONS

SASs	:	Self-Adaptive Systems
FMs	:	Feature Models
DSPLs	:	Dynamic Software Product Lines
FODA	:	Feature-Oriented Domain Analysis
SFD	:	Service Feature Diagrams
CSFD	:	Cardinality-based service feature diagrams
TDM	:	Textual Design Methodology
SPLs	:	Software Product Lines
HyFM	:	Hybrid Feature Model
ASPLe	:	Autonomic Software Product Line engineering
DA	:	Domain Analysis
CTC	:	Cross-tree constraints
VF	:	Validity Formula
EBNF	:	Extended Backus-Naur Form
ECUs	:	Electronic Control Units
Hy-PAB	:	Hybrid passive/active behaviour
SLAM	:	Simultaneous Localization and Mapping
TfMs	:	Temporal Feature Models
ExFM	:	Extended Feature Model
DCU	:	Device Control Unit

DE : Domain Engineering

ReSide : Reusable Service Identification

SPVs : Software Product Variants

AL : Adaptation Logic



LIST OF SYMBOLS

ConfigReuse	:	Configuration Reuse
ConfigAvilable	:	Configuration Reuse
CofigProg	:	Configuration Program
$\wedge$	:	And
$\vee$	:	Or
$*$	:	Repeat zero or more
$+$	:	Repeat at least once

1. INTRODUCTION

1.1 PREFACE

Nowadays, complexity of systems is increasing, causing increasing of effort for configurations and system maintenance. A new perspective of Self-adaptive system including software evolution, context aware, dynamic software product lines (DSPLs), and self-adaptation has assumed as greater importance for handling complex systems [1]. SASs are capable of changing their behaviour, functions and structures, reconfiguring process, during the run time in order to meet the required scenario according to the current context (the change in user requirements, in the system itself, or in its environment).

One of the important aspects that must be considered when developing a SAS is changing the system behaviour during the runtime without affecting the system itself. There are several works attempt to implement SAS architectures that have the capabilities of adapting, making decisions, and evolving during the run time, but still have lacks of implementing all required characteristic of SAS mechanisms [1]. SAS's are required to be provided with self-* properties in order to have the ability for executing self-adaptation concept [2]. Self-* properties [3] include, among others, self-configuration, self-healing, self-optimization, and self-protection.

However, the reconfiguration process will be triggered every time the system in need of adaption according to the current context [4]. Hence, configuration reuse process would be a better alternative for some context to reduce time, effort and error-prone. Software reuse means that appropriate available software components can be reused to reduce cost or time for computation, with a slight or no modifications [5]. Complexity can be decreased in the development of systems by reusing components or elements. Moreover, the development of the system itself can be fasten and less error-proneness by reusing components and processes.

SAS were poorly developed with reuse process support, hence this issue may prevent knowledge pass on or quality design [6]. Further, concerning the self-configuration property of SAS, neither the configuration feature nor its reuse process are supported and formally modelled especially in system feature modeling.

Several large-scale systems, including SASs, use FMs to represent their complex architectures [7], by managing, expressing, and reusing commonalities and variability information among their products. Each feature in the FM represents a distinct ability of the system. The reuse of features reduces time, cost, and effort and improves the quality of the product as well as that of the software process [8-10].

1.2 RESEARCH CONTEXT

This work deals with Self-Adaptive Systems , Variability Feature Modelling and System Reuse. Hence, approaches stated in its research context are; Variability Feature Modelling, Self-Adaptive Systems modelled with Feature Models and System Reuse.

1.2.1 Variability Feature Modelling

Feature diagrams were firstly presented by [11] which was a part of Feature-Oriented Domain Analysis (FODA). Feature diagrams are hierarchal representations of a system's commonalities and variabilities in terms of features. Each feature in the FM represents a distinctive ability or functionality of the system. The top feature of an FM is called root or concept. Each feature can possess multiple child features. Features are either mandatory or optional, and arranged alternatively (only one feature is selected) or in groups (at least one feature is selected).

Since that time, several studies have been conducted on FMs to address the variability and complexity of constituent systems, and preserve their alignment to variable systems by adding extensions in different domains, such as; Feature-RSEB [12], Cardinality-based feature diagrams [13], service feature diagrams SFD [14], Cardinality-based service feature diagrams CSFD [15], which is an extension work of [14] and [16], A Textual Design Methodology TDM [17].

Several research works were produced to represent feature models in formal way [18-22]. In [18], authors presented a formal specification framework for the software product lines (SPLs). This framework used algebra process (SPLA) to express and describe the FODA diagram [11]. Further, the framework is general and flexible to express and describe other extensions of feature modelling diagrams.

Nevertheless, many research works produced to deal with feature selection [23-28]. The benefit of feature selection, among others, is to eliminate noisy, redundant or inappropriate data, decrease in data size, improve accuracy of prediction, extract important features, reduce the execution time and lead to better learning performance [29] and [30].

Certain works have focused on the management and improvement of FM customization, which is a popular method to reduce cost and improve user satisfaction [31-34]. However, the software configuration is yet to be modeled as a feature in the FM.

1.2.2 Self-Adaptive Systems Modelled With Feature Models

The concept of adaption, making decisions and evolution is essential to systems, since user requirements and needs, environment, or the system itself change over the time, hence systems must adapt to remain useful [7]. Feature models (FMs) are extensively used to represent systems' variability [7].

Several studies [35-38] have incorporated FMs into the system adaptation process, but no notions or extensions have been introduced into FMs to enable them to model these adaptation processes or their reused processes.

A few works have enriched FMs using concepts and extensions to enable them to model SASs. The appended concepts, including elementary features, equipped with their relations, processes, and constraints, are designed to ensure coherence [4], [39], and [21].

All of the above works focused on modelling and reusing elementary features without taking into consideration the configuration reuse management, except for [21], where authors modelled the metamorphoses concept which is a grouping of features (configuration). So, basically they have modelled the configuration by the FM but without the reuse process.

One of the most significant work in the domain is [4]. Authors introduced a framework for SPLs that allow modeling of contextual influences, user customization and evolution. In particular, a Hybrid Feature Model (HyFM) is presented. It supports concepts of contextual information, validity formula, cross tree constraint and version aware. However, this approach does not handle configuration reuse management. Moreover, a reconfiguration process starts on every time a change in the context occurs or an artifact of the system has been evolved by an engineer. This leads to cost in time and effort.

1.2.3 System Reuse

Complexity can be decreased in the development of systems by reusing components or elements. Moreover, the development of the system itself can be fasten and less error-proneness by reusing components and processes. FMs are one of the new forms of reuse that are able to bring new opportunities for the reuse process beyond the conventional software components [40].

Reusability has previously been applied to SASs for example: Rainbow strategy [41], MADAM framework [42], MUSIC framework [43], and ENTRUST strategy [44]. However, none has produced a design or developed a self-adaptive system with reuse process. Moreover, the formal modeling of a configuration reuse process or reused configurations remains lacking, especially with respect to system feature models (FMs).

In [45] the authors provided an architecture of reusable communication with utilizing of templates of abstract component. However, no notions have been added to the FM. This architecture reduced the development time and enabled an easier adaptation during runtime.

In [6], an autonomic software product line engineering (ASPLe) methodology was presented. This methodology allows developers to develop SASs with reuse processes by selecting, specializing, and subsequently, integrating reusable artifacts. This methodology used FMs to model only the scope definition of ASPLe; hence, no concepts were further added to the FM to support the modeling of the developed SAS with reuse process.

In [46] an automated test reuse for highly configurable software was presented. Authors automatically reused existing tests (manually developed) in configurable system to produce new test variants.

However, the compound of the runtime variability of self-adaptive systems and the reuse process could lead to complex area. In order to reduce complexity of self-adaptive systems developed with reuse process, formal variability modelling of the self-adaptive system itself, its context properties, and its stepwise configurations as features and their reuse process must be supported by using the feature model .

1.3 PROBLEM STATMENT

Based on the above researches, the following challenges are concluded:

- I. Self-adaptive systems were poorly developed with reuse process support, which might prevent knowledge pass on or quality design.
- II. All of the aforementioned works have focused on modeling and reusing elementary features without considering the management of configuration features and their reuse process, except for [21] in which the authors modeled the concept of metamorphosis, which involves the grouping of features (pre-defined configuration). To this end, they modeled a pre-defined configuration using an FM, but did not model the reuse process.
- III. None of the above approaches has empowered the feature model itself with concepts for representing configuration features and configuration reuse process. Hence, FMs are not designed to support self-adaptive systems developed with reuse process.
- IV. The reconfiguration process has been triggered every time the system needs to adapt to the context
- V. The compound of the runtime variability of self-adaptive systems and the reuse process could lead to complex area.
- VI. Research gap between the systems reuse and self-adaptive systems need to be reduced.

1.4 MOTIVATION

- I. The perspective of Self-adaptive systems (SASs) including software evolution, context-aware/ context-dependent, dynamic software product line (DSPL), and self-adaptation are increasing and used more and more these days. For example: autonomous cars, wearable devices and smart homes. The need to model such systems as system features has led to the need to represent and model their context properties as a part of the process of the domain analysis (DA) [40].
- II. Feature models (FMs) are extensively used to represent systems' variability and complexity, yet not flexible enough to represent and express any system. hence they need to be more expressive and have extensions in order to make them capable to represent more self-adaptive systems processes and concepts including SASs developed with reuse concepts.

- III. Reusability has previously been applied to SASs, but the formal modeling of a configuration reuse process or reused configurations remains lacking, especially with respect to system feature models (FMs).
- IV. Instead of triggering the reconfiguration process every time the system needs to adapt to the context, configuration reuse process is a suitable alternative solution in some contexts to reduce the development time, effort, error-proneness, and cost.
- V. In order to reduce complexity of self-adaptive systems developed with reuse process, formal variability modelling of the self-adaptive system itself, its context properties, and its stepwise configurations as features and their reuse process must be supported by using the feature model .

1.5 CONTRIBUTION

Based on the above insufficiencies, this thesis presents an extension to the FM to empower it to support and manage the modelling of self-adaptive systems developed with reuse process. Mainly, the proposed extension provides the following:

- I. Modelling configuration features concept.
- II. Efficient reuse process of configuration features.
- III. New features, context information, and constraints are introduced to ensure coherency between features.
- IV. Two types of configuration features are supported: default configuration feature and custom defined configuration feature.
- V. Two type of configuration features' reuse has been introduced: *direct reuse* and *indirect reuse*. These configuration features will be reused every time it fulfills the requirements of a new context during the run time.
- VI. Configuration feature management process has been defined to manage the configuration features and their reuse process from its creation to its destruction. In addition, it defines all the states of a configuration feature throughout its lifetime:
 - a) Manufacturing state

- b) Composing State
- c) Saving State
- d) Reusing State

1.6 THESIS OUTLINE

The thesis is structured as follows:

The background information is provided in chapter 2, which includes:

- I. Feature models (FMs)
 - a) Feature oriented domain analysis (Classical Feature models), where FODA notations [11] and Feature-RSEB extension [12] are discussed.
 - b) Context oriented domain analysis, where constraints and notations provided by the new extensions from different works that have been added to FMs in order to make them able to express and support SAS' context properties and requirements are discussed.
- II. Textual representation of FMs: information about the formal representation of FMs by interpreting the feature diagrams into formulas using logic.
- III. User Profile: information about customizing a configuration of the system either by the manufacturer or by a user is provided.
- IV. Configuration features: basic information about configuration features are provided.

In chapter 3 the works that are related to this thesis work are highlighted. In addition, the differences between them and our approach is discussed. the discussion about the related work is categorized into the following:

- I. Feature Variability Modelling: FODA [11] approach and other extension that have been made to the FMs to handle complexity and system's variability process in different domains is discussed.

- II. Self-adaptive systems: definition and approaches of self-adaptive systems are provided and discussed.
- III. Self-adaptive systems modelled with feature models: the literature works related to Self-adaptive systems based on Feature Variability Modelling are investigated and discussed. Further, the required processes, features, their relation, and constraints ensuring their coherence that have been added to the FM in order to model and represent requirements, context properties, reuse process, and concepts of the SAS are mined and identified.
- IV. System reuse: the software reuse process is identified and discussed. In addition, the related works to reuse process and to self-adaptive systems developed with reuse process are discussed.
- V. Research gap: the gap resulting from compounding the run time variability self-adaptive systems with the reuse process are discussed.
- VI. Open problems and thesis scope:
 - a) The challenges raised from compounding the self-adaptive systems with the reuse process and from formally representing them by the variability feature modelling systems are discussed.
 - b) Then, the solutions that are provided by this these in order to remedy the aforementioned challenges is presented.

In chapter 4, the contribution of this thesis is presented, where the configuration features concept and their reuse process of the new extension to the FM is identified and discussed with help of examples as follows:

- I. Modelling SASs developed with reuse process: challenges raised by modelling SASs developed with reuse process are discussed.

- II. Illustrative example: an illustrative example is presented to provide information about the concepts introduced in this work
- III. Modeling of Configuration Features: definitions of the proposed configuration features' extension along with its configuration features, relations, and constraints to ensure their coherency, which is able to address the challenges is presented with the help of an example.
- IV. Configuration feature management: the configuration features' concept management including:
 - a) the proposed framework,
 - b) its process and states will be presented with the help of examples.
- V. Case study: the case study with the results are presented

Chapter 5 consists the following:

- I. Applications area: the applications area of this approach is presented and discussed.
- II. Evaluation: the evaluation of this approach is done depending on the introduced concepts in variability feature modelling domain, self-adaptive systems approaches, reuse process concepts, and on the new concepts introduced and defined by this work and also, how many of these claims have been achieved.
- III. Conclusion and future work: conclusion of the work is presented, then future work directions is discussed.

2. BACKGROUND

In this chapter, the background information which our work is founded on is provided. Firstly, we provide information about feature models with its two types__ feature oriented domain and context oriented domain. Secondly, we provide information about two concepts__ user profile and configuration feature. As the understanding of these techniques and concepts is required throughout the contribution of this work.

2.1. FEATURE MODELS (FMs)

2.1.1. Feature Oriented Domain Analysis (Classical Feature Models)

Feature models (FMs) are diagrams that were firstly introduced by FODA [11]. FMs are hierarchal representations of a system's commonalities and variability in terms of features [11]. Each feature (elementary feature) in the FM represents a distinctive ability or functionality of the system. For example, gear type, engine type or seat type can be used in a FM as features of Car manufacturer system. a feature model is capable to represent and express all the features of a software product line in a single model. A valid selection of a group of features of a FM is called a configuration. The top feature of an FM is called its root or concept. The FM contains set of constrains that are as follows.

Parent-child: a vertical connection (relationship) of features (parent) with its subfeatures (child) in a FM. Each feature parent can possess multiple child features. For instance, feature F2 is a subfeature of feature F1, as shown in figure 2.1.

Cross-tree: a horizontal connection (dependency relation) among features of a FM. Cross-tree constraints consists tow type of constraints:

- a) *Require*: imposes the selection of the target feature (F2) if its source feature (F1) is selected. For instance, the source feature F1 implies the target feature F2 if a configuration holding feature F1 must hold feature F2 too.

- b) *Excludes*: imposes the non-selection of both features, source and target in the same configuration. For instance, the source feature F1 excludes the target feature F2 if a configuration holding feature F1 must not hold feature F2.

Moreover, notations used by the FM are as follows.

FODA [11] defined three type of parent-child constraints notations that are:

Mandatory: if a feature F1 is selected in a configuration, so its mandatory subfeature F2 must be selected too. Mandatory parent-child relationship is represented as a straight line in FODA [11] or with a filled circle in others. Figure 2.1 shows the relationship of mandatory parent-child between feature F1 and feature F2.



Figure 2.1: A mandatory parent-child relationship.

Optional: if a feature F1 is selected in a configuration, then its optional subfeature F2 might be selected or rejected. An optional parent-child relationship is represented by a hallow circle as shown in figure 2.2.

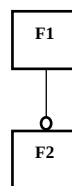


Figure 2.2: An optional parent-child relationship.

Alternative-group: if a feature F1 is selected in a configuration, then, only one subfeature Fi from its alternative-group (F2...Fn) is selected. In other words, a feature F1 could be composed exclusively by feature F2, or by feature F3 but not by both. An alternative-group relationship is represented by a hallow arc as shown in figure 2.3.

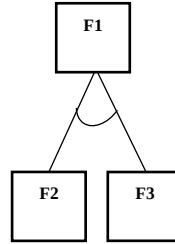


Figure 2.3: An alternative-group relationship.

Feature-RSEB [12], is an extension that has been made to FODA FM. Although this work is fully supported FODA and has similar semantics, authors provided new parent-child notation to the FM. This notation are or-group and and-group relationships.

Or-group: if a feature F1 is selected in a configuration, then at least one of its subfeaturus (F2..Fn) from its or-group must be selected. In other words, a feature F1 could be composed by feature F2, or by feature F3 or by both of them. An or-group relationship is represented by a filled arc as shown in figure 2.4.

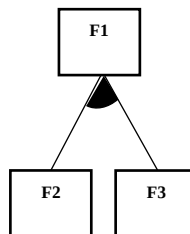


Figure 2.4: An or-group relationship.

And-group: if a feature F1 is selected in a configuration, then may be all of its subfeatures (F2..Fn) are selected too. In other words, a feature F1 is composed by feature F2 and feature F3 if these features are both compulsory as shown in figure 2.5.

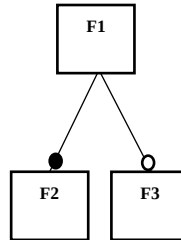


Figure 2.5: An and-group relationship.

2.1.2 Context Oriented Domain Analysis

The need to model SAS as system feature has led to new extensions to FMs in order to make them able to express and support SASs' context properties and requirements. These extensions, from different works, provided constraints and notations to the FM and its as follows.

Feature attribute: is a typed variable used to define additional characteristics of a feature [47]. In addition to many options supplied by features to a configuration, feature attributes allow the specification of more fine-grained selections of a configuration. Each feature can have a set of attributes.

Moreover, attributes might be allocate a type, for example, string, integer, a predefined enumeration) so the particular values can only originate from the type's domain. An example of feature attribute is the ambient Lighting feature in fig. 4.1. This feature has an attribute which represent the colors of light. the type of this feature attribute is enumeration that imposes it to be one of three values__ white, green and red.

Contextual Cross-tree constraints (CTCs): are Boolean formulas that are used to define additional dependencies among the features that are not defined by the structure of the FMs, feature attributes' expressions, or range of a feature's version [4].

Contextual cross-tree constraints are used because not each relationship between features can be represented by using the FMs' hierarchical structure. Besides, it is used in order to capture the environment's effect on the system.

A generic expression language is used to represent contextual CTCs. The language is capable to represent arbitrary propositional formulas by using contextual information, features, and their attributes as variables. The logical operators that are used in the formulas are: $\wedge$, $\vee$, $\leftrightarrow$, $\rightarrow$, and $\neg$. In addition, to constrain the domain of feature attribute, standard comparison operators can be used such as: $>$, $<$, $\geq$, $\leq$, $=$, and $\neq$.

Feature versions: to capture the different states of evolution of features, feature versions are included in the definition of FMs as arbitrary versions organized as a branching development history. A change in feature version states modifications the implementation of a feature to recognize evolved functionality—a user might prefer an older implementation of features over a newer implementation to maintain compatibility with other systems [48]. For example, in fig. 4.1, the feature *Connection Gateway* comes in two versions__ 1.0 version and 2.0 version.

Validity Formula (VF): which is a propositional formula that enforces the selection of a certain feature or feature's attribute depending on the context [4]. The propositional formula specifies the context under which a certain feature become selectable. For example, VFs are defined for features *eCall* and *EraGlonass* in fig 4.1. VFs restrict the feature *eCall* to exclusively be selected if the car location is Europe. Besides, feature *EraGlonass* to be selected if the car is in Russia. In this way, the effect of the context on the two features *eCall* and *EraGlonass* is straight way obvious to both__the developers and the users of the system.

Contextual information: a context can be defined as an object which has a name and it takes a value in a certain domain. Contextual information captures the environmental situation and external parameters, and is represented using identifier-value pairs [4]. Each context is assigned a value, and each value is connected to certain features in the FM via the validity formula (VF).

The impact of the contextual information on the system need to be modelled by the FM, by specifying under which situations the system must be reconfigured. This modelling can be done using two means: Validity Formula (VF), defines under which context a certain feature can be selected. As an example, in fig. 4.1, the geographical location of the car has been modelled as enumeration with the name *Location*, it has two literals values that represent wither the car is located in *Russia* or *Europe*

FMs also include the notion of user profiles [49] to allow the transformation of one configuration into another during run time in order to accommodate multiple users who might require different configurations of the same system under varying contexts, which will be discussed in section 2.3.

2.2 TEXTUAL REPRESENTATION OF FMs

The notation of an instance of a FM can be represented formally by interpreting the feature diagram into formulas using logic. In [50], authors were the first that connected feature diagrams with grammars. In [51], author produced a mapping technique in propositional formulas. In [52], author outlined the work to date the conversion of a feature diagram in propositional formulas besides, he presented a connection among feature diagrams, propositional logic, and grammars. moreover, the author proposed the using of logic truth maintenance system in order to analyze the propositional formulas based on the SAT solver [53].

In [54], an algorithm for extracting a propositional formula from a feature diagram and oppositely was presented. For more information about the interpretation/textual representation of feature diagrams, a comprehensive survey have been provided by [55]. In this work, we follow the notations used by [4].

Table 2.1 obtained from [4], shows the textual representation as a set of propositional constraint of a FM with its features and constraints using EBNF notations. Table 2.2 obtained from [4], shows the structure of Features of a feature diagram in propositional logic.

Table 2.1: the textual representation of a set of propositional constraint of a FM using EBNF notations [4].

Grammar of constraints.

```

constraint : boolean_expr;

boolean_expr : boolean_term (

('and' | 'or' | 'impl' | 'iff' | 'xor') boolean_term ) * ;

boolean_term : ('not')? ('true' | 'false' | relation) ;

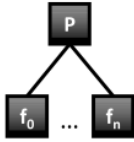
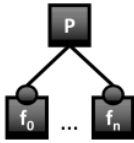
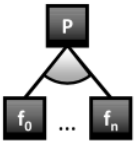
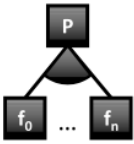
relation : expr

(( '<=' | '=' | '>=' | '<' | '>' | '!=') expr)? ;
expr : term (( '+' | '-' | '*' ) term)* ;
term : ('context[' | 'feature[' | 'attribute[') id ']' |

[-]?[0-9]+ | '(' boolean_expr ')' ; id : [a-zA-Z_][a-zA-Z0-9-
9_]*

```

Table 2.2: the structure of Features of a feature diagram in propositional logic [4].

HyFM	Translation
	$(\text{feature}[f_0] = 1 \text{ or } \dots \text{ or } \text{feature}[f_n] = 1)$ $\text{impl feature}[P] = 1$
	$\text{feature}[P] = 1 \text{ impl}$ $(\text{feature}[f_0] = 1 \text{ and } \dots \text{ and } \text{feature}[f_n] = 1)$
	$\text{feature}[P] = 1 \text{ impl}$ $(\text{feature}[f_0] + \dots + \text{feature}[f_n]) = 1$
	$\text{feature}[P] = 1 \text{ impl}$ $(\text{feature}[f_0] = 1 \text{ or } \dots \text{ or } \text{feature}[f_n] = 1)$

2.3 USER PROFILE

Multiple users might require different configurations of the same system under a specific context. To allow the transformation from one configuration to another during the run time, user profiles notion was introduced by [49].

User profiles composed of a set of preferences ordered in a linear way in a list according to their priority and importance. Where preferences are soft/weak constraint that can be satisfied if and only if no other higher priority preferences or constraint conflicts them.

Users are able to customize a configuration of the system by using preferences.

In addition, manufacturers of the system are able to customize a configuration as a sensible default by using preferences. For instance, the manufacturer has adjusted the latest version of *ConnectionGateway* as default to use. Meanwhile, the user prefer using the old version of *ConnectionGateway* as this feature is compatible with their smart devices. Moreover, assuming that the manufacturer has composed the following configuration preferences [4] with priorities:

- I. *GearAdvice*[2.0]
- II. *GearAdvice*
- III. *ConnectionGateway* $\rightarrow$ *ConnectionGateway*[2.0]
- IV. *AmbientLighting* $\rightarrow$ *AmbientLighting.Color* = *White*

In this default configuration, the newest version of *GearAdvice* ([2.0]) has to be selected. If the newest version of *GearAdvice* cannot be selected for any reason, then the next suggestion of the manufacturer is any version of the *GearAdvice*. To ensure the best connection among ECUs of the car system, the latest version of the feature *ConnectionGateway* has to be selected if it is possible.

Now assuming that the configuration preferences of the sporty driver [4] is as follows:

- I. $\neg \text{GearAdvice}$
- II. $\text{GearAdvice} \rightarrow \text{GearAdvice.style} = \text{Sport}$
- III. $\text{AmbientLighting} \wedge \text{AmbientLighting.Color} = \text{Red}$
- IV. $\text{ConnectionGateway} \rightarrow \text{ConnectionGateway}[1.0]$

Here, the *GearAdvice* feature of the car system is not desired by the sporty driver. Thus, the driver prefer to deselect this feature. But if the deselecting is not possible, then the driver prefer to select the *Sport* style feature. Selecting the *Sport* style feature require the selecting of the *AmbientLighting* feature with its color adjusted to *Red*. Furthermore, the driver prefer using the old version of *ConnectionGateway* [1.0] as this feature is compatible with their smartphone and if this feature is possibly selectable.

2.4 CONFIGURATION FEATURE

Configuration features comprise coherent features of a system along with a sorted collection of their relations and constraints to specify a given product in a SPL. It determines the software configurations that are constituted by elementary or configuration features along with their relations and constraints [21]. Figure 2.6, shows the FM of list software after adding a configuration feature.

For instance, a configuration *C* may selects a feature *F* if the feature *F* is a coherent component of the configuration *C*. A configuration *C* may requires a feature *F* as a default component (selected by FM's relations and constraints) if the feature *F* is associated to other features by the FM's relations and constraints. In others words, when the configuration *C* has to choose, without any constraint, between several variations of a feature, it prefers the default variation *F*.

Moreover, a configuration *C* may rejects a feature *F* if the feature *F*, for any reason, is not needed to be exist in the configuration *C*. A configuration *C* might change/evolve (metamorphism/transformation) to a configuration *C'* according to an event caused by its environment.

Configuration features might be reused under certain system conditions. Such reuse avoids the necessity to reconfigure the system for each change in the environment, consequently

reducing time, cost, and complexity. Yet, feature models are still not capable to express and represent such reuse process.

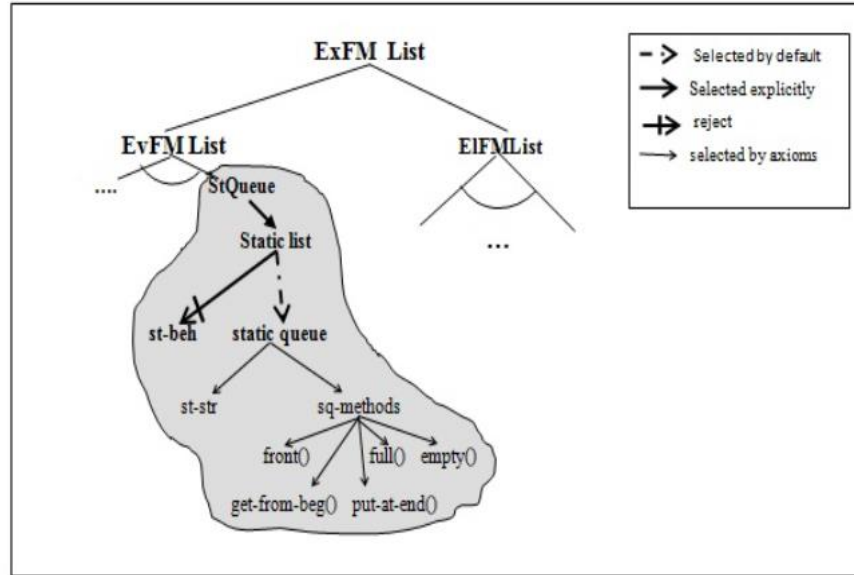


Figure 2.6: The FM of list software after adding a configuration feature [22].

2.5 SUMMERY

In this chapter, information about variability feature modeling with its two types__ feature oriented domain analysis (classical feature models) and context oriented domain analysis is provided. the earlier version of FODA and the Feature-RSEB extension with their features and features relations and constraints have been explained with the help of examples. Moreover, the new extensions that support context properties have also been explained. Further the interpretation of feature diagrams have been provided. At the end, information about user profile and configuration concepts is provided.

3. LITERATURE WORK

3.1 INTRODUCTION

Variability modelling is a substantial concept for formally modelling and representing complex systems including, among others, software product lines (SPLs), dynamic software product lines (DSPLs), self-adaptive systems. One of the most using techniques in modelling complex system is feature models (FMs), as it plays a main role in these systems. Nevertheless, the significance of representing and modelling the core of software systems and the problem domains is one of the major reasons behind continuity of these techniques and remain useful nowadays.

Moreover, FMs is one of the new forms of reuse that is able to bring new opportunities for the reuse process beyond the conventional software components [40].

However, the need to model such systems has brought about the need to make FMs more express and capable of representing self-adaptive systems, their context properties and requirements, and their reuse process.

In the following sections, firstly, important and relevant works in the variability feature modelling domain will be introduced. Secondly, approaches on self-adaptive systems and the modelling of these systems by feature model will be discussed. Finally, other works on system reuse, open problems and dissertation scope will be introduced.

3.2 FEATURE VARIABILITY MODELLING

Feature diagrams were firstly presented by [11] which was a part of Feature-Oriented Domain Analysis (FODA). Basically, FODA graph consists of Features that are nodes/strings organized in a hierarchal way. Features are either mandatory or optional, and arranged alternatively (only one feature is selected) or in groups (at least one feature is selected). The top node of the graph is known as root/concept. These nodes are either optional, represented by a hallow circle, or mandatory, represented without the hallow circle.

Relations among features are:

- a) Consist-of: *and* and *xor* decompositions.

b) Constraints: *requires* and *mutex*.

Figure 3.1 shows FODA feature model for a monitor engine system [56].

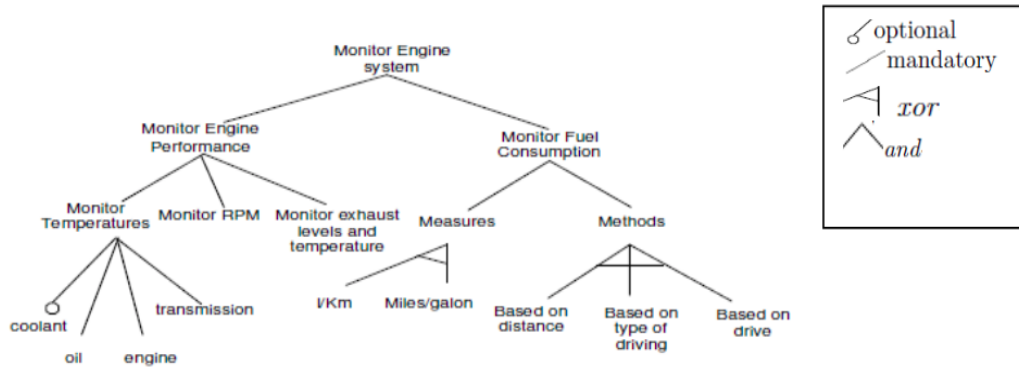


Figure 3.1: FODA: Monitor Engine System [56]

Since that time, several extensions have been made to feature diagrams by different authors to handle complexity and system's variability process in different domains, such as; Feature-RSEB [12], Cardinality-based feature diagrams [13], service feature diagrams SFD [14], Cardinality-based service feature diagrams CSFD [15], which is an extension work of [14] and [16], A Textual Design Methodology TDM [17].

In [14], a service feature diagrams (SFD) was presented. This work is based on the original feature model. It changed the basic framework of the original FM by presenting a new feature types along with their relevance. The abstract syntax of the SFD is represented by a metamodel. The SFD must have at least one feature. Root is a feature type that appears only once at the top of the FM. Each feature in the SFD must have a name and a type.

Concepts that are presented to the FM by this work are:

- I. Resource features: features that are used only one time.
- II. Sharable features: features that are used more than one time.
- III. Requestor and Provider: various choices of variable features become available by using the requestor or the provider in order to make a choice.

Figure 3.2 shows the produced SFD' concepts of the options of the manufacturer for the car entertainment system.

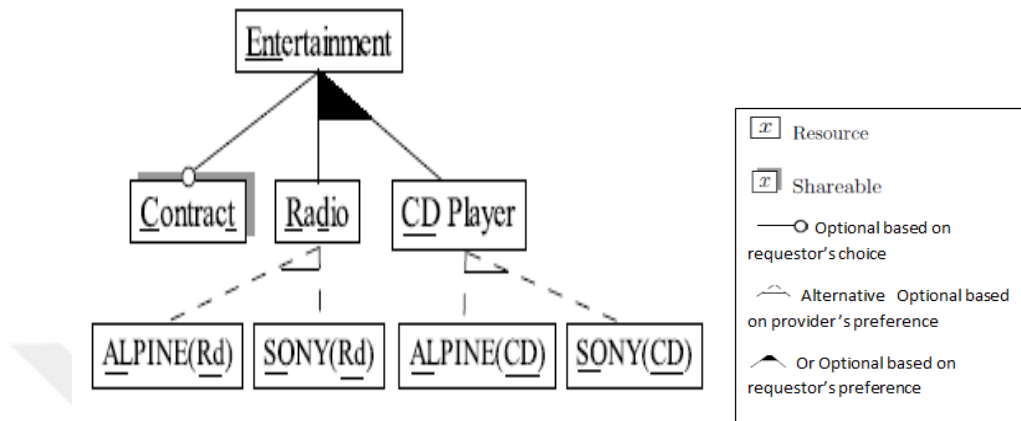


Figure 3.2: Manufacturer's Options for the Entertainment System of Car, [15].

In [17], A formal textual design methodology (TDM) that support the variability design and the implementation of software system was presented. It bridged the design model of product lines and the model of object oriented implementation. Moreover, a rich and briefed textual notations for the FM and class modelling was provided, besides, the class and feature concepts was combined in a simple way.

Concepts that are presented to the FM by this approach are:

- I. Meta-features Model: captures the significant characteristics of the feature modelling concepts, and categorizes the features according to their functionalities. This FM is a predefined design pattern that is able to define all the features in the methodology, and the features in the Features Meta-Model that base on it.
- II. Features Meta-Model design pattern: it is the input to the methodology and is composed of four kinds of features:
 - a) Features types.
 - b) Features Control.
 - c) Features Global.

d) Features Configuration.

Figure 3.3 shows the textual design methodology (TDM) that mix the feature and the class concepts.

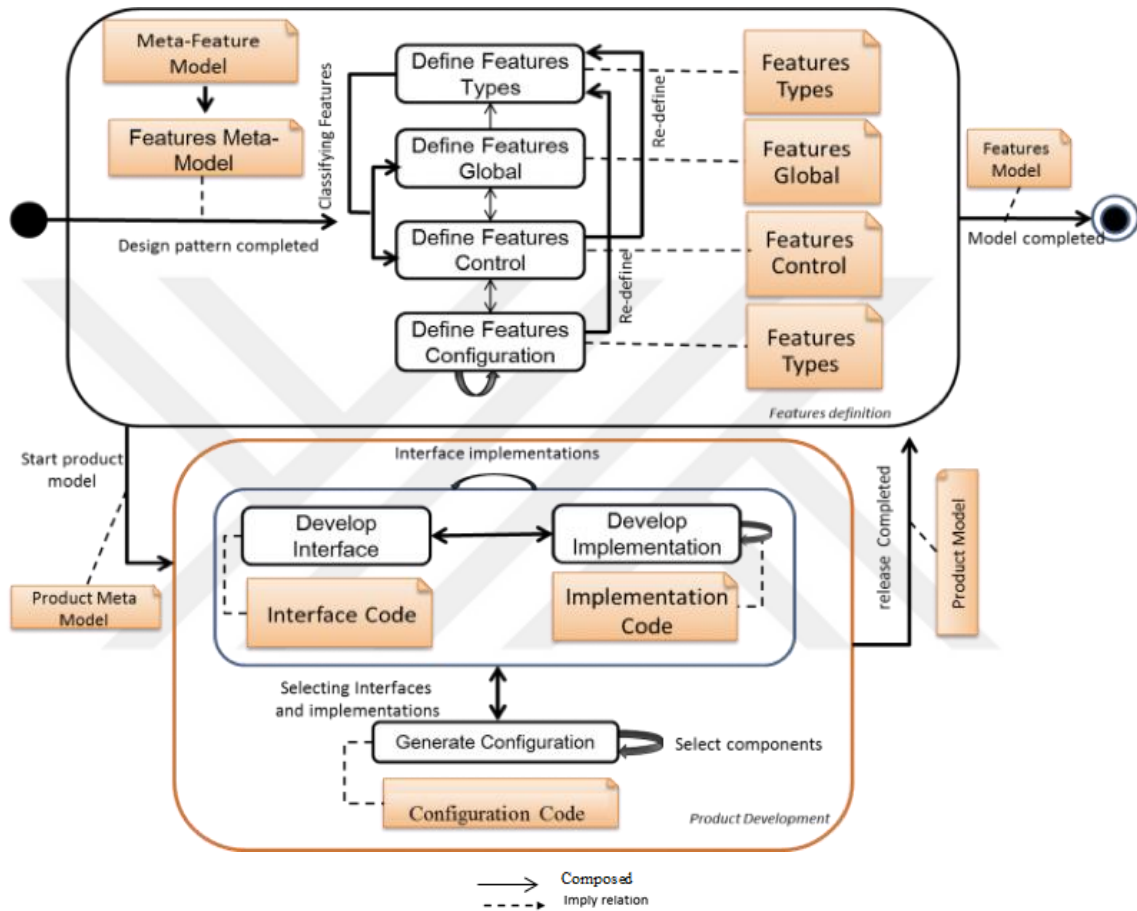


Figure 3.3: Textual Design Methodology (TDM) mixing class and feature concepts, [17].

As systems changes over time, inconsistencies with FM can arise, hence, many approaches were reported to deal with FM's inconsistencies by identifying them and providing solutions that help developers to fix defects (synchronizing changes made to other artifacts of the system in the FM) [57-60].

In [59], a dynamic-priority based approach that is interactively handling inconsistencies that might occur in the feature models was presented. For illustration of this work, assume that

we have a FM with its constraints and relations have been added except for this one: “C excludes D”. Where C and D are both features of the FM.

Now, after the operation of adding the excluded features, the FM will be inconsistent. In turn, the system will propose a recommended solution that is: removing the constraint (B require C), as it is the one with the lower priority. Where B and C are both features of the FM. The domain analysts will be the one who decide if the proposed solution is a satisfying one or not. If the recommended solution is not satisfying then the domain analysts will get the satisfiable solution by changing the priorities of the constraints. In this way, the system will propose a new solution according to the new constraints’ priorities.

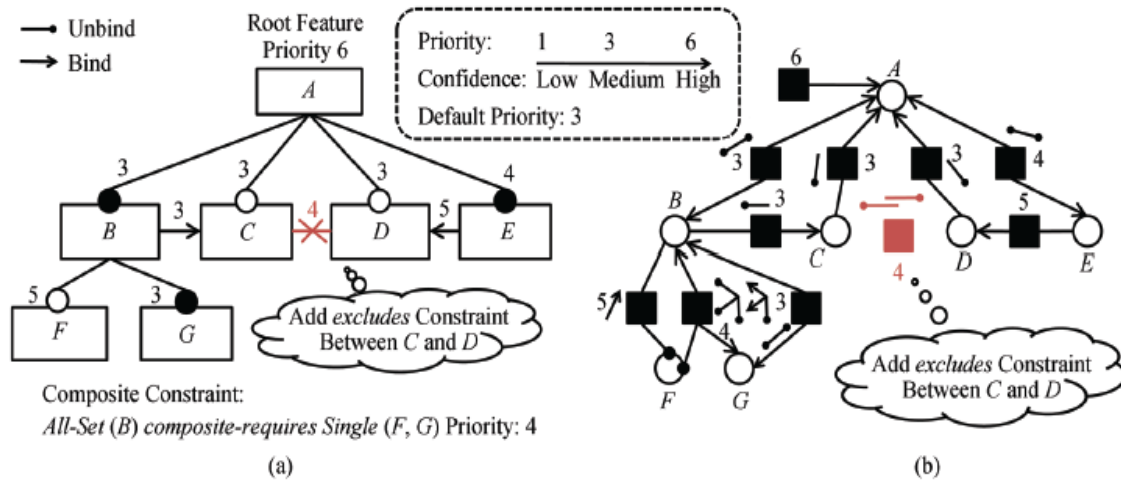


Figure 3.4: Example of dynamic-priority based In consistent Feature Models (IFM) fixing. (a) Example feature model. (b) Corresponding a locally-graph-better (LGB) method graph, for the example feature model in (a), [59].

Nevertheless, many research works produced to deal with feature selection [23-28]. The benefit of feature selection, among others, is to eliminate noisy, redundant or inappropriate data, decrease in data size, improve accuracy of prediction, extract important features, reduce the execution time and lead to better learning performance [29] and [30].

Certain works have focused on the management and improvement of FM customization (configuration), which is a popular method to reduce cost and improve user satisfaction [31-34]. However, the software configuration is yet to be modelled as a feature in the FM.

In [61], proposed a system of feature-based personalized recommender in order to guide the users through the process of decision-making and let them to focus on the proper and relevant parts of the configuration space. The system is based on the configurations from previous users in order to generate personalized recommendations for a current user.

In [34], a method for representing the FM as a two-layers model that involves the application layer and the infrastructure layer, with the ability to allocate the constraints among the features at the same level that is called “Inner Constraints”, beside, constraints among the features at varied layers that is called “Intra constraints”. This methodology has concentrate on addressing the problem of the FM configuration that is: the detection of differences among the requirements of the application and the obtainable abilities of the implementation platforms. Potentially, the implementation of a FM’s configuration need specific software and hardware infrastructures for example, operating system, database, and a hardware which cannot be obtainable by the stakeholders. Figure 3.5, shows the tow-layers FM.

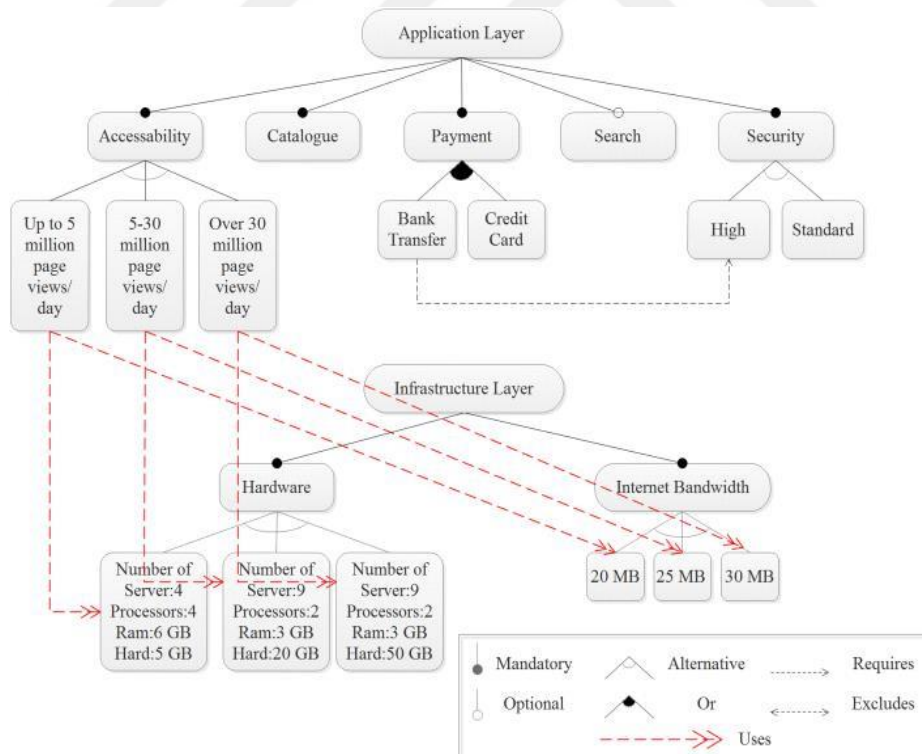


Figure 3.5: The tow-layers FM, [34].

Furthermore, several research works were produced to represent feature models in formal way [40-44]. In [40], authors presented a formal specification framework for the software product lines (SPLs). This framework used algebra process (SPLA) to express and describe the FODA diagram [21]. Further, the framework is general and flexible to express and describe other extensions of feature modelling diagrams.

In [41], an extension of the formal framework for algebra SPLA [40] is produced to describe more features such as cost-related. This extension supported the cost object and the comparison of production cost among various software product lines (SPLs) and it can be applied to different processes of decision making.

In [20] and [21], The elementary feature model and produced extended feature model were formalized by having two type of formalisms notations' combination. The EBNF notations is used to define the structure of the FMs while, the algebraic notations is used to define the FMs properties. Figure 3.6 shows formal specification of the elementary feature model structure using EBNF notations.

<EIFM> = “elementary feature model” <elementary feature model name>“;”
 <elementary features>
 “end” “elementary feature model” <elementary feature model name> “;”

Figure 3.6: The elementary feature model structure using EBNF notations [20, 21].

However, feature models need to be more expressive and have extensions in order to make them capable to model and represent self-adaptive systems developed with reuse process. Moreover, current FMs focus on representing and reusing elementary features without taking into consideration representing and reusing configurations features.

3.3 SELF-ADAPTIVE SYSTEMS

Bio-inspired modelling is the way used in supplying natural solutions for the artificial systems problems and enhancing the utilization of the computers. Hence the approaches of Bio-inspired computing are a main part of set of methods that support the style of natural computing [62-65].

Such approaches have been used for developing problem-solving techniques in wide areas, for instance, evolutionary computing [66], immuno-computing [67], neuro-computing (neural networks) [68], swarm intelligence [69], besides, in software modelling [1]. Yet, no direct and complete mapping between the computing concepts and the real world ones. Gap between the two worlds need to be reduced.

A new perspective of Bio-inspired Self-adaptive system including software evolution, context aware and self-adaptation has assumed as greater importance for handling complex systems [1].

Self-adaptive Systems (SASs) are capable of changing their behaviour, functions and structures according to the change in their environment, user requirements or the system itself. Figure 3.7, shows the conceptual architecture of SASs [6]. In other words, Self-Adaptive System (SAS), according to bio-inspired concept, is able to activate/deactivate its features (reconfiguration) or increase/destruction (adaptation from one configuration to another; Metamorphosis) at run time to meet the required scenario according to its conditions.

Systems are required to be provided with self-* properties in order to have the ability for executing self-adaptation concept [2]. Self-* properties [3] include, among others, self-configuration, self-healing, self-optimization, and self-protection.

There are several works attempt to implement SAS architectures that are able to adapt, make decisions and evolving during the run time, but still have lacks of implementing all required characteristic of SAS mechanisms [1].

One of the important aspects that must be considered when developing SAS is changing the system behaviour during the runtime without affecting the system itself. This can be achieved by counting on bio-inspired approaches [70-73], for example, Immunity model [74-76]. This model deals with biological self-healing, self-configuration and self-adaptation properties.

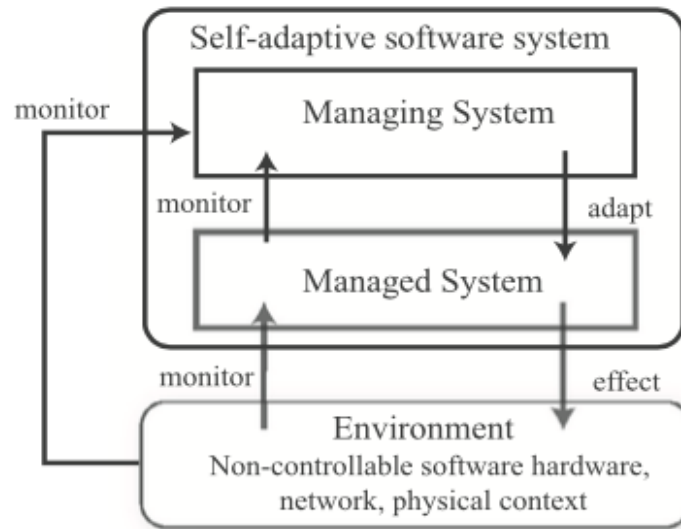


Figure 3.7: The conceptual architecture of SAS [6].

Many of large systems depend on Feature Models to represent their complex architecture, by recognizing, managing and reusing commonalities and variability (elementary features) among their products. Moreover, Feature Models play a chief role in many development projects [23] and [77].

The next section investigates and discusses the literature works related to Self-adaptive systems based on Feature Variability Modelling. Further, it aims to identify and mine, if there any, the required processes, features, their relation, and constraints ensuring their coherence that have been added to the FM in order to model and represent requirements, context properties, reuse process, and concepts of the SAS.

3.4 SELF-ADAPTIVE SYSTEMS MODELLED WITH FEATURE MODELS

The concept of adaption, making decisions and evolution is essential to systems, since user requirements and needs, environment, or the system itself change over the time, hence systems must adapt to remain useful [7]. Feature models (FMs) are extensively used to represent systems' variability [7].

Several works have been done on Feature Models to involve them with system adaptation process, by using the FM to model the variability and then using information provided by the FMs to identify the system configuration at runtime [78-86] and [35-38] but no notions or

extensions have been introduced into FMs to model these adaptation processes or their reused processes.

For example, in [81], an adaptation mechanism for mobile applications which is based on changes of the conditions was presented. In this work, the SPLs concepts is used for developing policies of self-adaptive for the mobile applications. It depends on three concepts of variability feature modelling in order to achieve the adaptation mechanism that are:

- I. Priorities of features: critical, important and useful.
- II. Availability of features: Activation and deactivation of the features.
- III. Features degradation: only changing the behaviours of mandatory features.
- IV. Further, authors used three feature models in order to define each kind of the features priorities.

In [82], a dynamic software product line (DSPL) methodology that support the reconfiguration policies at the run time for the mobile applications was presented. In this work, the FM is used to represent the abstraction of the applications variability. Besides, authors has generated configurations from the FM during the run time according to the current context by using algorithms of multi-objective evolutionary. Figure 3.8 shows the approach of dynamic reconfiguration.

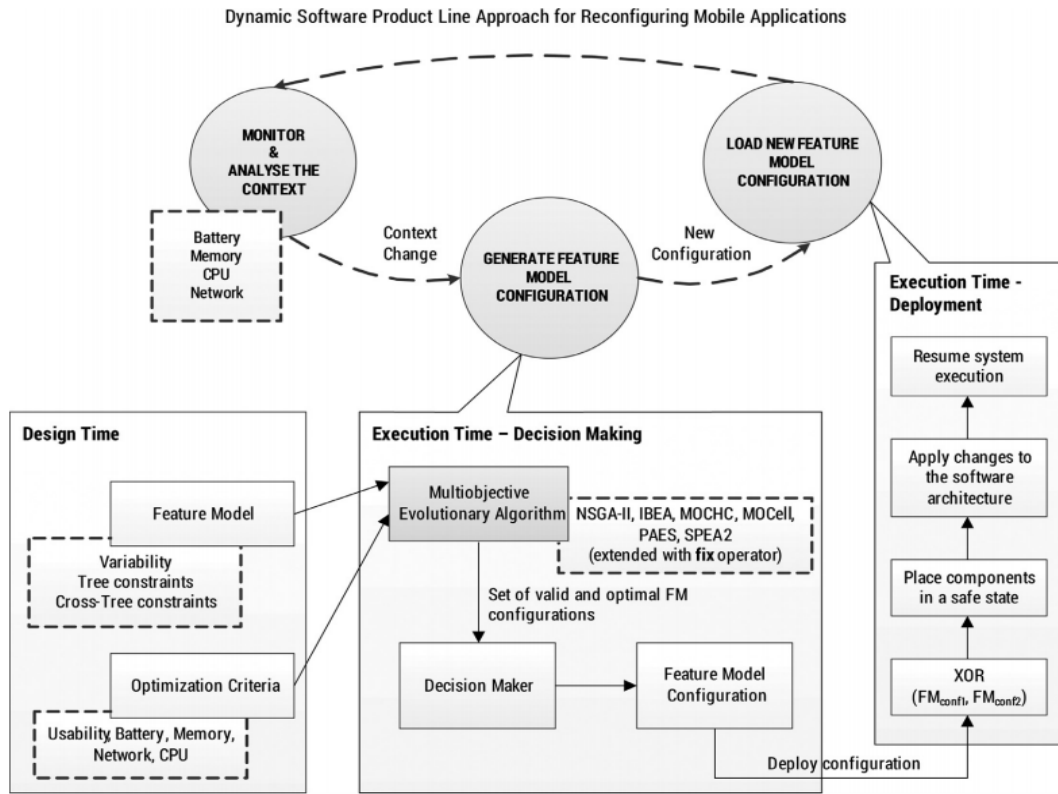


Figure 3.8: Dynamic reconfiguration approach, [82].

In [84], a method for designing and enhancing policies of self-adaptive systems by using multiple levels of the abstraction was proposed. Authors used FMs to model variability and design pattern to model reusability, and by combining variability and reusability in a single solution for reconfiguration, they could support the developing of self-adaptive systems (SAS) in a modular way. Figure 3.9 shows an over view of the proposed approach.

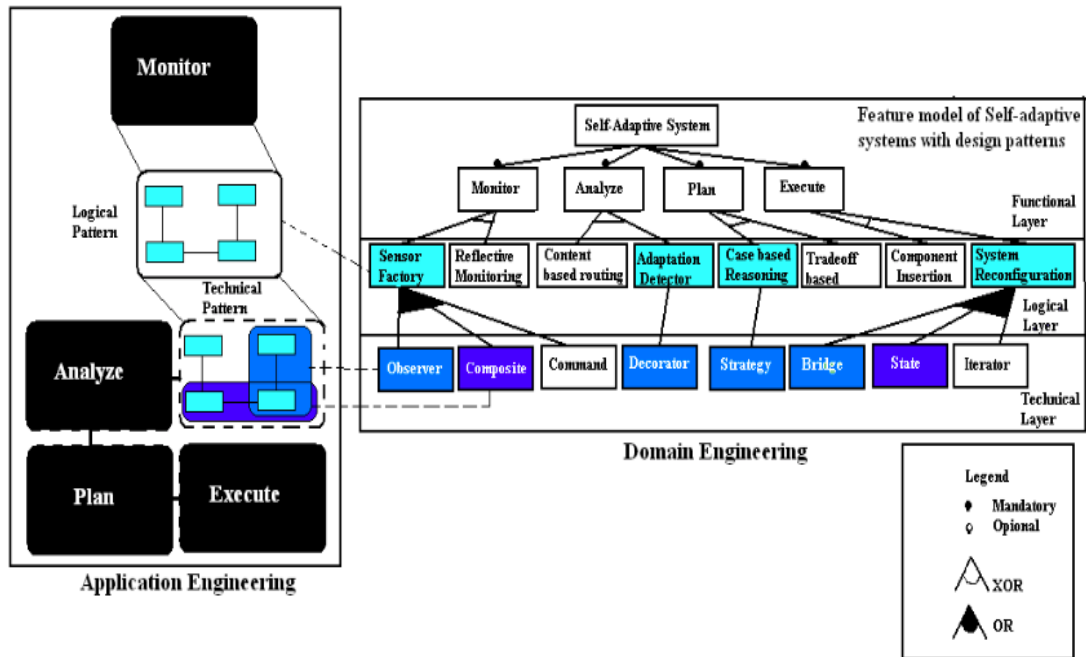


Figure 3.9: an over view of the proposed approach [84].

In [35], an idea of distributing a feature model of DSPLs by using the feature units' concept was presented to assist dynamic adaptation. Authors divided the FM into distributed and deployable sets of features, and deployed each set of features onto a distributed node. Nodes are able to communicate with each other, in the neighbourhood, and able to do matching and update process on its own feature unit, at whatever time, they find comparable and related feature units in their vicinity, as shown in figure 3.10.

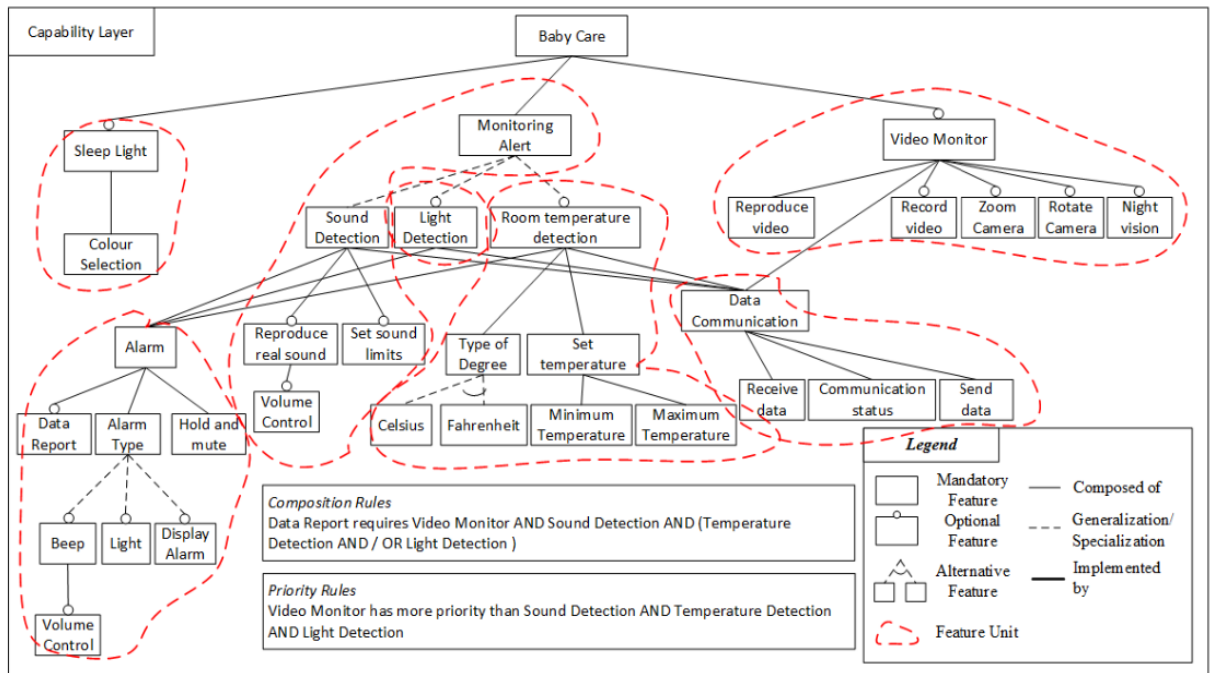


Figure 3.10: Partial feature model of the Baby Care System [35].

Furthermore, hybrid passive/active behaviour (Hy-PAB) architecture was introduced to make a feature unit capable of providing its services continuously, thus, providing availability of reconfiguration at any time. Figure 3.11 shows the Hy-PAB conceptual architecture.

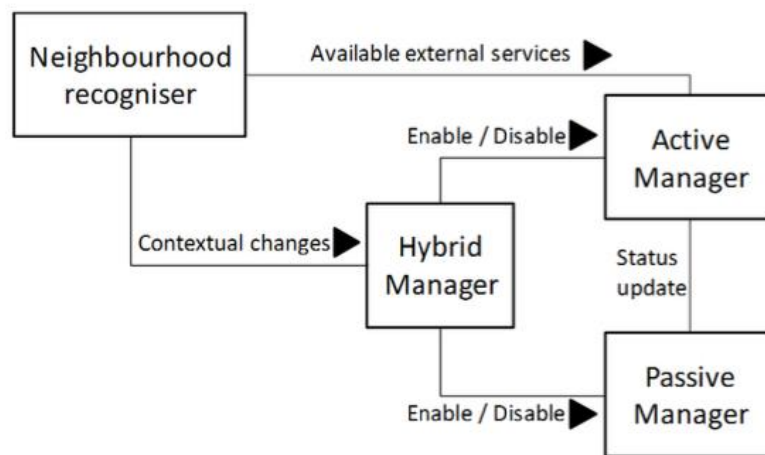


Figure 3.11: The Hy-PAB conceptual architecture [55].

In [36], a method for self-healing service composition reconfiguration was proposed. Features of a service compositions' family was represented by the FMs of SPLs. As FMs allow the definition of constraints on features, it guarantees that a configuration will perform a valid composition of service. Thus, allowing the service composition to go through adaptation process by selecting different subset of FM's features (configuration) that respects those constraints.

The presented method is basically comprised by two sub-processes; the first sub-process will do the calculating of each contribution of feature on non-functional properties of the composition of service, and then, it will pass it to the second sub-process, which in turn, will recognize the process of adaptation by making decision on which features should be part of the adapted composition of service. Fig. 3.12, shows the proposed method's architecture.

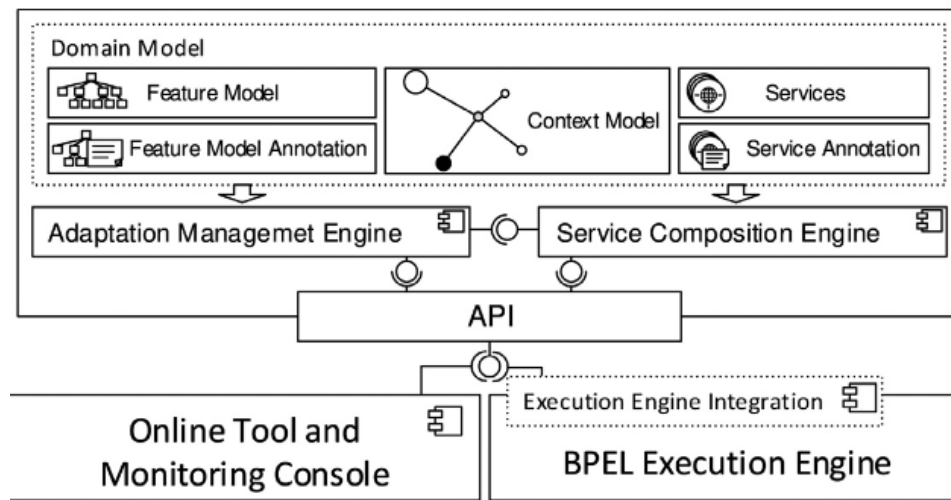


Figure 3.12: Architecture of the proposed method [36].

In [37], a method for switching among SLAM algorithms during the runtime based on reconfiguration and feature-based classification of SLAM algorithms was introduced. To perform the reconfiguration process at runtime, there must be a solution model for a feature model which maps all features in the FM to elements in the solution model. A modified model will be created during the mapping phase for the selected variant. Then, a configuration and a code will be generated from this selected variant. Thus, the generated code will be the

executable program that takes a place instead of the running model during the runtime. Figure 3.13, shows the concept for runtime reconfiguration.

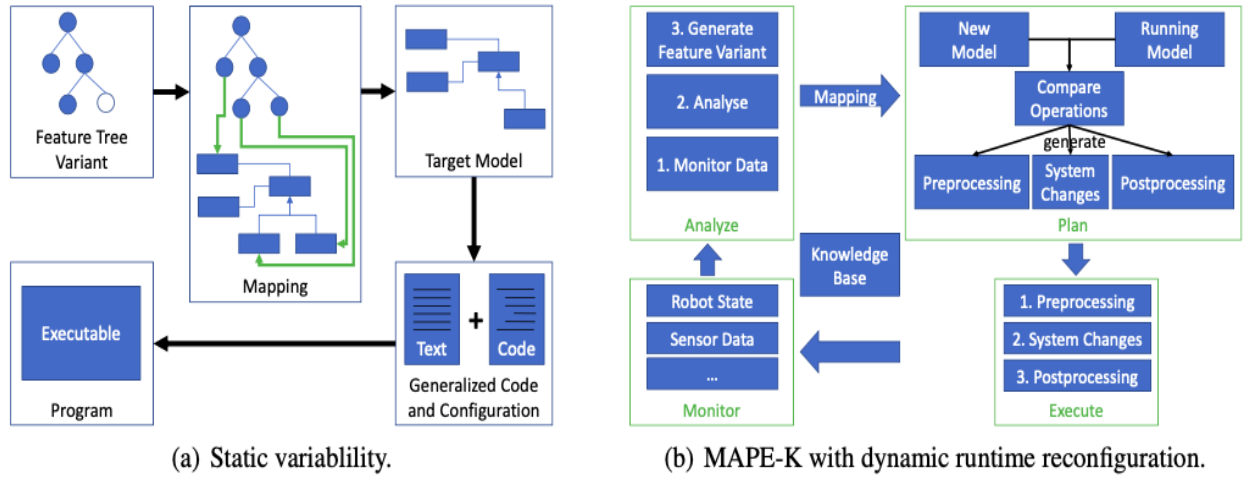


Figure 3.13: The concept for reconfiguration [37].

A few works have enriched the FMs by adding concepts and extensions to enable them to model SASs. The appended concepts, including elementary features, equipped with their relations, processes, and constraints, are designed to ensure coherence, in [87], a Temporal Feature Model (TFMs) was presented; an approach to model the FMs with their evolution as first-class entities. Figure 3.14, shows an example of features' temporal validities.

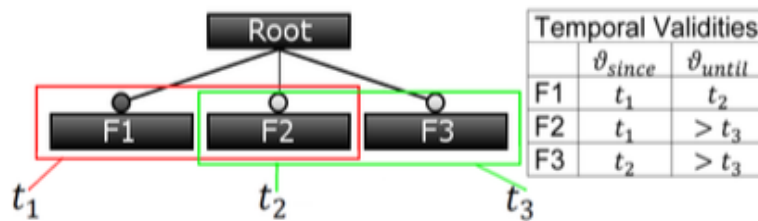


Figure 3.14: An example of temporal validities of features [87].

TFMs Supports the following concepts:

- I. Timespan concept; evolving elements are temporally valid for a limited timespan, which is a time interval relation, specified by the following two points: since, the start of the evolving element temporal validity, and until, is the end.
- II. Operations of evolution are: add, delete, move, change type, and rename. These evolution operations are applied on the elementary features.
- III. Group compositions process: supports the relocation of features and changing groups' compositions inside a configuration.
- IV. Cross-tree constraints concept for supporting creation and deletion of Feature children inside relocation groups.
- V. Cardinalities concept of features and groups for supporting the evolution of types of features and groups.

In one of the most significant works in the domain [4], based on their previous work [48], [88], [89] and [58], authors presented a framework for SPLs, which allow modelling of user customization, contextual influences and evolution (by Temporal Feature Models [58]). Particularly, a hybrid feature model (HyFM) was introduced, which is a combination of concepts of their previous works [48] and [88]. HyFM supports the following concepts:

- I. Feature attributes
- II. Contextual information
- III. Relations between contextual information and features selected according to current context is constrained by Validity Formulas (VFs).
- IV. Cross-tree constraint
- V. A finite set of feature versions.
- VI. Version aware constraints language.
- VII. Cardinality based.

Figure 4.3 (see chapter 4) shows the HyFM with the introduced concepts.

In [20] and [21], a feature-based variability modelling approach for representing software evolution was produced to support evolution and variability concepts. It extends the traditional feature model with concepts of evolution, by building an extended FM from the traditional FM. Figure 3.15, shows the methodology of feature-based variability modelling for supporting software evolution.

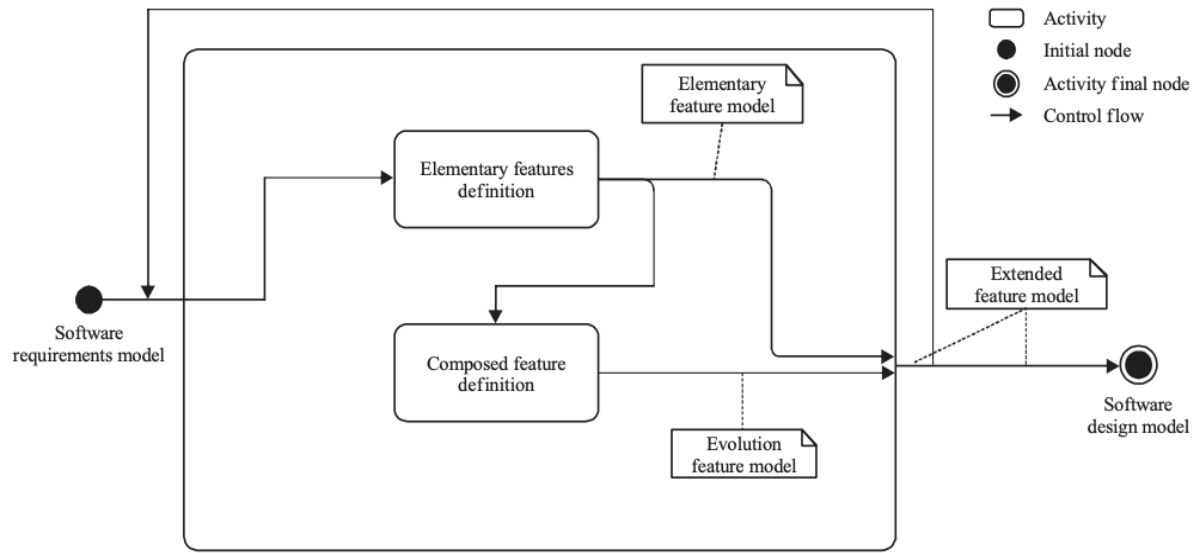


Figure 3.15: The methodology of feature-based variability modelling in software evolution [21].

The presented concepts are mainly:

- I. Composed feature concept (configuration) along with its relevant evolution relations, these evolution relations enable the feature model to support feature grouping and transforming/metamorphosing.
- II. Evolution relations: three types of select are presented__explicit select, axiomatic select and default select.
- III. Evolution constraint: reject, and at-age.
- IV. Evolution processes: Add, Delete and change.

Further, in [39] Authors extended FMs with binding time constraints extension in order to restrict DSPL staged reconfiguration processes. This extension has made DSPLs applicable

for applications that require safety-real-time-critical domains. Figure 3.16, shows the feature model of device control unit (DCU) extract with binding time notations.

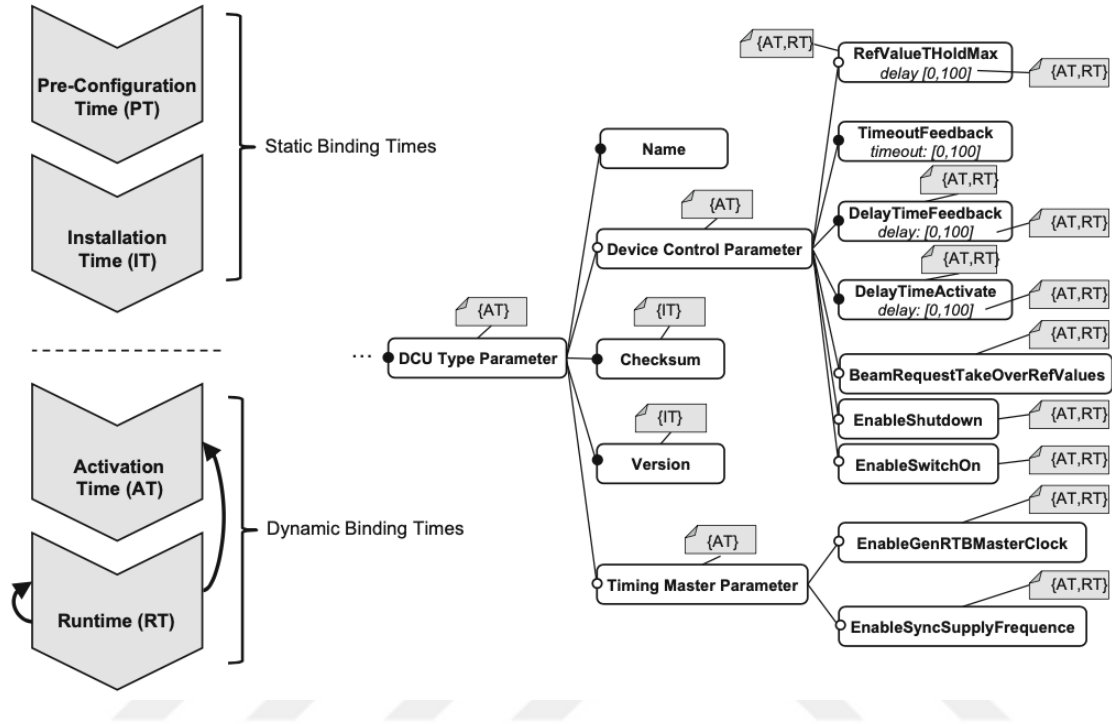


Figure 3.16: The device control unit (DCU) feature model extract with binding time annotations [39].

Concepts included in this extension of FMs are:

- I. Configuration stages to define arranging between features and differentiate between dynamic binding times and static ones of features.
- II. Complex binding time constraints to restrict product configurations processes taking into consideration the logical consistency and arranging limitations of a feature selections.
- III. Stateful reconfiguration models to control potential sequences of configuration changes of dynamic features during the life cycle of a product.

All of the aforementioned works have focused on modelling and reusing elementary features without considering the management of configuration feature and its reuse process, except for [21], in which the authors modelled the concept of metamorphosis, which involves the

grouping of features (configuration). To this end, they modelled the configuration using an FM, but did not model the reuse process. However, these approaches were not designed to support management of configurations features and their reuse.

3.5 SYSTEM REUSE

Krueger [90] defined the software reuse as “the process of creating software systems from existing software rather than building software systems from scratch”. Author lists abstraction, specialization, selection, and integration as core actions for any reuse approach. Considering and supporting such activities during developing techniques and strategies is a key into successful software system reuse.

Authors in [91] explains that the domain engineering (DE) is the best choice for reuse process since it increase quality and productivity. DE develops reusable artifacts of domain specific. Then, these developed artifacts are reused by engineering processes of application to produce specific applications [92]. For example, SPLs are vastly adopted systematic reuse techniques and strategies as they are set of software families that share commonalities (elementary features) and developed with mass-customization and reuse platform [92].

In [93], authors presented the automated Reusable Service Identification (ReSIde) approach. This approach identified reusable services depending on the analyzing a collection of similar object-oriented software product variants (SPVs). ReSIde approach is able to identify reusable functionalities of systems cloned codes which can be located as services over multiple software product distincts based on analyzing the commonalities and the variability among the source code of these software product distincts.

Complexity can be decreased in the development of systems by reusing components or elements. Moreover, the development of the system itself can be fasten and less error-proneness by reusing components and processes. FMs is one of the new forms of reuse that is able to bring new opportunities for the reuse process beyond the conventional software components [40].

Reusability has previously been applied to SAs for example: Rainbow strategy [41], MADAM framework [42], MUSIC framework [43], and ENTRUST strategy [44]. However, none has produced a design or developed a self-adaptive system with reuse process.

Moreover, the formal modeling of a configuration reuse process or reused configurations remains lacking, especially with respect to system feature models (FMs).

In [45] the authors provided an architecture of reusable communication with utilizing of templates of abstract component. This architecture reduced the development time and enabled an easier adaptation during runtime. Figure 3.17 shows the reusable adaptation logic (AL) model. However, this work is not formally modelled by system feature modeling.

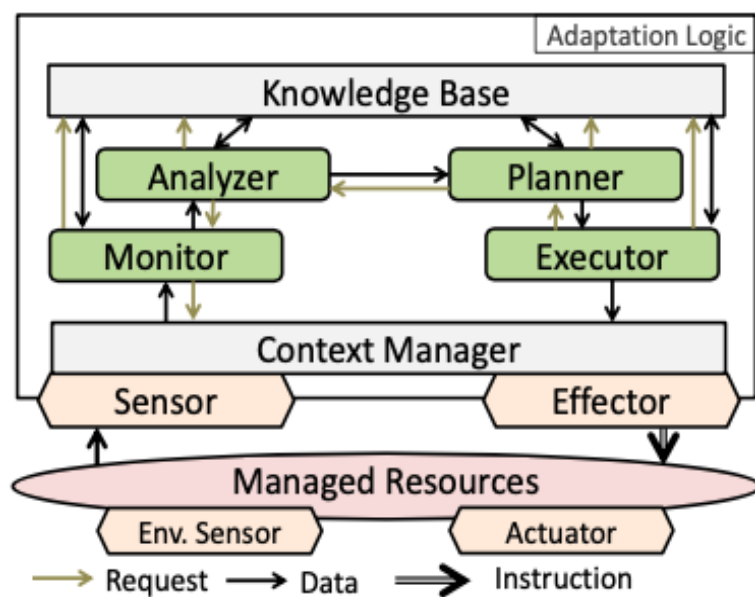


Figure 3.17: shows the reusable adaptation logic (AL) model [45].

In [6], an autonomic software product line engineering (ASPLe) methodology was presented. This methodology allows developers to develop SASs with reuse processes by selecting, specializing, and subsequently, integrating reusable artifacts. Figure 3.18 shows the ASPL strategy and the ASPLe methodology. This methodology used FMs to model only the scope definition of ASPLe; hence, no concepts were further added to the FM to support the modeling of the developed SAS with reuse.

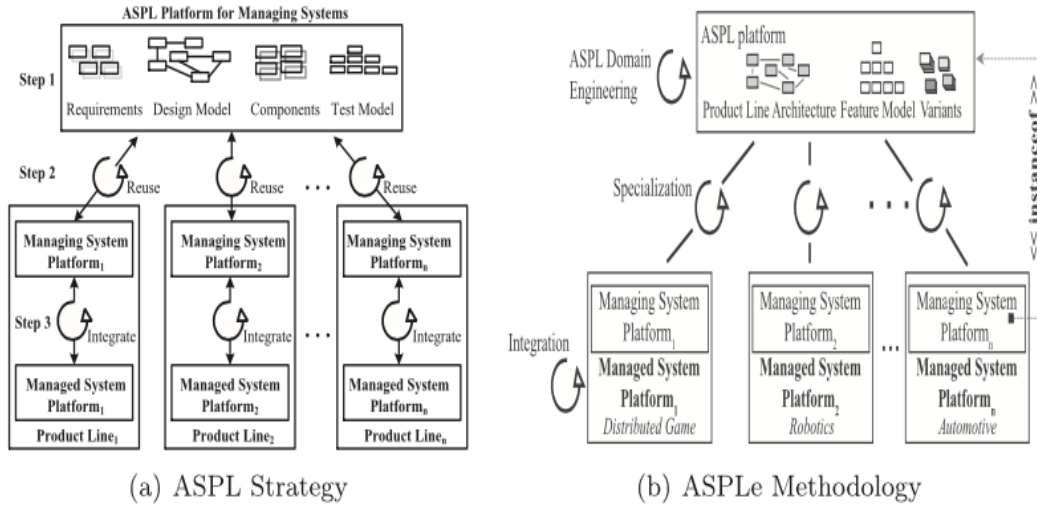


Figure 3.18: The ASPL strategy and the ASPLe methodology [6].

In [46] an automated test reuse for highly configurable software was presented. Authors automatically reused existing manually developed tests in configurable system to produce new test variants.

3.6 RESEARCH GAP

The compound of the runtime variability of self-adaptive systems and the reuse process could lead to complex area. This complexity is significantly caused by uncertainty. Where uncertainty stands for the things that are unknown or imprecise, for example, the system requirements and the operating environments at a particular point in time [94]. Uncertainty impacts on the system design and other actions of development. Current reuse works remedy the uncertainty in the reuse process, mainly with the domain modeling which does reduce the scope for the reuse process, thus uncertainty itself.

Mainly, there are two sources of uncertainty when self-adaptive system is developed [6]. The first source (which is out of the scope of this work) is deficiency of knowledge. Knowledge about the application domains and target applications is either not available or available to a limited extent. Hence, developers are undetermined about the goals of the stakeholders and deployment environments.

The uncertainty's second source is the runtime variability in self-adaptive systems and their context. Total knowledge about runtime variations in the requirements of a system, the system itself and its environment usually is not obtainable at the design time. Due to this insufficiency of knowledge, the analysts and designers of the system will be facing uncertainty in the requirements and the design specifications. In addition, the runtime variations are difficult to expect at the design time, and even if expected, there is no assurance that the variations are customized (configured) precisely.

3.7 OPEN PROBLEMS AND THESIS SCOP

In order to reduce complexity of self-adaptive systems developed with reuse process and decrease uncertainty, formal variability modelling of the self-adaptive system itself, its context properties, and its stepwise configurations as features and their reuse process must be supported by using the feature model .

Feature models need to be more expressive and have extensions in order to make them capable to model and represent self-adaptive systems developed with reuse process. Moreover, current FMs focus on representing and reusing elementary features without taking into consideration representing and reusing configurations features. Hence, they are not designed to support self-adaptive systems developed with reuse process, management of configurations features, and their reuse process.

This work aims to deal with the above challenges by introducing an extension to the feature model with new features, context information, and constraints to ensure coherency between features. This extension empower the FM to support the modelling of the configuration features and their reuse process. Configuration feature management process has been defined to manage the configuration features and their reuse process from its creation to its destruction. In addition, it defines all the states (manufacturing, composing, reusing and saving) throughout the lifetime of a configuration feature. Two types of configuration features are supported: *default configuration feature* and *custom defined configuration feature*.

Moreover, two type of configuration features' reuse has been introduced: *direct reuse* and *indirect reuse*. These configuration features will be reused every time it fulfills the requirements of a new context during the run time.

3.8 SUMMERY

In this chapter, firstly, a collection of the most relevant variability feature modelling approaches that are reported in the literature is provided. The first version of FMs, FODA, is explained. In addition, extensions that have been made to feature diagrams by different authors to handle complexity and system's variability process in different domains is discussed.

Secondly, information about self-adaptive systems (SASs) is provided. Then, the important works of SASs that modelled with FMs techniques is discussed. We discussed that FMs are not capable to model and represent self-adaptive systems developed with reuse process. Moreover, current FMs are not designed to support self-adaptive systems developed with reuse process, management of configurations features, and their reuse process.

Thirdly, information about system reuse is given and some relevant approaches in the domain is explained.

Finally, research gab between self-adaptive systems and system reuse is explained. then, open problems and thesis scope is discussed. In addition, we discussed that the combination of the runtime variability of self-adaptive systems and the reuse process could lead to complex area. Hence, SASs and their reuse process need to be formally modelled by FMs techniques.

4. CONFIGURATION FEATURE MODELLING AND REUS EXTENSION

This chapter introduces the configuration features concept and their reuse process, an extended form of the hybrid feature model (HyFM) [4]. The motivation behind the implementation of this extension is to enable the FM to formally model SASs developed with reuse process; modelling its context properties, and its stepwise configurations as features and their reuse process, thus reducing SASs' complexity.

In this chapter, firstly, challenges raised by modelling SASs developed with reuse process in section 4.1. In section 4.2 an illustrative example is presented to provide information about the concepts introduced in this work. In sections 4.3, definitions of the proposed configuration features' extension along with its configuration features, relations, and constraints to ensure their coherency, which is able to address the challenges is presented with the help of an example. In section 4.4, the configuration features' concept management including the proposed framework, its process and states will be presented with the help of examples. In section 4.5, the case study is presented. whereas in section 4.6 the chapter is concluded.

4.1 MODELLING SASs DEVELOPED WITH REUSE PROCESS

The perspective of self-adaptive systems (SASs) including software evolution, context-aware/ context-dependent, dynamic software product line (DSPL), and self-adaptation are increasing and used more and more these days. For example: autonomous cars, wearable devices and smart homes. The need to model such systems as system features has led to the need to represent and model their context properties as a part of the process of domain analysis (DA) [40].

Many of large-scale systems including SAS use feature models (FMs) to represent their complex architecture [7], by managing, expressing and reusing commonalities and variability information among their products.

Complexity can be decreased in the development of systems including SASs by reusing components or elements. Moreover, the development of the system itself can be fasten and less error-proneness by reusing components and processes. FM is one of the new forms of

reuse that is able to bring new opportunities for the reuse process beyond the conventional software components [40].

Reusability has previously been applied to SASs for example: Rainbow strategy [41], MADAM framework [42], MUSIC framework [43], and ENTRUST strategy [44]. However, none has produced a design or developed a self-adaptive system with reuse process. Moreover, the formal modeling of a configuration reuse process or reused configurations remains lacking, especially with respect to system feature models (FMs).

For example, the DSPL Car system [4], illustrated in section 4.2 and shown in figure 4.3, is equipped with two emergency call systems—one for Russia called *ERAGlonass* and another for Europe called *eCall*.

Assuming that the driver drives the car frequently between Europe and Russia. Now if the car is in Europe, then the reconfiguration process will be triggered to create a configuration that is responsible for emergency call in Europe, as shown in figure 4.1.

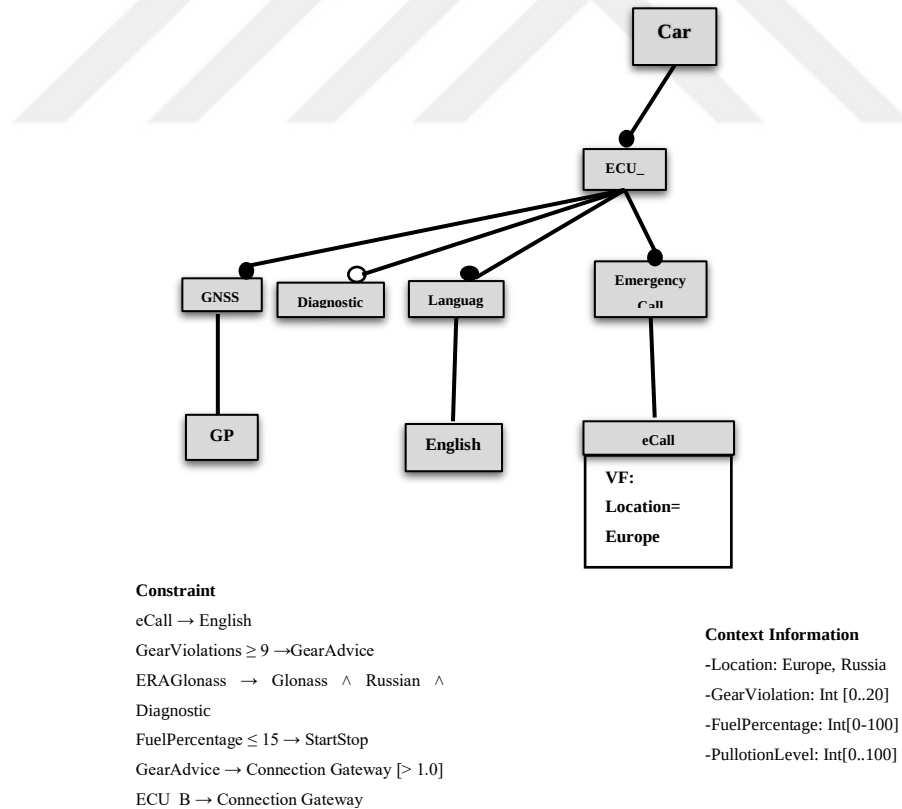


Figure 4.1: a configuration that is responsible for emergency call in Europe.

When the driver drives the car to Russia, the reconfiguration process will be triggered to create a configuration that is responsible for emergency call in Russia, as shown in figure 4.2. This scenario will be repeated each time the driver drives the car between Europe and Russia because the DSPL Car system is not developed with reuse process. Which means, the reconfiguration engine will be triggered each time the car changes its location to create the same configuration for the same context (either for Europe or for Russia).

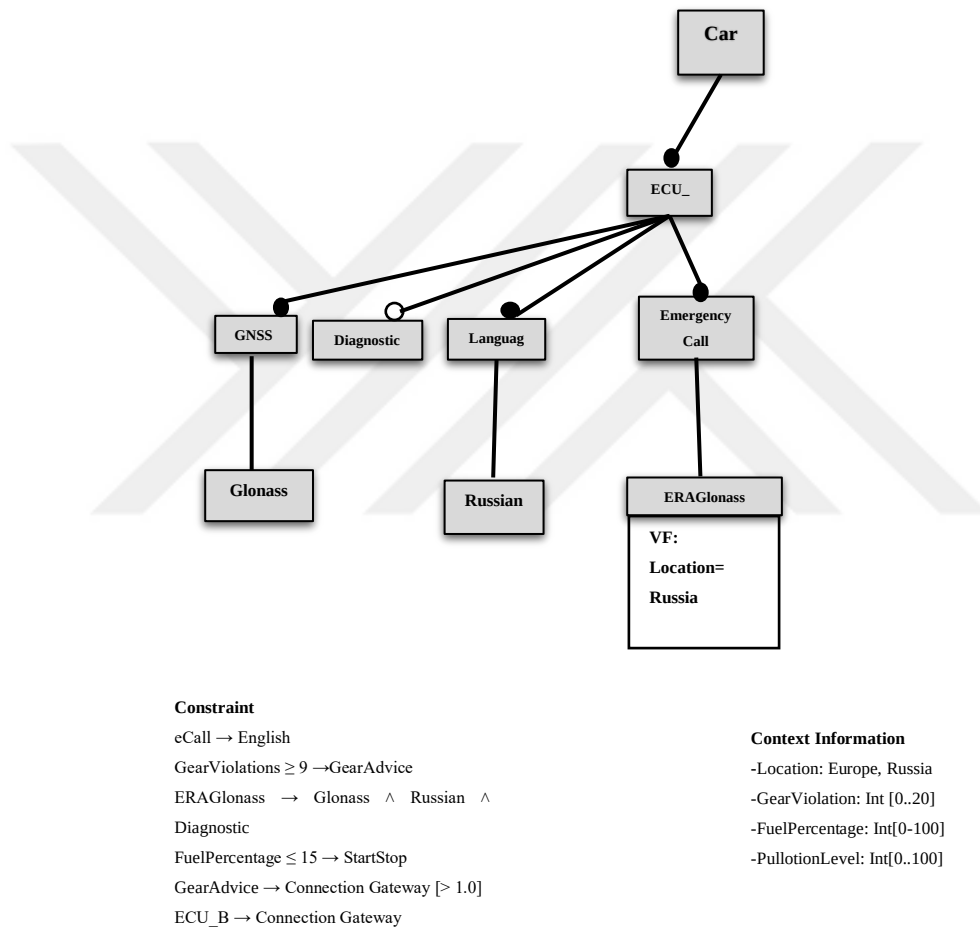


Figure 4.2: a configuration that is responsible for emergency call in Russia.

On the other hand, assuming that we have a SAS that is developed with reuse process, and reuses it's configuration for the same context instead of recreating them. Yet, the FM is not able to express and model neither the configurations nor their reusing process, since current FMs focus on representing and reusing elementary features without taking into consideration

representing and reusing configuration features. In conclusion, FMs are not designed to support self-adaptive systems developed with reuse process, management of configurations features, and their reuse process.

To deal with the above challenges, we propose an extension to the feature model with new features, context information, and constraints to ensure coherency between features. This extension empower the FM to support the modelling of the configuration features and their reuse process.

4.2 ILLUSTRATIVE EXAMPLE

In this section, an example of a car system, which is a dynamic software product line (DSPL), obtained from [4], is presented to illustrate the concepts introduced in this work. Figure 4.3 [4] depicts the feature model of a car system comprising two electronic control units (ECUs)—a mandatory ECU_A and an optional ECU_B. ECU_A is responsible for the emergency call system of the car—it triggers a preconfigured call with the satellite position of the car and additional relevant data when an emergency is detected.

The car considered in this study is equipped with two emergency call systems—one for Russia called *ERAGlonass* and another for Europe called *eCall*. The *ERAGlonass* feature requires selecting the following features: *Glonass* (positioning service in Russia), *Diagnostic* (sending extra diagnostic data) and *Russian*. The *eCall* feature requires the feature *English*, and one of *Glonass* or *GPS* features can be selected. The *ConnectionGateway* feature provides communication functionality for other ECUs to be able to communicate with the ECU_A.

The optional ECU_B provides three features that can be selected according to the driver desires. These features are: *StartStop*, *GearAdvice* and *AmbientLighting*. To maximize fuel and performance efficiency, the *GearAdvice* feature, offered by the optional ECU_B, suggests a suitable gear to use based on three options for driving style—*family*, *natural*, or *sports*. The feature *AmbientLighting* specifies the light's color in the cabin of the car that are: White (the standard mood), Green (power saving and calm mood) or Red (the powerful mood).

Four types of context information are captured using independent sensors.

- I. *Pollution*: represents the number of contaminants in the air.
- II. *Location*: captures the current position of the car.
- III. *GearViolation*: captures the number suboptimal uses of the gearshift by the driver.
- IV. *FuelPercentage*: captures the remaining fuel level.

The decision of selecting features of the car system depends on the values of these context. For instance, selecting between the features *eCall* or *ERAGlonass* depends on the current location of the car, which is either Europe or Russia.

Users of the car system are able to customize a configuration of the system by using preferences. In addition, manufacturers of the car system are able to customize a configuration as a sensible default by using preferences. For example, the manufacturer has set the latest version of *ConnectionGateway* as default to use. Meanwhile, the user prefer using the old version of *ConnectionGateway* as this feature is compatible with their smart devices.

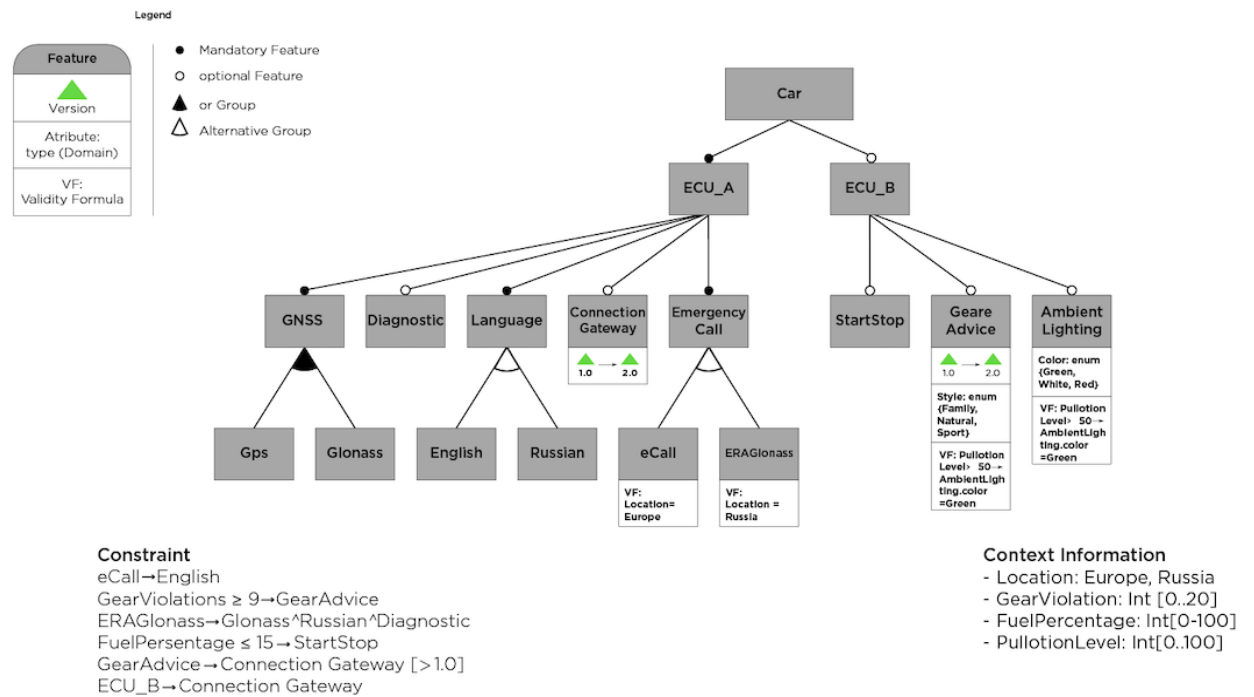


Figure 4.3: Feature model for a car system [4].

4.3 MODELLING OF CONFIGURATIN FEATURES

4.3.1 Enhanced FM With Configuration – Example

The example presented in Figure 4.4 presents a novel FM and the updates introduced to the FM, as depicted in Figure 4.3, by the introduction of the new configuration feature. This example (Figure 4.4) will be used in this section to illustrate the introduced concepts that support management of configuration features and its reuse process.

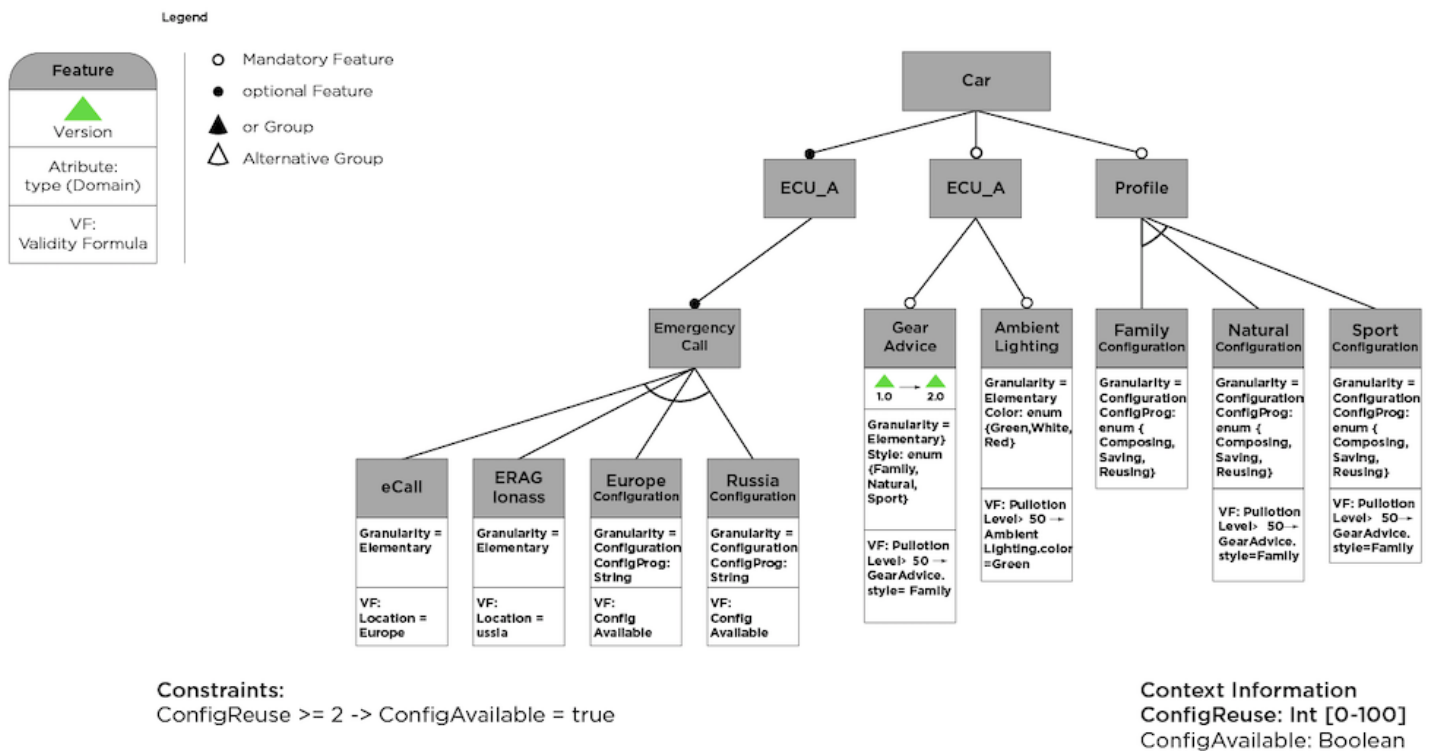


Figure 4.4: Enhanced feature model for a car system with configuration features.

4.3.2 Definitions

The introduction of configuration features necessitates the definition of the following new concepts (depicted in Figure 4.4):

- I. A *ConfigReuse* sensor that capture the number of times a reconfiguration process is triggered to achieve the same configuration. When the same configuration is created 2 times, it is saved for future reuse.
- II. A *ConfigAvailable* sensor that capture the availability of a configuration. When a configuration is saved, the value of this sensor is set to be true.
- III. A *Granularity* attribute that fixe the feature type (elementary or configuration feature).
- IV. A *CofigProg* attribute contains the selection program text that generate a particular configuration when triggered.
- V. A *Configuration* feature that is a coherent composition of several features specifying a given product in a SPL.
- VI. A *Configuration feature management process* that manages a configuration feature from its creation to its destruction.

4.4 CONFIGURATION FEATURE MANAGMENT

4.4.1 Configuration Feature Framework

Configuration feature is a coherent composition of several features specifying a given product in a DSPL/SPL. This feature is modeled similarly to other elementary features in a feature model. It can have versions, attributes, or validity formula, with the exception of having a child feature, and it is controlled by FM constraints.

For example, *EuropeConfiguration* is a child configuration feature inside a group that contains elementary features (*ecall*, *ERAGlonass*) and configuration features (*EuropeConfiguration*, *RussiaConfiguration*), as depicted in Figure 4.4.

A configuration feature is modeled using the following framework.

$\langle \text{Configuration feature} \rangle \rightarrow \langle \text{Configuration feature name} \rangle;$
 $\langle \text{versions} \rangle$
 $\langle \text{attribute} \rangle = (\langle \text{attribute name} \rangle: \langle \text{attribute domain} \rangle)^*$
 $\text{"VF:"} (\langle \text{validity formula} \rangle)^*$

Figure 4.5: Configuration feature framework.

4.4.2 Configuration Feature Management Process

The configuration feature management process manages a configuration feature from its creation to its destruction. It defines all the states throughout the lifetime of a configuration feature. These states (depicted in Figure 4.6) are:

- a) *Manufacturing*
- b) *Composing*
- c) *Saving*
- d) *Reusing*

The layout of the configuration feature process is as follows.

4.4.2.1 Configuration feature process: life cycle

→ manufactured configuration

→ configuration creation

manufactured configuration → Default Configuration

configuration creation → Composing

generated by default v defined configuration → Saving

Saved configuration → Reusing

$\text{Reusing} \wedge \text{ConfigAvailable} \rightarrow \text{needed_configuration_exists} \wedge \text{Reusing}$

$\text{Reusing} \wedge \text{ConfigAvailable} \rightarrow \text{needed_configuration_does_not_exist} \wedge \text{Composing}$

$\text{Reusing} \wedge \text{disposed configuration} \rightarrow \text{exit}$

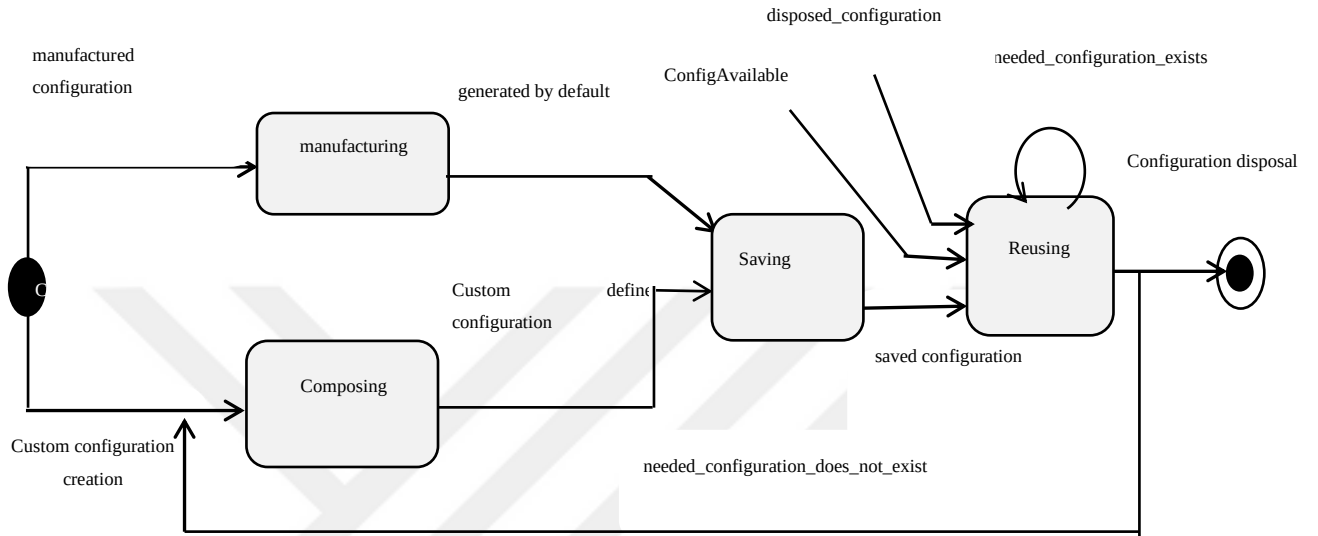


Figure 4.6: State diagram of the configuration process.

Two types of configuration features are supported:

- default configuration feature*: that is generated by the manufacturer to be available for reuse by the user of the system. for instance, preferences configurations have to be customized by the user of the car system each time he/she desires a specific feature/s or if a different driver drives the car. This process could be difficult to some of the users or tedious to others, instead, they can reuse the default configuration feature (depicted in Figure 4.4).
- custom defined configuration feature*: that is explicitly composed by a customer or the system depending on a particular context.

The HyVarRec [4] reconfiguration engine is responsible for creating and validating these configurations. The HyVarRec engine is able to successfully create new valid configurations that are suitable for the current context and maximize the specified user profiles. In addition, it is able to handle SPLs/DSPLs that are up to thousand features in a time that is less than a

minute [4], for more information about the HyVarRec reconfiguration engine see appendix A.

The created configuration is saved within the assets database to be reused every time it fulfills the requirements of a new context. The states of the two types of configuration features (depicted in Figure 4.6) are as follows.

4.4.2.2 Composing state

This state represents the creation of the two type of configuration features—the *default configuration feature* by the manufacturer, and the *custom defined configuration feature* by a user or the system according to an event. In addition, it represents the change that could be made to some of the configurations as they may need to add or delete a feature/s according to their changing environment. Where an event could be creation or contextual events.

For instance, assuming that the driver is driving frequently between Europe and Russia, then the frequent used configurations that is responsible for setting off emergency call systems of the car according to its location will be created and saved in the asset in order to be reused instead of triggering the reconfiguration process each time the car changes its location. Creation of the configuration feature named *EuropeConfiguration* that is responsible for setting off emergency call system in Europe will be shown in the next example 4.4.3.1. The composing state is specified as follows.

$$\begin{aligned} \text{Composing} \rightarrow & [\text{"Defined configuration"} \text{ } \langle \text{configuration name} \rangle] \wedge [\text{"ConfigProg"} \{ \\ & \text{"enabled features"} \text{ } (\langle \text{enable feature} \rangle)^* \vee \text{"disabled features"} \text{ } (\langle \text{disable} \\ & \text{feature} \rangle)^* \}]^+ \\ & [\text{"VF:" } (\langle \text{Formula} \rangle)^*] \end{aligned}$$

Figure 4.7: Composing state framework.

4.4.2.3 Saving state

This state represents the saving process of the created configurations features in the asset in order to be ready for reuse. For instance, the created configuration feature *EuropeConfiguration* will be saved in the asset after updating the FM by adding this new created configuration feature to the FM. Saving the configuration feature named *EuropeConfiguration* will be shown in the next example 4.4.3.1. The saving state is specified as follows.

```
Saving → update <FM> ^ update <assets>

update <FM> →      insert_feature<configuration name> : [<version>*, <attribute>*,
<VF>*]

update <asset> → save_in_DB<configuration name>
```

Figure 4.8: Saving state framework.

4.4.2.4 Reusing state

The state, *IndirectReuse*, corresponds to the state of a configuration feature that might be reused as a component of another configuration feature (integration of a configuration feature with another to serve a specific context). For instance, the preference configurations for driving style (sports, natural, family, and emergency) may be saved using this state as the emergency configuration might reuse one of the configuration features__sport, natural, or family configurations in order to adapt to the context of the emergency. This type of reuse is introduced alongside the *imply* and *exclude* constraints.

Alternatively, a configuration feature might be reused directly, corresponding to the *DirectReuse*. For instance, triggering the reconfiguration process each time the car changes its location from Europe to Russia and vice versa, is a frequent event and it can be modelled as a *DirectReuse* configuration feature. In such a case, keeping the appropriate configuration

feature available to reuse each time the frequent event occurs decreases the effort and time required by the reconfiguration process. Reusing the configuration feature named *EuropeConfiguration* will be shown in the next example 4.4.3.1. The reusing state is specified as follows.



Figure 4.9: Reusing state framework.

4.4.3 Configuration Management – Examples

4.4.3.1. Creation of the europeconfiguration feature

In this example, we assume that the driver location is in Europe, *Location = Europe*. The number of times the reconfiguration engine HyVarRec has triggered to reconfigure the same configuration that suits the current context (Car is in Europe) is 2 times, hence the context element *ConfigReuse = 2*. This prompts the configuration feature management process to compose this configuration as the configuration feature *EuropeConfiguration*. Subsequently, the FM is updated using DarwinSPL tool [89], for more information about the tool see appendix B, and then the configuration feature *EuropeConfiguration* will be saved in the assets, and the value of the sensor *ConfigAvailable* is set to *true*, *ConfigAvailable = true*. Now the configuration feature *EuropeConfiguration* is ready to be reused as long as it fulfil the requirements.

The explanation of the steps and the text program of the composing, saving, and reusing states is shown in the following example.

Composing state

In this state, the configuration feature *EuropeConfiguration* will be defined as follows.

First, the configuration name will be defined, which is *EuropeConfiguration*. Second, the group of features with features attributes (if there any) that composes the configuration feature will be selected (enabled)___ *eCall*, *English*, *Diagnostic*, and *GPS*. Third, feature/s that must be rejected for any reasons will be disabled, in this example there are no features to be disabled, *disabled feature: null*.

Finally, VF will be specified and since the configuration feature still in its first state and has not been saved yet, hence it is not available for reuse in this state. The VF will be set to: *not ConfigAvailable*.

The text program of the composing state for the configuration feature *EuropeConfiguration* is as follows.

Composing State

Composing →

Defined configuration

EuropeConfiguration

ConfigProg

enabled features

1.*enable eCall*

2. *enable English*

3.*enable Diagnostic*

4.*enable GPS*

disabled feature

null

VF: not ConfigAvailable

Saving state

In this state, the steps of saving the configuration feature *EuropeConfiguration* will be as follows.

To save the configuration feature made by the composing state, the saving state will first update the FM (HyFM) and assets by adding the new composed configuration feature to them. To update the FM, the configuration feature *EuropeConfiguration* will be inserted in the FM.

First the configuration feature's name will be inserted : *insert_featureEuropeConfiguration*. Second, versions of the configuration feature will be inserted: *Version = 1*. Third, attributes of the configuration feature will be inserted, in this example the configuration feature has two attributes:

- I. *Granularity = Configuration*.
- II. Configuration program ConfigProg with the selected and rejected features:

ConfigProg = [enable : {eCall, English, Diagnostic, GPS}
Disable: {}].

Fourth, VF will be specified. In this example, since the configuration feature has been saved, which means it is available for use after this state. The VF will be set to: ConfigAvailable.

Finally, the configuration feature *EuropeConfiguration* will be saved in the database.

The text program of the saving state for the configuration feature *EuropeConfiguration* is as follows.

Saving State

Saving $\rightarrow$ *update HyFM* $\wedge$ *update asset*

update HyFM $\rightarrow$

insert_featureEuropeConfiguration

[Version = 1

Granularity = Configuration

ConfigProg = [enable : {eCall, English, Diagnostic, GPS}

Disable: {}]

VF: ConfigAvailable]

update asset $\rightarrow$ *save_in_DB EuropeConfiguration*

Reusing state

In this state, the steps of reusing the configuration feature *EuropeConfiguration* will be as follows.

The reusing process will be triggered if the context element *ConfigAvailabe* is set to true and it will work as follows.

First, the *search_configuration()* function will take a place to search for an available configuration feature that suit the current context. If the needed configuration is exist (*needed_configuration_exist*), then the reusing process will start and the reused configuration feature will be reused until the context is changed (the configuration feature no longer fulfills the requirements).

Second, if the *search_configuration()* function could not find a suitable configuration feature for the current context (*needed_configuration_not_exist*), then the system will return to the composing state to create a configuration feature that suit the current context.

Third, if the configuration feature is no more needed (does not fulfill the requirements), the reusing state will stop reusing this configuration feature and it will be disposed.

In this example, The reusing process will be triggered as the context element *ConfigAvailable* is set to true. The configuration feature *EuropeConfiguration* is suitable for the current context, hence the *search_configuration()* function will return its name as the needed configuration: *search_configuration(EuropeConfiguration)*. The *needed_configuration_exist* will return true and the reusing process of the configuration feature *EuropeConfiguration* will be start and continue until the context is changed.

The text program of the reusing state for the *EuropeConfiguration* configuration feature is as follows.

Reusing State

Reusing $\wedge$ *ConfigAvailable* $\rightarrow$
 search_configuration(EuropeConfiguration)
 $\wedge$ *needed_configuration_exist* $\wedge$
 configuration_reuse(EuropeConfiguration)

Reusing $\wedge$ *ConfigAvailable* $\rightarrow$
 search_configuration(EuropeConfiguration)
 $\wedge$ *needed_configuration_not_exist* $\wedge$
 Composing

Reusing $\wedge$ *disposed_configuration* $\rightarrow$ *Exit*

4.4.3.2 Creation of the sporty driver profile feature

In this example, we assume that the manufacturer will generate the three driving styles of the car system—family, natural, and sports—as configuration features (*FamilyConfiguration*, *NaturalConfiguration*, and *SportConfiguration*). Here we will give an example of manufacturing the sports driving style of the car system as the configuration feature *SportConfiguration*.

The explanation of the steps and the text program of the composing, saving, and reusing states is shown in the following example.

Composing state

In this state, the configuration feature *SportConfiguration* will be generated as follows.

First, the configuration name will be defined, which is *SportConfiguration*. Second, the group of features and attributes (if there any) that composes the configuration feature will be selected (enabled)___ *GearAdvice.style = Sport*, *AmbientLighting*, *AmbientLighting.Color = Red*, and *ConnectionGateway*.

Third, feature/s that must be rejected for any reasons will be disabled, in this example, we assume that the *GearAdvice* feature of the car system is not desired by the sporty drivers, *disabled feature: GearAdvise*.

Finally, VF will be specified. In this example, since the configuration feature still in its first state and has not been saved yet, hence it is not available for use in this state. The first VF will be set to: *not ConfigAvailable*. The second VF is specified to ensure that the configuration feature *SportConfiguration* fulfils the requirements for the current context: *PullotionLevel > 50* $\rightarrow$ *GearAdvice.style = Family*. When the pollution level became more than 50, the *SportConfiguration* configuration feature does not remain operational, as it does not fulfil the requirements of the current context and the system will change the driving style to the family driving style: configuration features *FamilyConfiguration*.

The text program of the composing state for the *SportConfiguration* configuration feature is as follows.

Composing State

Composing →

Defined configuration

SportConfiguration

enabled features

1. *enable GearAdvice.style = Sport*
2. *enable AmbientLighting*
3. *enable AmbientLighting.Color = Red*
4. *enable ConnectionGateway*

disabled feature

1. *disable GearAdvise*

VF: not ConfigAvailable ^

(PullotionLevel > 50 → GearAdvice.style = Family)

Saving state

In this state, the steps of saving the configuration feature *SportConfiguration* will be as follows.

To save the configuration feature made by the composing state, the saving state will first update the FM (HyFM) and assets by adding the new composed configuration feature to them. To update the FM, the configuration feature *SportConfiguration* will be inserted in the FM.

First the configuration feature's name will be inserted : *insert_feature SportConfiguration*. Second, versions of the configuration feature will be inserted: *Version = 1*.

Third, attributes of the configuration feature will be inserted, in this example the configuration feature has two attributes:

- I. *Granularity = Configuration.*
- II. Configuration program ConfigProg with the selected and rejected features:

ConfigProg = [enable :{ *GearAdvice.style = Sport, ambientLighting,*
AmbientLighting.Color = Red, ConnectionGateway}
Disable: { GearAdvise }].

Fourth, VF will be specified. In this example, since the configuration feature has been saved, which means it is available for use after this state. The first VF will be set to: ConfigAvailable. In addition, The second VF is specified to ensure that the configuration feature *SportConfiguration* fulfils the requirements for the current context:

PullotionLevel > 50 $\rightarrow$ *GearAdvice.style = Family.*

Finally, the configuration feature *SportConfiguration* will be saved in the database.

The text program of the saving state for the *EuropeConfiguration* configuration feature is as follows.

Saving State

Saving $\rightarrow$ update HyFM $\wedge$ update asset

update HyFM $\rightarrow$

insert_featureSportConfiguration

[Version = 1

Granularity = Configuration

*ConfigProg = [enable :{ *GearAdvice.style = Sport, ambientLighting**

AmbientLighting.Color = Red,

ConnectionGateway}

Saving State (continued)

Disable: { GearAdvise }}

*VF: ConfigAvailable ^PullotionLevel> 50 → GearAdvice.style =
Family]*

update asset → save_in_DBSportConfiguration

Reusing sate

In this state, the steps of reusing the configuration feature *SportConfiguration* will be as follows.

The reusing process will be triggered if the context element *ConfigAvailabe* is set to true and it will work as follows.

First, the *search_configuration()* function will take a place to search for an available configuration feature that suit the current context. If the needed configuration is exist (*needed_configuration_exist*), then the reusing process will start and the reused configuration feature will be reused until the context is changed (no longer fulfill the requirements).

Second, if the *search_configuration()* function could not find a suitable configuration feature for the current context (*needed_configuration_not_exist*), then the system will return to the composing state to create a configuration feature that suit the current context.

Third, if the configuration feature is no more needed (does not fulfill the requirements), the reusing state will stop reusing this configuration feature and it will be disposed.

In this example, The reusing process will be triggered as the context element *ConfigAvailabe* is set to true. The configuration feature *SportConfiguration* is suitable for the current context, hence the the *search_configuration()* function will return its name as the needed configuration: *search_configuration(SportConfiguration)*. The *needed_configuration_exist*

will return true and the reusing process of the configuration feature *SportConfiguration* will be start and continue until the context is changed.

The text program of the reusing state for the *SportConfiguration* configuration feature is as follows.

Reusing State

```
Reusing ^ ConfigAvailable →
    search_configuration(EuropeConfigurationSportConfiguration) ^
    needed_configuration_exist ^ configuration_reuse(SportConfiguration)
```

```
Reusing ^ ConfigAvailable → search_configuration(SportConfiguration) ^
    needed_configuration_not_exist ^ Composing
```

```
Reusing ^ disposed_configuration → Exit
```

4.5 CASE STUDY

The tool, DarwinSPL [89] is used to study the feasibility of the illustrative example (discussed in Sections 4.2 and 4.3), for more information about DarwinSPL see appendix B. First, the three driving styles of the car system—*family*, *natural*, and *sports*—are generated by the manufacturer as configuration features (*FamilyConfiguration*, *NaturalConfiguration*, and *SportConfiguration*) under the control of the configuration feature management process. They are designated to be group members with the feature *profile* as the parent feature, as demonstrated in Section 4.3. Further, the three configuration features are saved within the assets database. They remain available for reuse depending on the context. Figure 4.10 depicts a screenshot of the FM following the creation of the three driving styles, modeled using the FM editor of DarwinSPL.

Firstly, assuming that the driver selects the *SportConfiguration* feature as his/her preferred configuration, this configuration will be reused as long as it fulfills the requirements of the

current context. For instance, if *PollutionLevel* > 50, then the *SportConfiguration* feature does not remain operational, as it does not fulfill the requirements of the current context. Hence, the HyVarRec reconfiguration engine will replace it with the *FamilyConfiguration* feature, as determined by the VF and CTC of the FM. Moreover, another person driving this car might change the preferred configuration to either the *NaturalConfiguration* feature or *FamilyConfiguration* feature based on his/her preferences.

Secondly, assuming that the driver drives the car frequently from Europe to Russia, the reconfiguration engine will be triggered each time he/she changes his/her location in order to provide a suitable configuration for the current context [4]. For instance, when the driver is in Europe, the reconfiguration process is triggered to compose a configuration for this context (*Location=Europe*) and the value of the context element sensor *ConfigReuse* is set to 1, as outlined in Section 4.3.

Now, if the driver drives to Russia and back to Europe, the reconfiguration process is triggered again to compose a configuration for the same context (*Location=Europe*) and the value of the sensor, *ConfigReuse*, is set to 2. This prompts the configuration feature management process to save this configuration as the *EuropeConfiguration* configuration feature within the assets database, and the value of the sensor *ConfigAvailable* is set to *true*. Subsequently, the FM is updated using DarwinSPL.

Figure 4.11 depicts the results of the updated FM with the configuration features, modeled using the FM editor of DarwinSPL. The textual representation as a set of propositional constraint of the FM with its features and constraints using EBNF notations is used to illustrate the modeling of the newly added concepts; contextual elements, validity formula, and constraints to ensure coherency between configuration features of the new FM's extension, are modeled using DarwinSPL.

Figure 4.12 depicts the contextual elements of the FM, modeled using the contextual information editor of DarwinSPL, figure 4.13 depicts the VF of the FM, modeled using the VF editor of DarwinSPL, and figure 4.14 depicts the constraints of the FM, modeled using the constraints editor of DarwinSPL. The results shows by these three figures, 4.12, 4.13, 4.14, demonstrate the coherency and applicability of every new added elements of the produced extension.

The introduced configuration features are reused based on the context until the context changes; e.g., the driver drives from Europe to Russia. When this happens, other configuration features of the system are reused depending on the context until they are to be disposed. This procedure enables the obtainment of reliable results by expending considerably less effort over considerably shorter durations of time, as the reconfiguration process does not need to be triggered as many times due to the utilization of the reuse process.

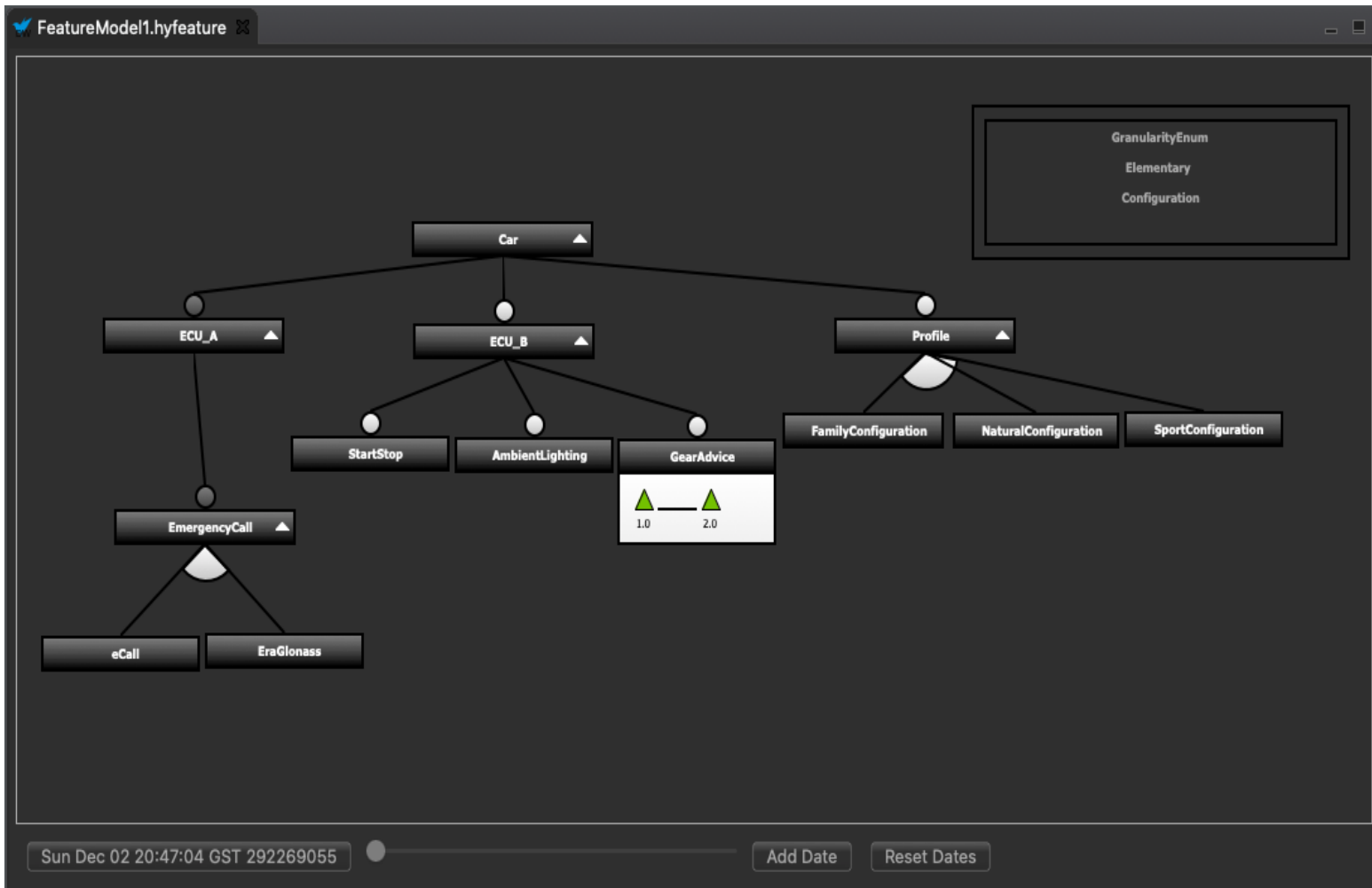


Figure 4.10: Screenshot of the FM following the creation of the three driving styles, modeled using the FM editor of DarwinSPL.

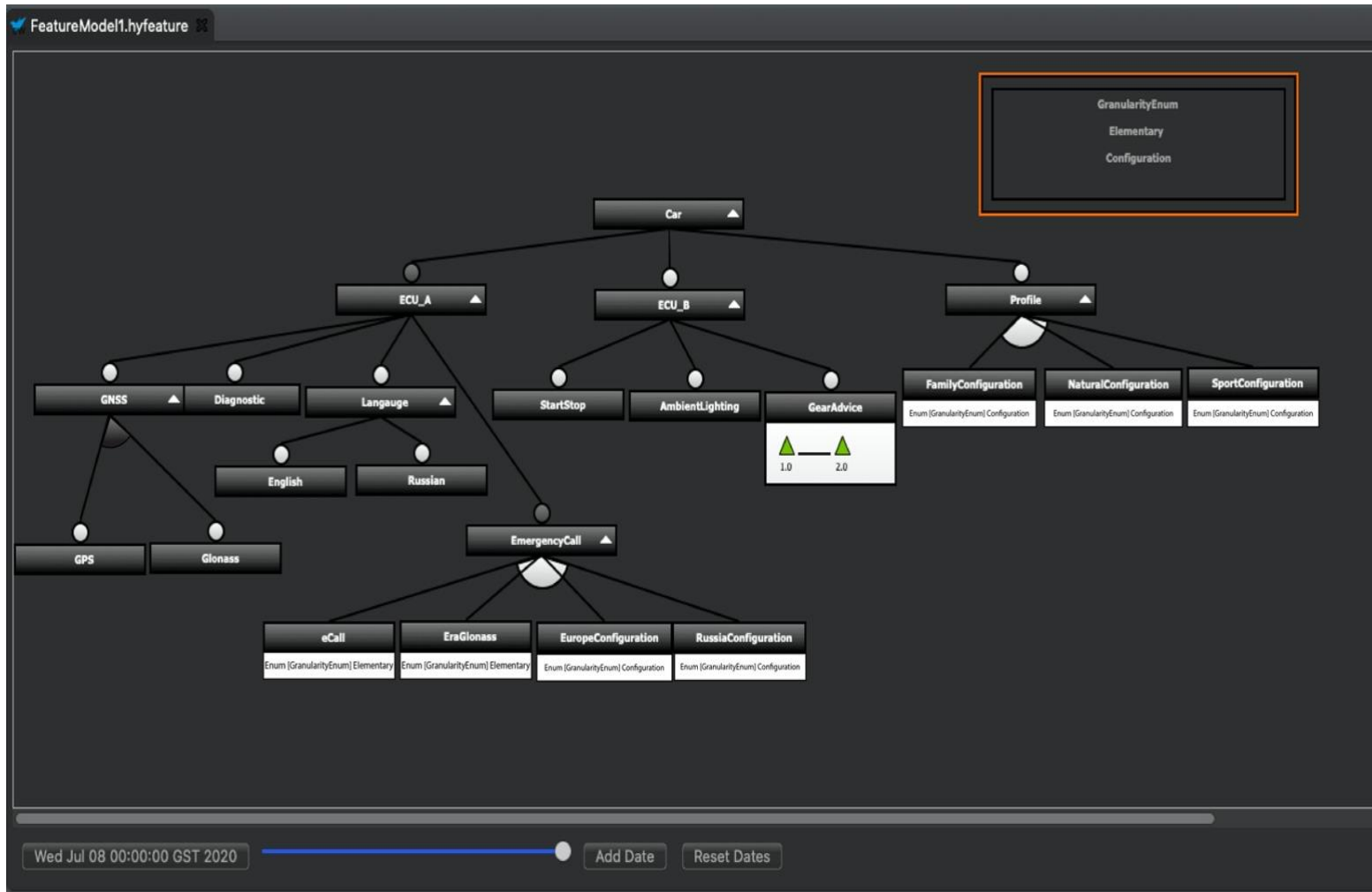


Figure 4.11: Updated FM with the configuration features, modeled using the FM editor of DarwinSPL.

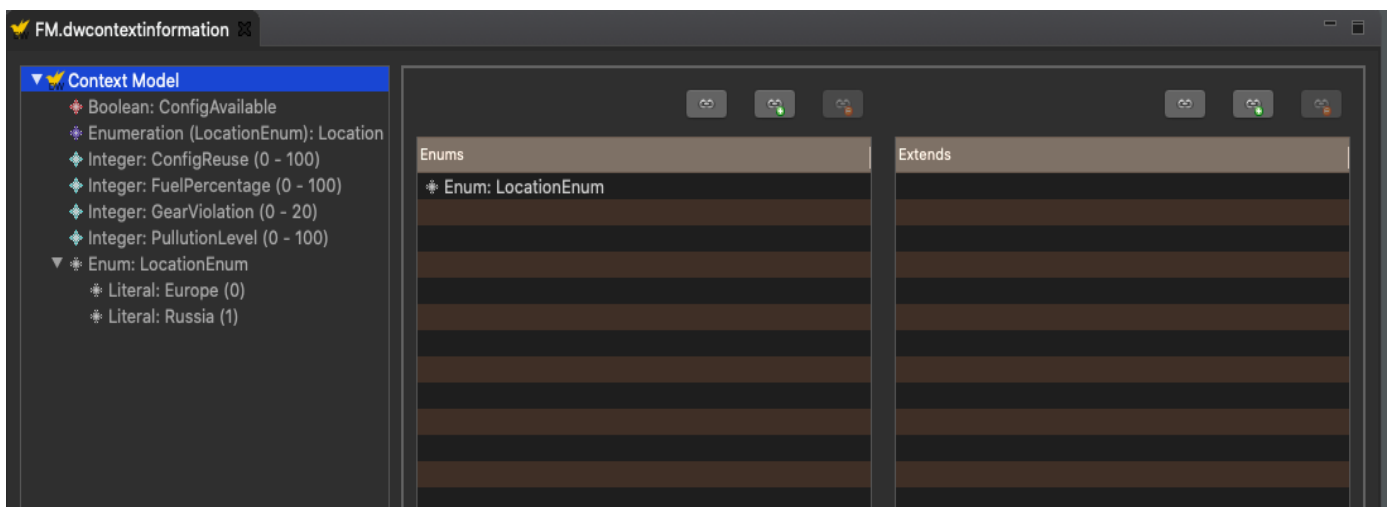


Figure 4.12: Contextual elements of the FM, modeled using the contextual information editor of DarwinSPL.

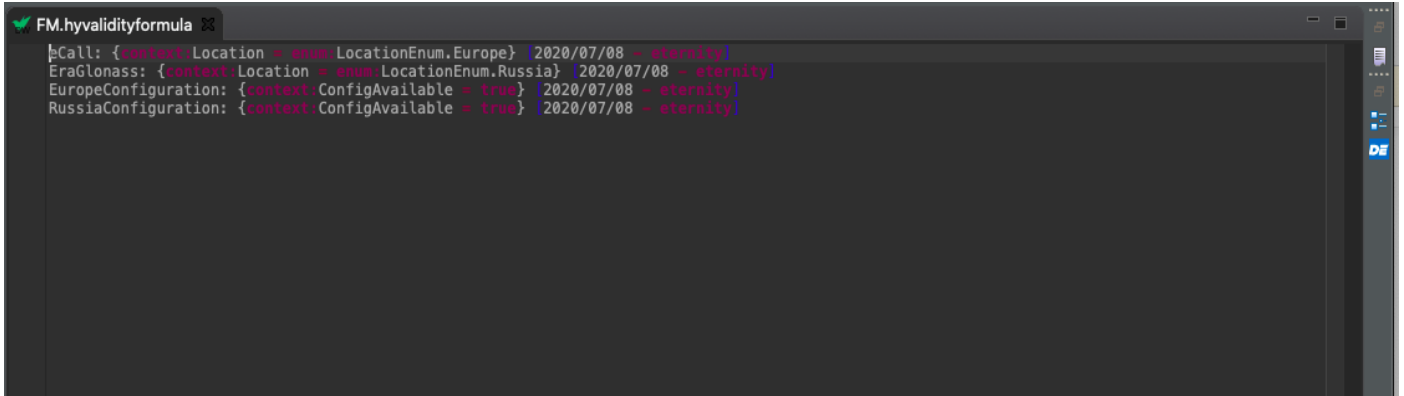


Figure 4.13: VF of the FM, modeled using the VF editor of DarwinSPL.

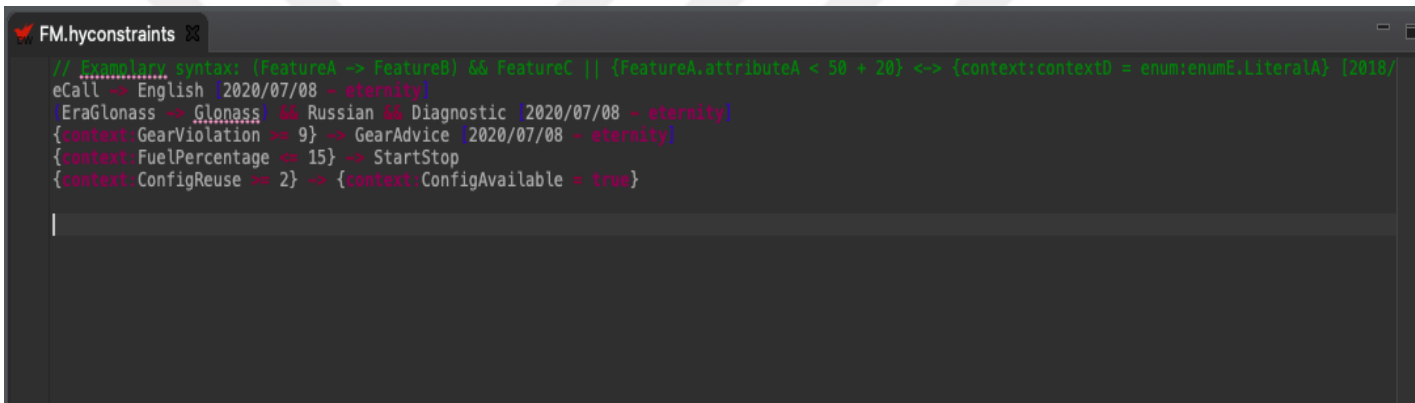


Figure 4.14: Constraints of the FM, modeled using the constraints editor of DarwinSPL.

4.6 SUMMARY

In this chapter, the challenges raised by modelling SASs developed with reuse process is discussed. Then illustrative example which has been used throughout the thesis is introduced. A solution to the raised challenges is provided by proposing an extension to the FM to model configuration feature and their reuse process. definitions of the proposed configuration features' extension along with its configuration features, relations, and constraints to ensure their coherency, that are able to address the challenges is presented with the help of an example. In addition, contextual elements are provided to support the modelling of the configuration feature and their reuse process.

The proposed extension including the frameworks, the configuration features management process, their processes and states is supported with the help of examples. At the end of the

chapter, the results and feasibility of the illustrative example is shown by using the DarwinSPL tool.

In the following Chapter, evaluation, application area, and conclusion and future work of the contribution will be discussed.



5. EVALUATION AND CONCLUSION

5.1 INTRODUCTION

In this chapter, the applications area of this work will be presented and discussed in section 5.1. in section 5.2, the evaluation of this work will be done depending on the introduced concepts in variability feature modelling domain, self-adaptive systems approaches, reuse process concepts, and on the new concepts introduced and defined by this work. After that, a comparison between this work and its obtained results and the closest and most significant works in the domain will be presented. Finally, in section 5.3, the chapter will be ended with a conclusion and future works.

5.2 APPLICATION AREA

Software engineering process will be stronger and much powerful by adding the methodology of feature model based configuration reuse scheme for self-adaptive systems to its techniques of variability feature modelling, since it has context properties, the runtime mechanisms, and the configuration feature concept with its reuse process. This methodology will empower the FM to make it able to model and reuse not only elementary features but also configuration features. In addition, it will reduce the gap between SASs and reuse process consequently reduce the complexity resulted from compounding of the runtime variability of self-adaptive systems and the reuse process.

This work is recommended to be applied to any area of variability feature modelling, such as product family engineering, dynamic software product lines, system families that require self- adaptive policies developed with reuse process, runtime reconfigurable and reuse mechanisms and self-management.

Real examples: Autonomous cars, smart homes, wearable devices, automated transportation systems and any system that require runtime reconfigurable process, reuse mechanisms and self-adaptive/ reconfiguration/ management policies.

5.3 EVALUATION

Feature models are used to represent and express all possible configuration (instances/products) of software systems in a hierarchical form by modelling and reusing systems' elementary features. Kang et al [11] presented the first feature diagrams approach back in 1990. This approach is recognized as one of the most significant contributions in software product line engineering's field [11]. Till now, FMs is one of the most using techniques to manage and reuse systems' commonalities and variabilities among their products, thus reducing systems' complexity. Further, FMs is one of the new forms of reuse that is able to bring new opportunities for the reuse process beyond the conventional software components [40]. the need to model such systems has brought about the need to make FMs more express and capable of representing self-adaptive systems, their context probertites and requirements, and their reuse process.

We believe that using the feature models to formally model self-adaptive systems developed with reuse process provides flexibility, reduce time and cost, and mitigate complexity of such compound systems.

As discussed in Chapter3, feature models need to be more expressive and have extensions in order to make them capable of modelling and representing self-adaptive systems developed with reuse process. Moreover, current FMs focus on representing and reusing elementary features without taking into consideration representing and reusing configurations features.

In this thesis, an extension to the feature model with new features, context information, and constraints to ensure coherency between features is proposed. This extension empower the FM to support the modelling of the configuration features and their reuse process.

Based on current researches in the domain of variability feature modeling techniques, SASs, and reuse processes, the following criteria are selected to evaluate the proposed scheme and compare its performance with those of existing methods.

- I. Development of SASs using reuse process support.
- II. Supporting the configuration feature concept using FMs.
- III. Efficient reuse of configuration features.
- IV. Managing product customization during run time.

The proposed scheme will be compared with the following methods:

- I. Specification and automated validation of staged reconfiguration processes for dynamic software product lines [39].
- II. Software Evolution: A Features Variability Modeling Approach. J. Software [21].
- III. Context-aware reconfiguration in evolving software product lines [4].
- IV. Feature model configuration based on two-layer modeling in Software Product Lines [34].
- V. ASPLe: A methodology to develop self-adaptive software systems with systematic reuse [6].

In a study [39], the FM was extended using binding time constraints to restrict SAS-staged reconfiguration processes. The concepts of configuration stages, used to define arrangements among features and differentiate between dynamic and static binding times of features; complex binding time constraints, used to restrict product configuration processes by taking logical consistency and arrangement restrictions of feature selections into consideration; and stateful reconfiguration models, used to control possible sequences of configuration changes of dynamic features during the product's life cycle were introduced in the indicated paper.

However, these configurations were not modeled using FMs and the self-adaptive system is not developed with reuse process. Whereas, in this thesis, the FM manages the configuration features as well as their reuse process.

In another study [21], a Feature-based Variability approach for modelling Software Evolution was presented to support evolution and variability concepts. It extends the traditional feature model concepts with evolution concepts, by building an extended feature model from the traditional feature model. A composed feature (configuration) was presented along with its relevant evolution relations, these evolution relations enable the feature model to have feature grouping and metamorphosing.

However, this approach has modelled the configuration using an FM, but did not model the reuse process. Further, the scope of the added extension of this approach is limited to predefined events (software evolution), planned at the requirements and design stages. While, in this thesis, as the added extension to the FM is integrated with the HyFM [4], it is capable of supporting random adaptation at run time, as well as managing the configuration features and their reuse process.

Whereas in [4], a HyFM was introduced, which supports feature attributes, contextual information, and relations between contextual information and features selected based on current context and constrained by VFs, cross-tree constraints, a finite set of feature versions, version-aware constraints, and cardinality. These additional concepts enable HyFM to model and manage SASs by focusing on the representation and reuse of elementary features without considering the modeling or reuse of configuration features.

In this thesis, as the proposed system is integrated with HyFM, it is capable of supporting SAS modeling and management, as well as modeling and reusing configuration features. Thus, the proposed method satisfies all of the aforementioned criteria.

In another study [34], the authors presented an FM as a two-layer model consisting of application layers and infrastructure ones to manage the required configuration that satisfy

the stakeholders' desires and the potential of the platform of available implementations. In addition, this FM is capable of defining two types of constraints—inner constraints among features belonging to the same layer and intra-constraints among features belonging to different layers.

However, this FM manages and reuses only elementary features, without considering the management and reuse of configuration features. Further, it does not support product customization during run time. Whereas, in this thesis, the FM manages and reuses elementary features as well as it manages and reuse the configuration features.

Whereas an autonomic software product line engineering (ASPL_e) methodology was presented in another study [6]. This methodology allows developers to develop SASs with reuse processes by selecting, specializing, and subsequently, integrating reusable artifacts.

This methodology used FMs to model only the scope definition of ASPL_e; hence, no concepts were further added to the FM to support the modeling of the developed SAS with reuse. Whereas, in this thesis, as the proposed system is integrated with the HyFM [4], it is capable of supporting SAS modeling and management, as well as modeling configuration features and their reuse process.

The conclusions of the aforementioned comparative analysis have been depicted in Table 5.1, where the criteria used have been numbered sequentially from 1 to 4.

Table 5.1: Comparison between various configuration management schemes modeled using FM

Criterion	[39]	[21]	[4]	[34]	[6]	[this thesis]
1	✗	✗	✗	✗	✓	✓
2	✗	✓	✗	✗	✗	✓
3	✗	✗	✗	✗	✗	✓
4	✗	✗	✓	✗	✗	✓

5.4 CONCLUSION

In one of the most significant tasks in the domain of FM-based SASs [4], the authors introduced a framework for SPLs that allow modeling of contextual influences, user customization, and evolution. In particular, a hybrid feature model (HyFM) was presented. It supports contextual information, validity formulae, cross-tree constraints, and version awareness. However, this approach is incapable of managing configuration features and their reuse process. Moreover, a reconfiguration process is initiated every time the context changes or an artifact of the system is evolved by an engineer. This escalates the required time and resources.

To rectify this shortcoming, an extension to HyFM [4] was proposed in this thesis to manage configuration features that support its modeling and coherent reuse, including new features, context information, and constraints to ensure coherency between features. This extension empower the FM to support the modelling of the configuration features and their reuse process, thus supporting SASs developed with reuse process. Configuration feature

management process has been defined to manage the configuration features and their reuse process from its creation to destruction.

The fundamental complexity of the management of configuration reuse processes lies in the adaptation of the reused configuration features. Adaptation before reuse becomes necessary, as the configuration to be reused might no longer be suitable for the new context. Further, once a configuration feature is created, it requires continuous maintenance in the repository, for instance, new configuration features need to be saved, configuration features with errors need to be tracked, and configuration features that are no longer being reused for any reason need to be removed.

The complexity of configuration reuse process management, the variability of configuration features, their optimal selection, and their evolution remain major challenges that may be addressed in future works.

5.5 SUMMARY

In this chapter, the applications area of this thesis is presented and discussed. Then, the evaluation of this thesis is done depending on the introduced concepts in variability feature modelling domain, self-adaptive systems approaches, reuse process concepts, and on the new concepts introduced and defined by this thesis. A comparison between this thesis and its obtained results and the closest and most significant works in the domain is presented. Finally, the chapter is ended with a conclusion and provided with an overview of the future works.

REFERENCES

- [1] M.L.L. Carvalho, M.L.G. Da Silva, G. S. da Silva Gomes, A.R Santos, I.d.C. Machado, M.L.J. Souza, and E.S. De Almeida, “On the implementation of dynamic software product lines: An exploratory study,” *The Journal of Systems and Software*, vol. 136, pp. 74–100, 2018.
- [2] C. Krupitzer, M. Pfannemüller, V. Voss, C. Becker, “Comparison of Approaches for developing Self-adaptive Systems,” In *Proceedings of ACM*, New York, NY, USA. 2018.
- [3] A. Berns, S. Ghosh, “Dissecting Self-* Properties,” *International Conference on Self-Adaptive and Self-Organizing Systems*, pp. 10–19, 2009.
- [4] J. Mauro, M. Niekeb, Ch. Seidl, and I. ChiehYu, “Context-aware reconfiguration in evolving software product lines,” *Science of Computer Programming*, Elsevier , vol. 163, pp. 139–159, 2018.
- [5] N. Padhy, R.P. Singh, and S.C. Satapathy, “Software reusability metrics estimation: algorithms, models and optimization techniques,” *Computers & Electrical Engineering*, vol. 69, pp. 653–668, 2018.
- [6] N. Abbas , J. Andersson , and D. Weyns, “ASPLe: A methodology to develop self-adaptive software systems with systematic reuse,” *The Journal of Systems and Software* vol. 167, 2020.
- [7] M. Marques, J. Simmonds, P. O. Rossel, and M. C. Bastarrica, “Software product line evolution: A systematic literature review,” *Inform Software Tech*, vol. 105, pp. 190–208, 2019.
- [8] A. Tahir, S.A. Licorish, and S.G. MacDonell, “Feature Evolution and Reuse-An Exploratory Study of Eclipse,” *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, IEEE, pp. 582–587.

- [9] E.N. Teixeira, F.A. Aleixo, F.D. Amâncio, J.E. Oliveira, U. Kulesza, and C. Werner, "Software process line as an approach to support software process reuse: A systematic literature review," *Inform Software Tech*, Aug 13, 2019.
- [10] M.B. Duran, and G. Mussbacher, "Reusability in goal modeling: A systematic literature review," *Inform Software Tech*, Mar 9, 2019.
- [11] K. Kyo, C. Sholom, H. James, N. William, and P. Spencer, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Software Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-90-TR-21, 1990.
- [12] M. Griss, J. Favaro, and M. d'Alessandro, "Integrating feature modeling with the RSEB," In: *Proceedings of the Fifth International Conference on Software Reuse (ICSR)*, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 76-85, 1998.
- [13] K. Czarnecki, S. Helsen, and U. Eisenecker, "Formalizing cardinality-based feature models and their specializations," *Software Process Improvement and Practice*, vol. 1, pp. 7-29, 2005.
- [14] M. Naeem, and R. Heckel, "Towards matching of service feature diagrams based on linear logic," *Proceedings of Workshops of SPLC*, Munich, Germany, 2011.
- [15] M. Naeem, "Matching of Service Feature Diagrams based on Linear Logic," PhD Dissertation. Department of Computer Sciences, University of Leicester, UK, 2012.
- [16] G.M Assad, M. Naeem, and H. AbdulWahab, "Towards cardinality-based service feature diagrams," *Computational Ecology and Software*, vol. 5, pp. 69-76, 2015.
- [17] O. Younes, and S. Ghoul, "Systems Versioning: A Features-Based Meta-modelling Approach," *Online Special Journal Issues, International Science Index*, vol. 8, 2014.

- [18] C. Andres, C. Camacho, and L. Llana, “A formal framework for software product lines,” *Information and Software Technology*, vol. 55, pp. 1925–1947, 2013.
- [19] C. Camacho, L. Llana, and A. Núñez, “Cost-related interface for software product lines,” *Journal of Logical and Algebraic Methods in Programming*, Elsevier, vol. 85, pp. 227-244, 2016.
- [20] S. Ibraheem, and S. Ghoul, “Feature-based Variability Modeling in Software Evolution,” LAP LAMBERT Academic Publishing, Germany, 2016.
- [21] S. Ibraheem, and S. Ghoul, “Software Evolution: A Features Variability Modeling Approach,” *Journal of Software Engineering*, vol. 11, pp. 12-21, 2017.
- [22] A. Safilian, T. Maibaum, and Z. Diskin, “A theoretical framework for cardinality-based feature models: The semantics and computational aspects,” *Journal of Logical and Algebraic Methods in Programming*, Elsevier, vol. 97, pp. 30–54, 2018.
- [23] X. Lian, L. Zhang, J. Jiang, and W. Goss, “An approach for optimized feature selection in large-scale software product lines,” *The Journal of Systems and Software*, Elsevier, vol. 137, pp. 636–651, 2018.
- [24] X. Lian, and L. Zhang, “An evolutionary methodology for optimized feature selection in software product lines,” In: *The 26th International Conference on Software Engineering and Knowledge Engineering*, Hyatt Regency, Vancouver, BC, Canada, pp. 63–66, 2014.
- [25] X. Lian, and L. Zhang, “Optimized feature selection towards functional and non-functional requirements in software product lines,” In: *Software Analysis, Evolution and Reengineering (SANER)*, IEEE 22nd International Conference, pp. 191–200, 2015.

- [26] M. Trabelsi, N. Meddouri, and M. Maddouri, "A New Feature Selection Method for Nominal Classifier based on Formal Concept Analysis," *Procedia Computer Science*, Elsevier, vol. 112, pp. 186–194, 2017.
- [27] J. Guoa, J. Whiteb, G. Wanga, J. Li, and Y. Wanga, "A genetic algorithm for optimized feature selection with resource constraints in software product lines," *The Journal of Systems and Software*, vol. 84, pp. 2208–2221, 2011.
- [28] Y. Xue, J. Zhonga, T.H. Tanc, Y. Liua, W. Caia, M. Chenc, and J. Sun, "IBED: Combining IBEA and DE for optimal feature selection in software product line engineering," *Applied Soft Computing*, Elsevier, vol. 49, pp. 1215–1231, 2016.
- [29] J. Miaoa, and L. Niu, "A Survey on Feature Selection. *Procedia Computer Science*," Elsevier, vol. 91, pp. 919 – 926, 2016.
- [30] S. Visalakshi, and V. Radha, "A Literature Review of Feature Selection Techniques and Applications," *IEEE International Conference on Computational Intelligence and Computing Research*, 2014.
- [31] T. Clark, F. Ulrich, I. Reinhartz-Berger, and A. Sturm, "A multi-level approach for supporting configurations: A new perspective on software product line engineering," pp.170–178, 2017.
- [32] J.A. Pereira, P. Matuszyk, S. Krieter, M. Spiliopoulou, and G. Saake, "Personalized recommender systems for product-line configuration processes," *Computer Languages, Systems & Structures*, vol. 54, pp. 451–471, 2018.
- [33] Y. Li, T. Yue, S. Ali, and L. Zhang, "Enabling automated requirements reuse and configuration," *Software & Systems Modeling*, vol. 3, pp. 2177–2211, 2019.

- [34] E.D. Farahani, and J. Habibi, "Feature model configuration based on two-layer modeling in Software Product Lines," *International Journal of Electrical & Computer Engineering*, vol. 9, pp. 2088–8708, 2019.
- [35] B.I. Moritani, J. Lee, "An approach for managing a distributed feature model to evolve self-adaptive dynamic software product lines," *21st International System and Software Product Line Conference (SPLC)*, pp.107–110 2017.
- [36] M. Bashari, E. Bagheri, and W. Du, "Self-adaptation of service compositions through product line reconfiguration," *Journal of Systems and Software*, vol. 144, pp. 84–105, 2018.
- [37] C.h. Werner, S. Werner, R. Schone, S. Gotz, and U. Aßmann, "Self-adaptive synchronous localization and mapping using runtime feature models," *The 7th International Conference on Data Science, Technology and Applications*, pp. 409–418, 2018.
- [38] A. Mousa, J. Bentahar, and O. Alam, "Context-aware composite SaaS using feature model," *Future Generation Computer Systems*, vol. 99, pp. 376–390, 2019.
- [39] M. Lochau, J. Bürdek, S. Hölzle, and A. Schürr, "Specification and automated validation of staged reconfiguration processes for dynamic software product lines," *Software & System Modeling*, vol. 1, pp. 125–152, 2017.
- [40] R. Capilla, B. Gallina, C. Cetina, J. Favaro, "Opportunities for software reuse in an uncertain world: From past to emerging trends," *Journal of software: evolution and process*, vol. 8, 2020.
- [41] D. Garlan, S.W. Cheng, A.C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: architecture-based self-adap- tation with reusable infrastructure," *Computer*, vol. 10, pp. 46–54, 2004.

- [42] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjørven, "Using architecture models for runtime adaptability," *IEEE Software*, vol. 2, pp. 62–70, 2006.
- [43] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, and U. Scholz, "Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments," In *Software engineering for self-adaptive systems*, Springer, Berlin, Heidelberg, pp. 164-182. 2009.
- [44] R. Calinescu, , D. Weyns, S. Gerasimou, M.U. Iftikhar, I. Habli, and T. Kelly, "Engineering trustworthy self- adaptive software with dynamic assurance cases," *IEEE Transactions on Software Engineering*, vol. 11, pp. 1039–1069, 2018.
- [45] C. Krupitzer, F.M. Roth, S. Vansyckel, and C. Becker, "Towards reusability in autonomic computing," In: *2015 IEEE International Conference on Autonomic Computing*, pp. 115–120.
- [46] S. Fischer, G. K. Michelon, R. Ramler, L. Linsbauer, and A. Egyed, "Automated test reuse for highly configurable software," *Empirical Software Engineering*, pp. 1-38, 2020.
- [47] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, "Automated reasoning on feature models," *Advanced Information Systems Engineering*, Berlin Heidelberg, Springer, pp. 491-503, 2005.
- [48] C. Seidl, I. Schaefer, and U. Aßmann, "Capturing variability in space and time with hyper feature models," *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems – VaMoS '14*, 2014.
- [49] M. Nieke, J. Mauro, C. Seidl, and I.C. Yu, "User profiles for context-aware reconfiguration in software product lines" *International Symposium on Leveraging Applications of Formal Methods*, Springer, Cham, pp. 563–578, 2016.

- [50] M. de Jonge and J. Visser, “Grammars as Feature Diagrams,” In Workshop of Generative Programming 2002 (GP2002), Austin, Texas.
- [51] M. Mannion, “Using First-Order Logic for Product Line Model Validation,” In Proceedings of The Second International Conference on Software Product Lines (SPLC’02), pp. 176–187, London, UK, 2002.
- [52] D. S. Batory, “Feature Models, Grammars, and Propositional Formulas,” In Proceedings of Ninth International Conference on Software Product Lines (SPLC’05), , Rennes, France, pp. 7–20, 2005.
- [53] D. K. Forbus and J. de Kleer, “Building Problem Solvers,” volume 1 of Artificial Intelligence, MIT Press, 1993.
- [54] K. Czarnecki and A. Wasowski, “Feature Diagrams and Logics: There and Back Again,” In Proceedings of the 11th International Software Product Line Conference (SPLC’07), Washington, DC, USA, pp. 23–34, 2007.
- [55] D. Benavides, S. Segura, and A. R. Cortés, “Automated Analysis of Feature Models 20 Years Later: A Literature Review,” Information Systems, vol. 6, pp. 615–636, 2010.
- [56] Y. Bontemps, P. Heymans, P.Y. Schobbens, and J.C. Trigaux, “Semantics of FODA Feature Diagrams,” In Proceedings SPLC (2004) Workshop on Software Variability Management for Product Derivation – Towards Tool Support, pp. 48–58.
- [57] J. Guo, Y. Wang, P. Trinidad, and D. Benavides, “Consistency maintenance for evolving feature models,” Expert Systems with Applications, vol. 5, pp. 4987-4998, 2012.
- [58] M. Bhushan, S. Goel, and K. Kaur, “Analyzing inconsistencies in software product lines using an ontological rule-based approach,” The Journal of Systems and Software, Elsevier, pp. 1-13. 2017.

- [59] B. Wang, Y.F. Xiong, Z.J. Hu, H.Y. Zhao, W. Zhang, and H. Mei, “Interactive Inconsistency Fixing in Feature Modeling,” *Journal of computer science and technology*, Springer Science, 2014.
- [60] M. Tanhaei, J. Habibi, and SH. Hosseinabadi, “A feature model based framework for refactoring software product line architecture,” *Journal of Computer Science and Technology*, vol. 5, pp. 951–986, 2016.
- [61] J. A. Pereira, P. Matuszyk, S. Krieter, M. Spiliopoulou, G. Saake, “Personalized recommender systems for product line configuration processes,” *Computer Languages, Systems & Structures*, vol. 54, pp. 451–471, 2018.
- [62] S. Ghoul, “A road map to bio-inspired software engineering,” *Research Journal of Information Technology*, vol. 8, pp. 75 – 81, 2016.
- [63] A.K. Kar, “Bio inspired computing –A review of algorithms and scope of applications,” *Expert Systems With Applications*, Elsevier, vol.59, pp. 20–32, 2016.
- [64] M. Bakhouya, and J. Gaberb, “Bio-inspired Approaches for Engineering Adaptive Systems,” *Procedia Computer Science*, Elsevier, vol. 32, pp. 862 – 869, 2014.
- [65] X.S. Yang, and Z. Cui, “Bio-inspired Computation: Success and Challenges of IJBIC,” *International Journal of Bio-Inspired Computation*, vol.1, pp. 1-6, 2014.
- [66] B.Y. Qua, Y.S. Zhua, Y.C. Jiaoa, M.Y. Wua, P.N. Suganthand, and J.J. Liang, “A survey on multi-objective evolutionary algorithms for the solution of the environmental/economic dispatch problems,” *Swarm and Evolutionary Computation*, vol.38, pp. 1–11, 2018.
- [67] A. Louati, S. Darmoul, S. Elkosantini, and L. Ben Said, “An artificial immune network to control interrupted flow at a signalized intersection,” *Information Sciences*, pp. 70–95, 2018.

- [68] A. Oveisi, J.M. Bagetti, and T. Nestorović, “Nonlinear observer-based recurrent wavelet neuro-controller in disturbance rejection control of flexible structures,” *Engineering Applications of Artificial Intelligence*, vol. 69, pp. 50–64, 2018.
- [69] J. Li, S. Fong, K. Raymond, b. Wong, W. Victor, and b. Chu, “Adaptive multi-objective swarm fusion for imbalanced data classification,” *Information Fusion*, vol. 39, pp. 1–24, 2018.
- [70] X.W. Yao, W.L. Wang, S.H. Yang, and Y.F. Cen, “Bio-inspired self-adaptive rate control for multi-priority data transmission over WLANs,” *Computer Communications*, Elsevier, vol. 53, pp. 73–83, 2014.
- [71] S. Ichiro, “Bio-inspired Self-adaptive Agents in Distributed Systems,” *Distributed Computing and Artificial Intelligence*, Springer Berlin Heidelberg, vol.151, pp. 221-228, 2012.
- [72] E. Naffar, and S. Ghoul, “A Genetic Framework Model for Self-adaptive Software,” *Journal of Software Engineering*, vol. 3, pp. 255-265, 2017.
- [73] S.M. Alfar, “Self-Adaptive Software to Unpredicted Relevant Events,” Msc thesis in Computer Science, Philadelphia University, Jordan, 2017.
- [74] S. Wenli, J. Dandan, Y. Yuhao, and Y. Ye, “Fault simulation of electro-hydraulic servo system for fault self-healing based on immune principle,” *Computational Intelligence and Design*, IEEE, vol. 2, pp. 136-139, 2017.
- [75] P.V. Florin, and A. Grigore, “Nature-inspired Approaches in Software Faults Identification and Debugging,” *Procedia Computer Science*, Elsevier, vol. 92, pp. 6-12, 2016.
- [76] R. Ligeiro, “Monitoring applications: An immune inspired algorithm for software-fault detection,” *Applied Soft Computing*, Elsevier, vol. 24, pp. 1095–1104, 2014.

- [77] V. Bischoff, K. Farias, L.J. Gonçalves, and J.L.V. Barbosa, “Integration of feature models: A systematic mapping study,” *Information and Software Technology*, Elsevier, pp. 1–17, 2018.
- [78] A. Elkhodary, S. Malek, and N. Esfahani, “On the role of features in analyzing the architecture of self-adaptive software systems,” *CEUR Workshop Proceedings*, vol. 509, pp. 41-50, 2009.
- [79] N. Esfahani, A. Elkhodary, and S. Malek, “A Learning-Based Framework for Engineering Feature-Oriented Self-Adaptive Software Systems,” *IEEE Transactions on Software Engineering*, vol. 39, 2013.
- [80] G.H. Alférez, V. Pelechanob, R. Mazoc, C. Salinesic, and D. Diazc, “Dynamic adaptation of service compositions with variability models,” *Journal of Systems and Software*, Elsevier, vol. 91, pp. 24–47, 2014.
- [81] R. Mizouni, M. Abu Matar, Z. Al Mahmoud, S. Alzahmi, and A. Salah, “A framework for context-aware self-adaptive mobile applications SPL,” *Expert Systems with Applications*, Elsevier, vol. 41, pp. 7549–7564, 2014.
- [82] G. Pascual, R. Lopez-Herrejonb, M. Pintoa, L. Fuentes, and A. Egyedb, “Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications,” *Journal of Systems and Software*, Elsevier, vol. 103, pp. 392–411, 2015.
- [83] H.C. Lee, and S.W. Lee, “Decision supporting approach under uncertainty for feature-oriented adaptive system,” *Proceedings - International Computer software and Applications Conference*, pp. 324-329, 2015.

- [84] M.L. Berkane, L. Seinturier, and M. Boufaïda, “Using variability modelling and design patterns for self-adaptive system engineering: application to smart-home,” *International Journal of Web Engineering and Technology*, vol. 10, pp. 65–93, 2015.
- [85] A.M. Sharifloo, A. Metzger, C. Quinton, L. Baresi, and K. Pohl, “Learning and evolution in dynamic software product lines,” In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2016 IEEE/ACM 11th International Symposium on IEEE, pp. 158-164.
- [86] F. Mohamed, R. Mizouni, M. Abu-Matar, M. Al-Qutayri, and J. Whittle, “An Integrated Platform for Dynamic Adaptation of Multi-Tenant Single Instance SaaS Applications,” *IEEE 5th International Conference on Future Internet of Things and Cloud*, pp. 257-264, 2017.
- [87] M. Nieke, C. Seidl, and S. Schuster, “Guaranteeing configuration validity in evolving software product lines,” in: *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '16*, ACM, New York, NY, USA, pp. 73–80. 2016.
- [88] J. Mauro, M. Nieke, C. Seidl, I.C. Yu, “Context aware reconfiguration in software product lines,” in: *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems – VaMoS '16*, 2016.
- [89] M. Nieke, G. Engel, and C. Seidl, “DarwinSPL: an integrated tool suite for modeling evolving context-aware software product lines,” in: *Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems, VAMOS '17*, ACM, New York, NY, USA, pp. 92–99, 2017.
- [90] C.W. Krueger, “Software reuse,” *ACM Computing Survey (CSUR)*, vol. 2, pp. 131–183, 1992.

- [91] W.B. Frakes, and K. Kang, “Software reuse research: status and future,” IEEE Transaction on Software Engineering, vol.7, pp. 529–536, 2005.
- [92] K. Pohl, G. Böckle, and F. Van Der Linden, “Software Product Line Engineering: Foundations, Principles, and Techniques,” Springer-Verlag New York, Inc., 2005.
- [93] A. Shatnawi, A. Seriai, H. Sahraoui, T. Ziadi, and A. Seriai, “ReSId: Reusable service identification from software families,” The Journal of Systems & Software, Elsevier, vol. 170, 2020.
- [94] H. McManus, and D. Hastings, “A framework for understanding uncertainty and its mitigation and exploitation in complex systems,” INCOSE International Symposium, vol. 1, pp. 484–503, 2005.
- [95] L. De Moura, and N. Bjørner, “Satisfiability modulo theories: introduction and applications,” Communication of the ACM, vol. 9, pp. 69–77, 2011.

APPENDIX A

A. THE CONTEXTUAL RECONFIGURATION ENGINE HyVarRec

In the following, the contextual reconfiguration engine [4] is described and explained. Firstly, the HyVarRec is defined. Secondly, the HyVarRec engine's general execution flow is described. Thirdly, the encoding of the FM's entities and constraints are detailed, besides, how the entities of the FM are translated into an optimization problem is explained.

A.1 Reconfiguration Engine HyVarRec

the HyVarRec is a context-aware reconfiguration engine that take into account the running configuration, the manufacturer and user profiles, and the current context to create a proper configuration that maximizes the user preferences.

HyVarRec engine can be invoked each time the context changes or the SPL has evolved by an engineer. HyVarRec engine is able to successfully create new valid configurations that are suitable for the current context and maximize the specified user profiles. In addition, it is able to handle SPLs/DSPLs that are up to thousand features in a time that is less than a minute [4].

Figure A.1 shows the results of generating 100 feature models each one with 100 features. On average HyVarRec engine was able to generate a valid context-aware configuration in 0.686 seconds and 0.66 seconds to show that no valid context-aware configuration can be produced, as shown in figure A.1.a and figure A.1.b.

While Figure A.2 shows the results of generating 100 feature models each one with 500 features. On average HyVarRec engine was able to generate a valid context-aware configuration in 2.93 seconds and 2.65 seconds to show that no valid context-aware configuration can be produced, as shown in figure A.2.a and figure A.2.b.

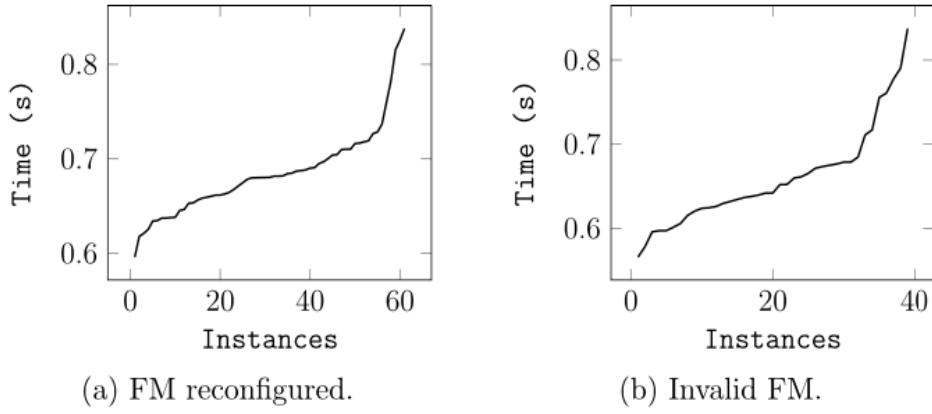


Figure A.1: the bench of 100 results [4].

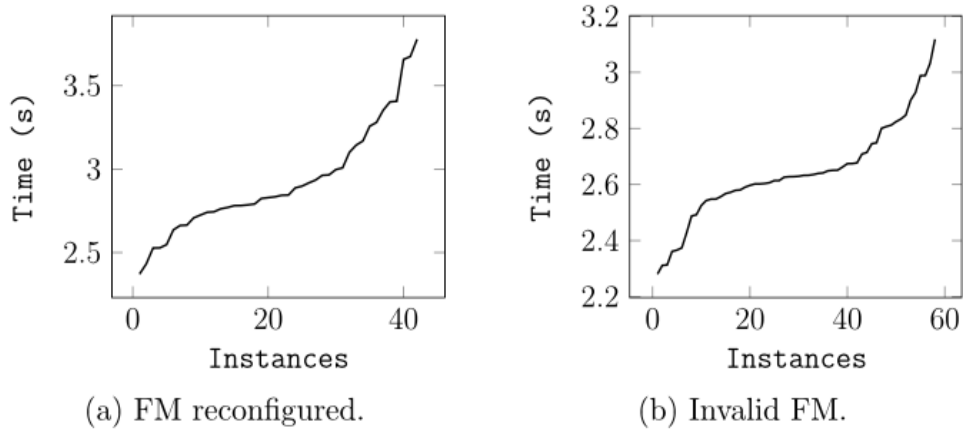


Figure A.2: the bench of 500 results [4].

A.1.1 The HyVarRec engine's general execution flow

the reconfiguration process will be triggered each time a variant of a D/SPL needs to be changed (adapted). The necessity for system reconfiguration might vary according to the domain. For example, this process will be started on every context change or each time an engineer evolved the D/SPL artifacts and desire all the systems to be adapted to the new

version of D/SPL. So , the reconfiguration process of the HyVarRec engine starts on every context change and each time an engineer evolves the D/SPL artifact. But, after adding our extension, configurations for the same context will be saved for reuse purposes instead of triggering the reconfiguration engine HyVarRec. Thus, the time and effort of reconfiguring the same configuration for the same context will be reduced.

The HyVarRec engine, As depicted in Figure A.3, requires input of different sources, that are: the FM, the present configuration (C_0) of the distant device, the present values for the contextual information (Ctx), the manufacturer profile (M), and the user profile (P). The main function of the HyVarRec engine is to supply valid configurations (C_{new}) for the context (Ctx). These valid configurations firstly maximize the preferences of the user profile (P), then the manufacturer profile (M).

In case of two equal quality configurations with regard to the user and manufacturer preferences maximization, the configuration that decrease the difference between the initial configuration (C_0) will be supplied. This shows that the HyVarRec engine first tries to reduce the number of rejected features that are needed to transform configuration C_0 into configuration C_{new} and, then, to maximize the number of features and attributes whose values can be preserved the same.

Finally, HyVarRec engine will produce the configuration C_{opt} , as the optimal output configuration for the given context, trying to satisfy as many preferences of the user profile P and the manufacturer profile M as possible.

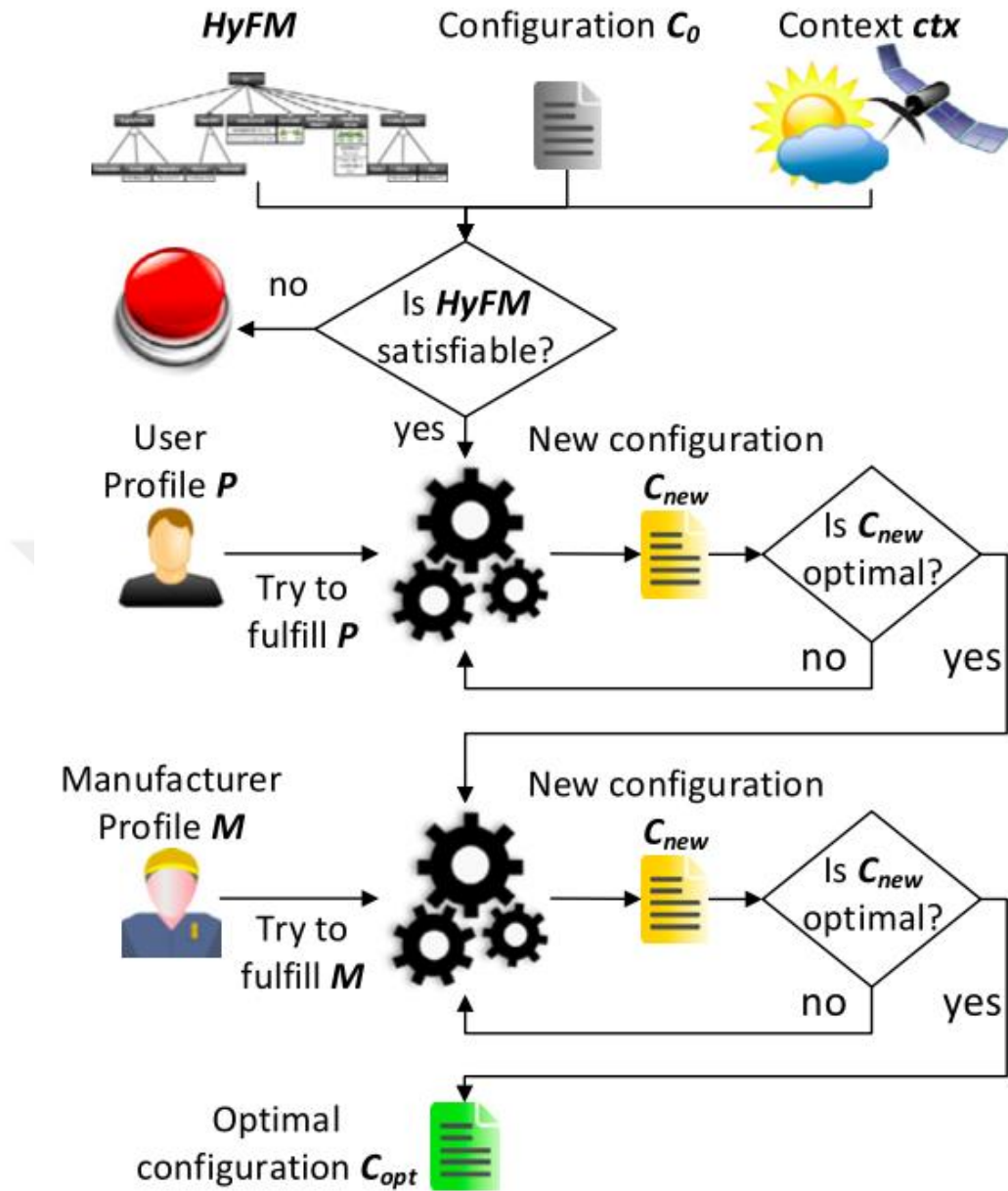


Figure A.3: the execution flow of the HyVarRec engine [4]

A.1.2 Encoding of the feature model's constraints

The reconfiguration engine HyVarRec depends on the Satisfiability Modulo Theory (SMT) [95]. In particular, the HyVarRec engine tries to find a solution for optimization problems

where the FM's constraints are defined by using logical formula and are used to decrease the space of allowable solutions. In addition, the goal here is to not find any solution instead, a solution that either minimizes or maximizes a certain objective function. According to the methodology presented in [47], authors transformed the “classical” feature model's part of the HyFM [4] into the propositional formulas on features.

To allow the HyVarRec engine to support different feature models modeling engines, the FM with its entities and constraints are required to be given as a JSON object.¹ conventionally, the feature with the id f will be represented as `feature[f]`, while the attribute with the id a will be represented as `attribute[a]` moreover, the context with the id c will be represented as `context[c]`.

Users are required to detail the attributes and the contexts domain, since they can take different values. Particularly, to make the notation simple, the feature attributes and context are required to have a limited domain that is represented by an adjacent set of integers. For example, the domain of attributes or contexts of Boolean type is: $[0,1]$, the domain of attributes or contexts of Integer type have their own specified domain. For the domain of attributes or contexts of enumeration type with n literals is: $[0,(n-1)]$.

Attributes of features are defined by using a list where every feature attribute is presented by determining its id using the following: the keyword `id`, the limits of feature attribute domain (using `min` and `max` keywords) and the feature that is associated with it (using `featureId` keyword). For example, we can defined the Boolean feature attribute a that is associated with the feature f as follows:

¹ The JSON input schema formally describing the input for HyVarRec can be retrieved from https://github.com/HyVar/hyvar-rec/blob/master/spec/hyvar_input_schema.json.

```

"attributes": [

    { "id": "attribute[a]",

      "min": 0,

      "max": 1,

      "featureId": "feature[f]"

    }
]

```

Context is also defined in a similar way to the features attributes. For example, we can define the context c which will take a value between 3 and 10 as shown below.

```

"contexts": [

    { "id": "context[c]",

      "min": 3,

      "max": 10

    }

]

```

The HyFM will be given in a textual representation with its entities and constraints and it will be represented by using list of propositional constraints. The EBNF grammar that define the propositional constraint has been defined using the ANTLR conventions (see Table 2.1, chapter 2).

The way of defining the HyFM is by making a list of all the constraints that represent its structure, validity formulas, and cross tree constraints. For example, if we assumed that the HyFM have only the root/concept feature f . This feature will be represented just by the constraint that require the root feature f to be selected. The representation will be as follows.

"constraints":[

"feature[f] = 1"]

Unlike the features attributes and context elements, features do not necessitate to be directly defined as their domain will automatically be set to $\{0,1\}$ where the value 1 means the feature is selected. Hence, features can be automatically presented by mentioning them the constraints. The FM 's hierarchal structure is translated into constraints and is shown early in Table 2.2, chapter 2.

Features that have more than one version will be decomposed as a parent feature that have one child feature for each version. the parent feature is forced to be selected by an added constraint if and only if one of its children features (version) is selected. For example, the feature *ConnectionGateway* of the illustration example in Figure 4.1, chapter 4, have two versions that are version [1.0] and [2.0]. Hence, the feature *ConnectionGateway* will be encoded as feature[1], its version [1.0] as feature[2] and its version [2.0] as feature[3] as follows.

$\text{feature}[1] = 1 \leftrightarrow (\text{feature}[2] + \text{feature}[3]) = 1$

User profiles and manufacturer profiles are encoded by using a list of preferences. Both preferences are sharing the same syntax and it is captured by using the EBNF grammar as follows.

preference: constraint |

`('min' | 'max') '(' 'attribute[' id ']' ')'`

A preference of user or manufacturer profile can be an arithmetic expression, a Boolean expression, or a requirement in order to maximize or minimize a given attribute's value. For the Boolean expression the preference indicates is to satisfy it and the arithmetic expression indicates that the preference to maximize their value.

There is one difference between the user preferences and the manufacturer one's which is associated with their priority: priority of the user preferences higher from manufacturer one's. Priority is expressed by listing the preferences in advanced. For example, if the feature GearAdvice is translated as feature[1], the feature ConnectionGateway as feature[2], and version [2.0] of the feature ConnectionGateway as feature[3], then number 2 and number 3 of the configuration preferences of the sporty driver (section 2.3) will be translated as follows.

"preferences":[

"feature[1] = 1", "feature[2] = 1 impl feature[3] = 1"]

For the cross tree constraints (CTCs), the encoding of the user profiles and the manufacturer one's into preference expression will be straightforward. The last thing needed as the input of the HyVarRec engine is the context and the current configuration. This must be presented by using the *configuration* keyword, listing every selected feature by using *selectedFeatures* keyword, the values of the attributes by using *attribute_values* keyword, and the contexts by using *context_values* keyword. For example, assuming we have a FM that having a selected feature *f* with its attribute *a* that set to the value (0) and a context *c* that set to the value (1). This configuration will be defined as follows.

"configuration": {

"selectedFeatures": ["feature[f]"],

```
"attribute_values": [  
  
  { "id": "attribute[a]",  
  
    "value": 0 } ],  
  
"context_values": [  
  
  { "id": "context[c]",  
  
    "value": 1 } ]  
}
```

Finally, by given all the aforementioned information, the HyVarRec engine can be run.

APPENDIX B

B. DarwinSPL TOOL KIT

B.1 DarwinSPL Tool

DarwinSPL is an open source software project that is able to model, use, and analyze evolving SPLs and context-aware SPLs. This tool can be downloaded from <https://gitlab.com/DarwinSPL/DarwinSPL> and its installation and setup instructions along with usage tutorials videos for giving an overview about the tool are also available.

