Sudan University of Science and Technology

College of Graduate Studies

A Safety-Based Architectural Design Method for Software Product Lines

طريقة تصميم معماري مبني على السلامة لخطوط إنتاج البرمجيات

Thesis submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

in

Computer Science

by

Mozamil Ebnauf Elgodbe Suliman

Sudan University of Science and Technology, Sudan

Supervised by

Prof. Hany H. Ammar

West Virginia University Morgantown, WV, USA

Co-Supervised by

Prof. Aisha Hassan

I I U M, Kuala Lumpur, Malaysia

December 2022

بسم الله الرحمن الرحيم

قال الله عز وجل في كتابه العظيم:

(اقْرَأْ بِاسْمِ رَبِّكَ الَّذِي خَلَقَ (1) خَلَقَ الْإِنْسَانَ مِنْ عَلَقٍ (2) اقْرَأْ وَرَبُّكَ الْأَكْرَمُ (3)
الَّذِي عَلَّمَ بِالْقَلَمِ (4) عَلَّمَ الْإِنْسَانَ مَا لَمْ يَعْلَمْ (5))

الآيات من 1 إلى 5 من سورة العلق.

Declaration

I declare that the research described in this thesis is original work and has been done by myself under the general supervision of my supervisor which I undertook at the Sudan University of Science and Technology.

This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References. I have followed the guidelines provided by the Institute in writing the thesis.

I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the University.

Some of the material presented in this thesis has been published in the following papers:

- Mozamil Ebnauf, W. Abdelmoez , Aisha Hassan, Hany H. Ammar, M. Abdelhamid, “State-driven Architecture Design for Safety–critical Software Product Lines,” in 2019 7th IEEE International Conference on Mechatronics Engineering (ICOM), October 2019, Kuala Lampur, Malaysia.
- Mozamil Ebnauf, Hany H. Ammar , ”Safety-driven Software Product Line Architectures Design (SSPLA) Methodology for Embedded Systems”, the Red Sea University Journal of Basic and Applied Science, Vol.2 Special Issue (1), 18 June 2017.

- مزمل ابنعوف, هاني عمار, ”منهجية التصميم المقاد بالسلامة لمعماريات خط الإنتاج”، المجلة العلمية لاتصالات الجمعية العربية للحاسبات، الجزء الخاص للمؤتمر الدولي لعلوم و هندسة الحاسوب (ICCA 2016) العدد الأول من المجلد التاسع 2016.
- Mozamil Ebnauf, Hany H. Ammar , ”Safety-driven software product line architecture design, A survey paper”, International Journal of Computer Applications Technology and Research (IJCATR), Volume 5, Issue 10, October 2016.

Signature _____ Date:

Declaration of the thesis' supervisor:

I, the signing here-under, declare that I'm the supervisor of the sole author of the thesis for the Doctor of Philosophy entitled:

A Safety-Based Architectural Design Method for Software Product Lines

Supervisor's name: Prof. Hany Ammar

Supervisor's signature: _____

Date: _____

Assigning the copy-right to CGS

I, the signing here under, declare that I'm the sole author of the thesis for the Doctor of Philosophy entitled "A Safety-Based Architectural Design Method for Software Product Lines", which is an original intellectual work. Willingly, I assign the copyright of this work to the College of Graduate Studies (CGS), Sudan University of Science and Technology (SUST). Accordingly, SUST has all the rights to publish this work for scientific purposes.

Candidate's name :Mozamil Ebnauf Elgodbe Suliman

Candidate's signature: _____

Date: :_____

DEDICATION

I dedicate this work to

my parents,

wife,

children,

and siblings

for their love, encouragement and support.

ACKNOWLEDEMENT

The Ph.D. study is a great learning and self-discovery experience. I owe my gratitude to a great many people who helped me through this journey.

It has been a great pleasure working with the faculty, staff, and students at the Sudan University of Science and Technology, during my tenure as a doctoral student.

First and foremost, I would like to express my heartfelt gratitude and sincere thanks to my principal supervisor, **Professor Dr. Hany H. Ammar** for his kind guidance, helpful advices, valuable information and constant support throughout the course of my research. His insight, determination, dedication, and desire to explore new ideas has been an inspiration. His approach to graduate students helps to create more than just a thesis, but to create a researcher.

I would also like to thank my external supervisor **Prof. Aisha Hassan** for her considerable guidance and supervision during the period of my research I spent in Malaysia. She provided me the chance to study at International Islamic University Malaysia for an year under Visiting Student Programme.

My special thanks also go to many other researchers I have worked with: **Dr. Walid Abdelmoez**, Arab Academy for Science Technology and **Dr. M. Abdelhamid**, Maritime Transport Egypt P.O.BOX1029, for their time and effort. Their thoughtful and insightful opinions provided many ideas for me to advance my research work.

I am grateful to my colleagues at the Faculty of Computer Science and Information Technology, Sudan University of Science and Technology for their feedback.

I thank also the Ministry of the High Education-Sudan and Omdurman Islamic University for funding the research reported in this thesis.

Additionally, I want to thank the International Islamic University Malaysia (Garden of Knowledge and Virtue) for providing me the chance to study in Kulliyyah of Engineering which helped me in successfully completing my thesis work. I also thank Qatar Research Fund (a member of Qatar Foundation) for its support which helped us to complete a part of my research work successfully which has been published in a scientific paper.

Finally, I deeply thank my parents, my wonderful wife and my family for their love, encouragement and support during the entire period of my study.

ABSTRACT

The safety is considered one of the most critical issues in the design of the modern systems (e.g. cyber-physical systems). With the increasing attention of software safety, how to improve software safety has already become a more important concerned issue, especially for the safety-critical systems. The Software Product-Line (SPL) and reusable software components are suitable approaches for these systems, which are often re-engineered from existing systems. Currently, the influence of the architecture in assurance of software safety is being increasingly recognized. However, the safety-based architectural design methods are limited in SPLs because of the complexity and variabilities existing in SPL architectures. For that, this work seeks to find an efficient and effective method that can be used into the design process of the safety-critical SPLAs which enhances and manages the safety of SPLs. The work proposed a method for safety-driven software product line architecture design (SSPLA). For efficiency, a number of efforts have been made. In this context the proposed design method mentioned above is configured and adapted to be state-based architecture design method. Also as a pattern based development of the reference architecture can support the development and application process of the product lines a new safety design pattern of statechart is developed. The result is an object-oriented design pattern which handles the safety attribute. Additionally, as there is a tight interplay between safety and security, and in order to address the influence of the security issues in the safety design using patterns, a pattern development approach is proposed which is then used to enhance the proposed safety design pattern of statechart. In order to show the applicability of our work as well as evaluate it, a simplified safety assessment model is developed as well as using of two case studies. The evaluation results show that there is a considerable improvement in the safety design of the SPLA after applying our work. The results have proved that the state-based approach highly supports the development of the safety critical systems and it is effective to handle the safety and security together in the design of the safety pattern which provides more benefits as it is a high level reuse. Finally, this research will benefit both architects and safety engineers who can design SPLAs or develop software products.

المستخلص

تعتبر السلامة (Safety) واحدة من أهم القضايا في تصميم الأنظمة الحديثة (مثل الأنظمة الفيزيائية السيبرانية (CPS) وأنظمة إنترنت الأشياء (IoT)). مع تزايد الاهتمام بالسلامة (Safety) في البرمجيات، أصبحت كيفية تحسين أمان البرامج بالفعل قضية ذات أهمية كبرى ، خاصة بالنسبة للأنظمة الحرجة للسلامة (Safety-critical Systems). تعد خطوط إنتاج البرمجيات (SPL) ومكونات البرامج القابلة لإعادة الاستخدام منهجيات مناسبة جداً لهذه الأنظمة، والتي غالباً ما تتم إعادة هندستها أوتصميمها من أنظمة موجودة. حالياً ، قد تم بشكل متزيد إدراك تأثير المعمارية في ضمان سلامة البرمجيات. ومع ذلك، فإن طرق التصميم المعماري المبني على السلامة محدودة في خطوط إنتاج البرمجيات وذلك بسبب التعقيد والتباينات الموجودة في معماريات خطوط إنتاج البرمجيات. لذلك، يسعى هذا البحث إلى إيجاد طريقة فعالة وذات كفاءة بحيث يمكن استخدامها في عملية تصميم معماريات خطوط إنتاج البرمجيات الحرجة للسلامة، والتي تعزز وتدير سلامة وأمان خطوط إنتاج البرمجيات. عليه فقد قدم البحث **طريقة تصميم معماري قائمة على السلامة لخطوط إنتاج البرمجيات** (SSPLA). ولتعزيز فعالية وكفاءة طريقة التصميم تم بذل عدد من المجهودات. في هذا السياق، فإن الطريقة المقترحة للتصميم المعماري لبرمجيات خطوط الإنتاج المذكورة أعلاه قد تم تحسينها لتصبح **طريقة تصميم معماري مبني على حالة النظام** (State-based Design). كذلك وبما أن التطوير القائم على النمط (Pattern-based Development) للمعمارية المرجعية يمكن أن يدعم عملية التطوير والتطبيق لخطوط الإنتاج، فإنه قد تم تطوير **نمط جديد لتصميم السلامة قائم على مخطط الحالة** (Safety Pattern of Statechart). وقد كانت النتيجة هي نمط تصميم كائن المنحى (OO) والذي يعالج أو يدير خاصية السلامة. إضافة إلى ذلك ولكون أن هناك إرتباط قوي بين السلامة والسرية و لمعالجة تأثير قضايا السرية في عملية التصميم المعماري باستخدام الأنماط ، تم اقتراح **منهجية تطوير أنماط** والتي تربط بين أنماط السلامة وأنماط السرية. ومن ثم تم إستخدام هذه المنهجيةً لتحسين نمط تصميم السلامة المقترح والمذكور أعلاه وذلك لمعالجة السرية في النمط. تعتبر هذه النسخة المطورة من النمط **كنمط جديد للسلامة والأمن (Safety & Security Pattern)**. أخيراً، ولتقييم عملنا و إظهار قابلية تطبيقه تم تطوير نمط مبسط لتقييم السلامة بالإضافة إلى استخدام حالتين دراسيتين . وقد أظهرت نتائج التقييم أن هناك تحسناً كبيراً في تصميم سلامة النظام بعد تطبيق عملنا. ووفقًا لهذه النتائج يمكننا إثبات أن **المنهجية القائمة على حالة النظام (State-based Method)** تدعم بشكل كبير تطوير الأنظمة الحرجة للسلامة والأمن. وقد أثبتت النتائج أيضًا أنه من الكفاءة التعامل مع السلامة والأمن معًا في تصميم نمط السلامة والذي قد يوفر المزيد من الفعالية لكونه إعادة استخدام على المستوى العالي. أخيراً، سيفيد هذا البحث كلاً من المهندسين المعماريين ومهندسي السلامة في مجال البرمجيات والذين يقومون بتصميم معماريات خط إنتاج البرمجيات أو حتى بتطوير منتج برمجي في مجال الأنظمة المدمجة والأنظمة الفيزيائية السيبرانية.

TABLE OF CONTENTS

DEDICATION ……..…… vi

ACKNOWLEDEMENT……..……..…….… vii

ABSTRACT ……..……...……...……viii

ABSTRACT IN ARABIC …….……...……. ix

TABLE OF CONTENTS …….…….. x

LIST OF TABLES.....xvi

LIST OF FIGURES xvii

LIST OF ABBREVIATIONS xx

CHAPTER I: INTRODUCTION 1

1.1 Introduction ……..…….……1

1.2 Problem Statement and Its Significance …….……...…3

1.3 Research Objectives …….……..……4

1.4 Research Questions ……..……..……5

1.5 Scope of Research …….……..…...……5

1.6 Thesis Structure ……..……..…….……6

1.7 Bibliographic Notes …….……..……7

CHAPTER II: LITERATURE REVIEW 8

2.1 Introduction ……8

2.2 Background ……..…..…9

2.2.1 Safety and Software Safety ……..……...……...……9

2.2.2 Cyber-physical Systems ……...…….……14

2.2.3 Software Architectures Design ……..……...……15

2.2.4 Software Product Line …….……16

2.2.5 The Complexity in Software Product Lines Development……16

2.2.6 Software Product Lines Architectures Design ……..……18

2.2.7 Safety–based SPLAs Design ……....…20

2.2.8 Scenario-based Architecture Design for SPL ……..……23
2.2.9 Safety-related Test Cases ……..…….…….....…25
2.3 Literature Survey ……..…….. 26
2.3.1 Research Method ……...……...……26
2.3.1.1 Research Questions ……..……..……27
2.3.1.2 Research Tasks …….……..……27
2.3.1.3 Literature Search Process …….…….…… 28
2.3.1.4Threats to Validity …….…… 29
2.3.2 Taxonomy and Classification ……..…… 29
2.3.2.1 First classification scheme ……...…….…… 29
2.3.2.2 The second classification scheme in term of SPLA design ……...…….…… 31
2.3.3 Mapping …….…….. 44
2.3.4 Survey Results and Discussion ……..…….....…45
2.4 Architectural design patterns …….……..45
2.4.1 Safety Design Pattern and Statechart Pattern …….…..…45
2.4.2 Safety Pattern with Security Control ……46
2.5 Chapter Summary and Open Research Issues …….47
CHAPTER III: RESEARCH METHODOLOGY 49
3.1 Introduction …….…… 49
3.2 Research Methodology ……...…… 49
3.3 Languages, Notations and Used Tools …….52
3.3.1 Language and Notations ……..…… 52
3.3.1.1 UML Language …….…..52
3.3.1.2 The Statechart Semantic ……..……..……53
3.4 Evaluation Notations, Metrics and Tools …….……55
3.4.1 The Relative Safety Improvement- a safety assessment metric 55
3.4.2 Markov Chain methods ……..…….…55

3.5 Software Tools ……56
3.5.1 UML Modeling Tools ……..…..…..56
3.5.2 Matlab ……....……..……58
3.5.3 MS Excel ……..…….60
3.5.4 Academic and Scientific Writing Tools ……61
3.6 Brief Description of the Case Studies……..…….62
CHAPTER IV: ARHITECTURE DESIGN FOR SAFETY-CRITICAL SPLs 63
4.1 Introduction ……..…63
4.2 Overview of Software Architecture Design …….……64
4.2.1 Safety-based Architecture Design ……...…. 64
4.2.2 A Comparison between Some of the More Related Works …..65
4.2.3 Safety-based Software Product line Architecture Design Methods …….....67
4.2.4 Safety Patterns for product lines ……..…….... 68
4.3 The Proposed SSPLA Design Method …… 69
4.3.1 The method Process ……..…….. 69
4.3.2 The Process Steps ……..……..70
4.4 State-driven Architecture Design for Safety-critical SPLs ….…….……77
4.5 Addressing the variability of the SPLs in the statechart ……..78
4.6 Illustrative Example …….……79
4.7 Chapter Summary ……88
CHAPTER V: SAFETY PATTERN OF STATECHART FOR SPLAs 90
5.1 Introduction ……..90
5.2. Overview of the Architectural and Design Patterns ……..……92
5.2.1 Pattern composition ……. 95
5.2.1.1 Safety Pattern and Statechart Patterns ……..…… 95
5.2.1.2 Safety Pattern with a Security Control ……..…… 96
5.2.2 Pattern-based SPLAs Development ……..…….. 97

5.3 Safety Design Pattern of Statechart - a proposed pattern ……..…..93

5.3.1 Pattern Description ……99

5.3.2 The Statechart Pattern Extension to Capture the Variability in the SPLs ……..……...…103

5.4 Chapter Summary ……..……..……103

CHAPTER VI: ENHANCING THE SAFETY DESIGN TO ADDRESS THE SECURITY ISSUES USING PATTERNs 104

6.1 Introduction ……104

6.2. Related Works ……..……106

6.3 The Proposed Pattern Development Approach for Safety and Security 108

6.3.1 A Proposed Pattern Development Engineering Lifecycle …..109

6.3.2 The Process Steps ……..……..…109

6.4 Developing a Safety and Security Pattern ……..…112

6.4.1 The Safety Design Pattern of Statechart ……...…112

6.4.2 Applying the proposed pattern Development Approach ……112

6.4.3 The Development Process ……..……113

6.4.4 The Enhanced Version of our Safety Pattern of Statechart …115

6.5 Chapter Summary ……...…...…119

CHAPTER VII: SAFETY ASSESSMENT FOR SSPLAs 120

7.1 Introduction ……120

7.2 The proposed Safety Assessment Model for SPLAs ……..……..…123

7.2.1 The Model Steps ……..……124

7.2.2 The Mathematical Calculations of the assessment process….124

7.2.2.1 Specifying a Markov Chain Process ….....……124

7.2.2.2 Maximum Likelihood Estimation for Markov Chains (MLE) ……..…… 125

7.3 A Scenarios-based Assessment ……..…126

7.4 Defining the Final Results- The Relative Safety Improvement …….127

7.5 Illustrative Example ……...……128

7.5.1 The EBS SPL System ……128
7.5.2 The implementation of the Assessment Model ……..……130
7.5.3 The calculations of the Safety Assessment Process ……..…..132
7.5.4 The Results and Discussion ……..……..……135
7.6 Chapter Summary ……..……135
CHAPTER VIII: IMPLEMENTATION AND EVALUATION 136
8.1 Introduction ……136
8.2 The Case Study 1 ……...……137
8.2.1 The EBS SPL Architecture Development Life Cycle …..…..138
8.2.1.1 The method Process ……138
8.2.1.2 The Inputs of the process ……138
8.2.1.3 Applying the Development Process Steps …..……144
8.2.2 The description of the developed EBS SPL Architecture …...154
8.3 The Case Study 2 ……...……156
8.3.1 Safety and Security Issues ……..……156
8.3.2 Safety and Security Risk Scenarios ……..……..……157
8.3.3 The Statechart Models ……158
8.3.4 The description of the developed Microwave Oven SPLA.…159
8.4 The Evaluation ……..……..……162
8.4.1 The Evaluation-Using Individual Risk Scenarios-With Example 1 ……..……...…163
8.4.2 The Evaluation Using Individual Risk Scenarios-With Example 2 ……..……...…169
8.4.3 The Final Results and Discussion ……...……172
8.5 Chapter Summary ……..……175
CHAPTER IX: CONCLUSION 176
9.1 Thesis Summary ……176
9.2 Thesis Contributions ……..……178

9.3 Future Work ……..……..……179

LIST OF PUBLICATIONS ……..……181

REFERENCES ……...……182

LIST OF TABLES

2.1	Papers on architecture design approaches (high level taxonomy/a broad classification) ……..	28
2.2	Papers on quality-oriented architecture design approaches ……	32
2.3	Papers on software product line architecture design approaches …....	34
2.4	Distribution over research focus ……	37
2.5	Papers on SPLA design. …….	37
2.6	Papers on Quality-based SPLA design……	39
2.7	Papers on Safety-based SPLA design…….	39
4.1	A comparison table between a more related works……	62
7.1	The abbreviations of the Markov states……..…….	128
8.1	Feature/Use Case Dependencies……..……	138
8.2	The abbreviations of the Markov states……..…….	160
8.3	The probability of the system to be in each state for scenario 1 that before and after using the safety and security pattern-Example 1..….	164
8.4	The probability of the system to be in each state for scenario 1 that before and after using the safety and security pattern-Example 2..….	167

LIST OF FIGURES

2.1 : Product Line Architecture for a Microwave Oven [18] ……..... 15

2.2 : High level taxonomy of Architecture Design Approaches …… 26

2.3 : The taxonomy Related to SPLA Design Approaches ..……..…… 26

2.4 : Map of research focus on software product line design. Research focus on the Y axis; contribution type on the left side of the X axis, and research type on the right side of the X axis ……..…….…..... 40

3.1 : Elements of design of a “real world” research project (based on (Mustapi′c, 2004)(Robson, 2002)) ……..…… 47

4.1 : An overview of our proposed method (SSPLA) ……..…..…… 68

4.2 : Door Control Software product line use cases ……..…….……. 77

4.3 : The Feature Model foe Door Control Software product line ……..... 78

4.4 : A part of the statechart specifications for Door Control PL System..... 80

4.5 : Abstract view of safety-driven statechart specification of the system …….. 82

4.6 : The architecture of the Door Control PL developed by our proposed method …..…… 83

4.7 : Architecture of the Door Control PL developed by traditional method (Feng and Lutz, 2005)…….…….. 84

5.1 : Safety-driven design pattern of statechart -the structure of the solution in term of statechart diagram ……... 97

5.2 : Safety-driven design pattern of statechart-the design solution structure in UML notation …….…….. 98

6.1 : Software Architecture Pattern development model for Safety with Security Control …… 107

6.2 : Three-step authentication for secure connection between external entities (user devices) and ECUs (CAN) (Han, Weimerskirch and Shin, 2014)……... 110

6.3 : The Safety and Security Design Pattern-the structure of the solution in term of statechart diagram ……..……..... 113

6.4 : The Safety and Security Design Pattern-the design solution structure in UML notation ……..... 114

7.1 : The Safety Assessment Model Steps…… 119

7.2 : The Statechart Design Model of Automated Electromechanical Braking System (EBS) Software Product Line after Using the Pattern ……..... 125

7.3 : The Statechart Design Model of Automated Electromechanical Braking System (EBS) Software Product Line after Using the Pattern …..……. 126

7.4 : A compact Discrete Time Markov Chain Family of the EBS product line, designed normally without safety control and before using our safety design pattern ……..……...……. 127

7.5 : A compact Discrete Time Markov Chain Family of the EBS product line, designed with safety control or after using our safety design pattern…….. 127

7.6 : The Markov model of the common behavior of EBS product line without using the proposed safety design pattern ……...…….. 128

7.7 : The Markov model of the common behavior of EBS product line after using the proposed safety design pattern ……..... 130

8.1 : EBS Product Line Use Cases Model……..... 136

8.2 : The feature model of EBS product line Using UML Notations…..……... 138

8.3 : The Statechart Model of the EBS SPL before using safety control/without using our proposed safety and security design pattern……. 139

8.4 : The Statechart Model of the EBS software product line after using our safety and security pattern…… 145

8.5 : The Statechart Model of the Scenario 1 (Scenario of usage)- before using our safety and security pattern- Example 1 …….. 146

8.6 : The Statechart Model of the Scenario 1 (Scenario of usage)- after using our safety and security pattern- Example 1 …… 147

8.7 : The Statechart Model of the Scenario 2 (Scenario of usage)- before using our safety and security pattern- Example 1 …… 148

8.8 : The Statechart Model of the Scenario 2 (Scenario of usage)- after using our safety and security pattern- Example 1 …… 149

8.9 : An abstract view of the EBS product line architecture after using our safety design pattern- A design solution structure in UML notation ……. 151

8.10 : The Statechart Model of the Smart Microwave Oven product line after Using our Safety Pattern …….. 156

8.11 : An abstract view of the Microwave Oven product line architecture after using our safety design pattern- A design solution structure in UML notation 157

8.12 : The Markov model of scenario 1 before using the safety security pattern- 160

Example 1 ……

8.13 : The Markov chain with the transition probabilities - scenario 1 before using the safety and security pattern-Example 1 ……. 161

8.14 : An equation containing the produced Transition Probabilities Matrixes of Scenario 1 before using the safety and security pattern-Example 1……. 161

8.15 : The Markov model with transition probabilities of scenario 1 after using the safety security pattern-Example 1 ……. 163

8.16 : An equation containing the produced Transition Probabilities Matrixes of Scenario 1 after using the safety and security pattern-Example 1…… 163

8.17 : Comparison between the probability of the system to be in unsafe state in each scenario before and after using the pattern-Example 1 …… 164

8.18 : The Markov model with transition probabilities of scenario 1 after using the safety security pattern-Example 2 …… 166

8.19 : An equation containing the produced Transition Probabilities Matrixes of Scenario 1 after using the safety and security pattern-Example 2 ……... 166

8.20 : Comparison between the probability of the system to be in unsafe state in each scenario before and after using the pattern-Example 2 …… 167

8.21 : The Relative Safety Improvement after using the developed pattern-Example 1 …….. 170

8.22 : The Relative Safety Improvement after using the developed pattern-Example 2 …… 171

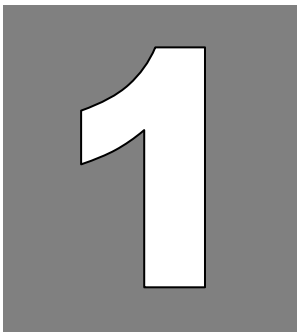
LIST OF ABBREVIATIONS

ADV+	Advantages
C.C	Cruise Control system
CAN	Controller Area Network
CHASSIS	Combined Harm Assessment of Safety and Security for Information Systems
CPS	Cyber-Physical Systems
CU	Control Unit
CVs	Commonalities and Variabilities
CVA	Commonality and Variability Analysis
DFs	Design Faults
DoS	Denial of service
DTMCF	Discrete Time Markov Chain Family
DTMC	Discrete Time Markov Chain
EBS	Electromechanical Braking System
ECUs	
FFA	Functional Failure Analysis
FMEA	Failure Mode and Effect Analysis
FSM	Finite State Machine
FTA	Fault Tree Analysis
GP	Gateway Pattern
ICEAS2017	Some of the content in this Chapter has been presented at 1st International Conference on Engineering and Applied Sciences, ICEAS2017,Red See, Sudan
ICOM2019	7th IEEE International Conference on Mechatronics Engineering, ICOM2019, Putrajaya, Malaysia

ID	IDentification
IEEE	Institute of Electronic and Electrical Engineers
IoT	Internet of things
MBD	Model-Based Design
MCs	Markov Chains
MLE	Maximum Likelihood Estimation for Markov Chains
MOODS	Models for Object-Oriented Design of State
OO	Object-Oriented
OODP	Object-oriented Design Pattern
PLUS	Product Line UML-based Software
QADA	Quality-driven Architecture Design and quality Analysis
RA	Reference Architecture
REF	Reference
RFs	Random Failures
RI	Relative Improvement
RQ	Research Question
RSI	Relative Safety Improvement
SAD	State-driven Architectural Design
SAM	Safety Assessment Model
SDP	Safety-driven design pattern
SE	Smart Environments (SE)
SGP	Security Gateway Pattern
SHARD	Software Hazard Analysis and Resolution in Design
SPL	Software Product Line
SPLA	Software Product Line Architecture
SSPLA	Safety-driven Software Product Line Architecture

TPM	Transitions Probabilities Matrix
TCAS	Aircraft Collision Avoidance System
UML	Unified Modeling Language
VP	Variation Point
XCA	Extended Commonality and Variability Analysis

CHAPTER I



INTRODUCTION

1.1 Introduction

At the last decades, technological developments have enabled to be taken classic systems place by automatic and advanced systems (Karthika, Rahamtula and Anusha, 2018). Cyber-physical systems (CPS) and Internet of Things (IoT) have distinct origins but overlapping definitions and both combine the word embedded systems (Burns, 2019). These systems contain computational (software), communication and physical components. However, such systems are at least partially controlled by software. The software has a major role and responsibility in such systems. The difficult design problems are often assumed to be readily solved using software; and the software must compensate for any deficiencies in hardware platforms (Sommerville, 2009), (Bures T. et al, 2017).

As the high quality, short delivery time, and high productivity have become more and more important in developing embedded software for modern products (Nagamine, Nakajima and Kuno, 2016), the Software Product-Line (SPL) and reusable software components are suitable approaches for such systems, which are often re-engineered from existing systems. A successful SPL supports systematic software reuse and reduces the development effort, meanwhile, improves the quality of the member products.

Software architecture design is one of the critical steps in software development process. Developing a reference architecture which represents the base structure of

the member products is the main task of the software product line architecture design.

The safety is considered one of the most critical issues in the design of the modern systems, specifically the cyber-physical systems (CPS). And as product-line engineering becomes more widespread, more safety-critical software product lines are being built.

With the increasing attention of software safety, how to improve software safety has already become a more important concerned issue, especially for the safety-critical systems (Huang, 2013). Safety-based design at architecture level can effectively improve software or system safety. Safety-based methods have received increasing attention and have been well developed for single system architecture designs. However, the safety-based design methods are limited in SPLs because of the complexity and variabilities existing in SPL architectures.

For that, this work searches to define an effective and efficient method to enhance and manage the architecture design process of the software product line systems. The research firstly focused on how to consider safety in SPL in the architecture design phase and proposed a safety-driven SPL architecture design method. The key aspect of this method is the use of the concept, design patterns, which improves the design process. A number of efforts have been made to make the method effective and efficient. One of the critical effort is making the design process activities of the method compatible and consistent (e.g. state-based design).

As the Unified Modeling Language (UML) statechart diagram is a powerful tool for specifying the dynamic behavior of reactive objects, this facility can be used to describe the system behavior in term of safety. Based on this facility the proposed SPL architecture design method mentioned above is configured and adapted to be state-based architecture design method. This adaptation results in a new state-driven architectural design method. This adaptation means that most of the process steps should be based on or around the statechat semantic.

Design patterns can be used to enhance the design of systems in different application domains. It is evidence that a pattern based development of the reference architecture can support the development and application process of the product lines. In this context a new statechart-based safety design pattern is developed. The proposed design pattern is called ***Safety Design Pattern of Statechat***. This pattern

extends capabilities of both the statecharts design patterns and safety patterns. The pattern allows an object to alter its behavior and change its internal state when there is a safety violation, and to protect it from introducing in unsafe states. The result is an object-oriented design pattern which handles the safety attribute. To extend the statechart pattern to capture the variability existing in the SPLs and because the complexities exist in the PL the thesis proposed using of parameterization approach (proposed in (Gomaa, 2011)).

Due to the tight interplay between safety and security, combining safety and security in the engineering process has become a critical process. In this context, the thesis aims at addressing the influence of security issues in the safety design process using patterns. ***A Pattern Development Approach*** that interlinks safety and security patterns has been proposed. This approach is then used to enhance our proposed safety design pattern of statechart (presented in Chapter 5) to address the security in the pattern (see Sec. 6.4). This developed version is considered as ***a new safety and security pattern***.

To evaluate our work, a simplified safety assessment model (SAM) is developed. Finally, we have motivated our work with the help of two case studies. These two case studies are to illustrate how all these works can improve the safety design of the SPLAs. The evaluation results show that there is a considerable improvement in the system safety design after applying our work.

1.2 Problem Statement and Its Significance.

Software has become responsible for most of the critical functions of complex systems. The safety is considered one of the most critical issues in the design of the modern systems (e.g. cyber-physical systems (CPS), Internet of things (IoT)). The number of products with embedded software increases across all application areas continuously. Thus, the complexity between the hardware and software is steadily increasing. This leads to an increment of software defects. Therefore, effective approaches are needed to ensure the product quality. The Software Product-Line (SPL) and reusable software components are suitable approaches for these systems, which are often re-engineered from existing systems.

Software architecture design is one of the critical steps in software development process (Sommerville, 2009). Developing a reference architecture which represents the base structure of the member products is the main task of the software product line architecture design (Systems, 2000).

With the increasing attention of software safety, how to improve software safety has already become a more important concerned issue, especially for the safety-critical systems. Currently, the influence of the architecture in assurance of software safety is being increasingly recognized. However, the safety-based architectural design methods are limited in SPLs because of the complexity and variabilities existing in SPL architectures. For that, this work seeks to find an efficient and effective method that can be used into the design process of the safety-critical software product line architectures which enhances and manages the safety of software product lines.

The significance of this research is that it presents several significant advances to the fields of safety engineering and design. It presents a process of concurrently developing a system concept from the safety and functional perspective. We believe this work presents an important step in making the design and safety processes more efficient and effective for the software product line. Finally, this research will benefit both architects and safety engineers who can design software product line architectures or develop software product in domain of embedded systems and cyber-physical systems.

1.3 Research Objectives

From the literature it is clear that the existing methods for the safety-based architectural design are not adequate to enhance the architectures design of the modern software product line systems. The limitation is because of the complexity and variabilities existing in SPL architectures. The main objective of this research is to find an efficient and effective architectural design method that can be used into the design process of the safety-critical software product line architectures.

Other specific objectives are highlighted as follows:

- Developing safety design pattern that can be adapted and used in the design of the safety-critical SPLA.

- Enhancing the safety design to address the influence of the security issues on the safety using patterns.
- Defining a safety assessment model to show the Relative Improvement (RI) in the safety design of the SPLA after using the proposed method and the safety design pattern.

1.4 Research Questions

Based on the objectives described in the previous section, there are some research questions have been derived.

The main question that is addressed in this research is: how can we define efficient and effective architectural design method that can be used into the design process of the safety-critical software product line architectures?

There are additional sub-questions as follows:

- How can we develop an Object-oriented Design Pattern (OODP) to address the safety attribute in the system?
- How can the safety design be enhanced to address the influence of the security issues on the safety using patterns?
- How to effectively evaluate and assess the safety-driven software product line architectures in order to show the relative safety improvement in the design after using our method as well as addressing the safety risks?

1.5 Scope of Research

From the literature it is clear that the existing methods for quality-oriented architectural design are not sufficient for software product line design. Also the safety-based design methods are limited in software product line architectures design and that because of the complexity existing in SPL architectures.

We are seeking to find an efficient and effective method that can be used into the design process of the safety-critical software product line architectures which enhances and manages the safety of software product lines. We have to use a powerful tool for specifying the dynamic behavior of reactive objects. The facility of this tool can be used to describe the system behavior in term of safety.

In order to support the design process of the safety-critical SPL architectures the work searches to extend the capabilities of both the traditional safety patterns and statechart design patterns to develop a new safety-driven design pattern of statechart.

As there is a high impact of the security on the safety, especially in the smart environments (e.g. Cyber-physical Systems and Internet of Things), the research also seeks for how to effectively address this issue in the design process of the software product line architectures.

According to above, the thesis only considers the software architectures design for safety-critical product lines and does not include the operational, maintenance and decommissioning phases of the product line lifecycle.

1.6 Thesis Structure

The rest of this thesis is organized as follows: Chapter 2 presents research background and number of past efforts related to the current work. Existing literature on architectural design for Safety-critical software product line is also surveyed. The survey includes an analysis of closely related researches for software product line architectures design. The chapter also mentions some of open research issues. The research methodology, tools, languages and case studies description are covered in Chapter 3.

Chapter 4 defines one of the main contributions of this thesis. It describes safety-driven SPL architecture design (SSPLA) method proposed for the designing process of the safety-critical software product line architectures. Also the chapter presents the adaptation of this method to be a statechart-centric method which is then called State-driven Architectural (SAD) Design Method for Safety-critical Software Product Lines. At the end, the chapter shows how to address the variability of the product line in the statechart.

The developed safety design pattern—Safety-driven design pattern of statecharts which constitutes an essential part of this work is presented in Chapter 5. This pattern extends capabilities of both the statecharts design patterns and safety patterns. Chapter 6 describes our solution of how to address the influence of the security issues on the safety design. It presents a systematic pattern development approach which proposed to interlink safety and security patterns which has been developed in

order to enhance the safety patterns. It also describes the using of this pattern approach to develop a new safety and security pattern.

Chapter 7 describes a proposed safety assessment model. This model is a simplified mathematical model for safety assessment of the product lines architectures. Adapting this assessment model to be a scenario-based assessment method and adding a metric or the concept of Relative Safety Improvement RSI) are also presented in this chapter. Chapter 8 describes the implementation and evaluation processes. The chapter presents an evaluation of the main contributions of the research presented in the previous chapters. This evaluation is carried out by means of tools support and case studies. The chapter also presents the final results of the evaluation process as well as a short discussion on the all results.

Finally, the thesis is concluded by summarizing the main conclusions of the research and contributions of the thesis as well as providing some future recommendations in Chapter 9.

1.7 Bibliographic Notes

Some parts of this thesis are based on work that has been previously presented in earlier publications. The survey study which considered as a systematic literature review of software product line architecture design was published in (Mozamil Elgodbe and Ammar, 2016). The new safety-driven design method for software product line architectures was published in (Mozamil Ebnauf and Hany H. Ammar, 2017). The adaptation and configuration to this method to be state-based architectural design method was published in (Ebnauf and Al., 2019). The new safety design pattern of statechart was published in (Ebnauf and Al., 2019). A simplified safety assessment model that is used to show the safety improvement in the design after using our work and also to facilitate the evaluation process of the architecture for safety-critical software product line systems was published in (Ebnauf and Al., 2019).

CHAPTER II

LITERATURE REVIEW

2.1 Introduction

(Some of the content in this Chapter has been published in the International Journal of Computer Applications Technology and Research, Volume 5, Issue 10, pp 627 - 640, ISSN: 2319-8656, Oct. 2016), (Mozamil Elgodbe and Ammar, 2016).

The work presented in this thesis is based on the overlapping areas of software product-line architectures design, safety-based design, architectural design patterns, statechart semantic and safety with security control. The focus is on the use of state-based design for safety-critical software product line architectures.

To connect the knowledge and provide a comprehensive overview of the current state of the art, this chapter provides a systematic literature review of the existing research on software product line architecture (SPLA) design based on quality attributes. The chapter primarily aims at surveying existing research on software product line architecture (SPLA) design, and to give an overview of the intersection of the areas of software product line architecture design and safety-driven design in order to classifying existing work, and discover open issues for further research. Also this chapter presents the basic concepts of the architectural design patterns and offers a brief literature review of statechart design patterns.

In general, with the chapter we aim to achieve the following objectives:

- to give an overview of the intersection of the areas of software product line architecture design and safety attribute.
- provide a basic classification framework in form of a taxonomy to classify existing architecture design approaches.

- offer a brief literature review of statechart design patterns.
- point out current trends, gaps, and directions for future research.

The rest of the chapter is organized as follow. First, Section 2.2 presents the overview of cyber-physical systems, software architectures design, software product line, software product lines architectures design and safety–based SPLAs design. A survey study is presented in Section 2.3 which considered as a systematic literature review of software product line architecture design. Section 2.4 presents the basic concepts of architectural design patterns as well as offering a brief literature review of statechart design patterns. Finally, Section 2.5 presents the chapter summary and identifies future research directions based on the survey results.

2.2 Background

This section reviews the research background information. In this section we present an overview of the main concepts that are frequently relevant in the context of safety, software safety, cyber-physical systems, software architectures design, software product line, software product lines architectures design and safety–based SPLAs design.

2.2.1. Safety and Software Safety

In this Sub section we present an overview of the main concepts that are frequently relevant in the context of Safety and software safety.

2.2.1.1 The Safety

Definition. “Safety is the ability of an item not to cause unacceptable consequences during its use" (Rehn, 2009).

Safety and reliability represent the main non-functional requirements that should be provided in the design of safety-critical applications (Armoush, 2010). The safety is one of the most important quality attributes of today's software and their importance is even increasing (Rehn, 2009). It is absence of catastrophic consequences on the user(s) and the environment (Armoush, 2010). The current work in systems engineering methods has focused on supporting a safety-centric design

process (Jan Bosch, 2001). The General idea of these approaches is that safety should be a driver for design.

2.2.1.2 The Safety-critical Systems

The term Safety Critical System (SCS) refers to the system which has potentially destructive power. Once such a system produced a failure, many serious consequences may be caused, such as casualties, property loss and environmental damage etc (Huang, 2013).

Recently, software application in SCS is more and more extensive, and the scale also increasingly grows. From railway transit field to the aerospace field and from the power system to the medical system, this type of software plays a key role in command and control aspect for software safety (Huang, 2013). The research in the domain of SCS safety focus on how to reduce the probability of unsafe system conditions that various SCS elements lead to, or weaken the SCS's consequences that failures produce, through using a variety of management, organization, technical measures (Huang, 2013). We can divide the safety design of SCS into structural design optimization and fault-tolerant design. The former aims to reduce software defects, while the later aims to prevent a number of the known software failure. Safety tactics can combine with the existing safety design technique to effectively guide the selection of protective mechanisms to improve the safety (Huang, 2013).

The Following are Some Examples of the Safety-critical Systems

1. The Door Control System. The Door Control System is a safety-critical product line. The software must function correctly to prevent intruders from entering and must respond correctly to life-threatening scenarios such as fires. A Smart Home system serves as an invisible housekeeper: it has sensors and agents to interact with humans and the environment to offer people convenience and safety. For example, the entrance doors can be opened only by inputting fingerprints or voiceprints.

2. ATP System. According to the definition of the CBTC system (Huang, 2013), ATP is Automatic Train Protection system, which includes the follow aspects for the safety protection function: Train self-checking, Speed measure and locate, Speed supervision, Train safe stopping, Safe direction and door control, CBTC operation mode and Runaway protection, for more details see reference ((Huang, 2013)).

3. The EBS SPL System. The EBS system is considered one of the subsystems of Cruise Control System (C.C). The main function of EBS system is to automatically stop the car or the vehicle and in safely way when there is an obstacle in front of the vehicle. In such braking systems, sensors, communication media, and actuators replace mechanical devices (Varshosaz and Khosravi, 2013). In reality, there could be malfunctions with nonzero probabilities. For example, failure of the sensors to detect obstacles in an admissible interval, possible message loss, and (Control Unit) CU failure are some examples of undesirable but possible characteristics of such systems.

4. The Smart Microwave Oven Control System. In this thesis and as a second case study, we use the Smart Microwave Oven Control Systems Software Product Line. The Smart Microwave Oven is a special home appliance that has several operations, such as setting command, setting timer, starting and so on, and that can be operated remotely. The microwave oven will form the basis of this product line, which will offer options from basic to top-of-the-line (Gomaa, 2004).

Sometimes the device may malfunction. Example, the microwave may keep cooking for an hour, which is not required by the users. Another risk example, the Microwave oven may blow up or become too hot to touch. Additionally, some of unauthorized influences (e.g. DoS- (Denial of Service), the use of IoT devices for malicious purposes) lead to failures and failures of the critical systems that are part of the IoT. For example, disconnect the line between the remote system and the Microwave oven which can lead to dangerous situations. Consequently, we need a safety and security control. The control means it can detect such malfunctioning or even attacks and deal with that by updating the state of the devices, stop it (using operate use case), and inform the user what happened. Also it can address the security issues that influence the safety of the system.

2.2.1.3 Safety Patterns and Tactics

For this sub section the following definitions are used:

A. Patterns

Definition. A pattern describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the ways in which they collaborate" (Armoush, 2010).

From the software engineering point of view, a design pattern is a description or template for how to solve a problem that can be used in many different situations (Buschmann and Maunier, 2001). It is a general reusable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code (Buschmann and Maunier, 2001).

A design pattern describes a design problem which repeatedly occurred in previous designs, and then describes the core of the solution to that problem.

B. A tactic

Definition. A tactic is a design decision for realizing quality goals at the architectural level (Rehn, 2009). Tactics are rather simple ideas. They are _ne grained but abstract and thus as opposed to patterns expressible in just a few sentences. Although tactics are fine grained, they are not atomic. They can be refined, so there is a hierarchical structure of tactics (Rehn, 2009). For example redundancy is a tactic which can be specialized by the tactics replication, functional redundancy and analytic redundancy.

Patterns tend to be much more complex, because they package several tactics and this in a more concrete way. So tactics are building blocks for patterns.

2.2.1.4 The Software Safety

While modeling software safety it is important to note that no software works in isolation. The entire system must be designed to be safe. The system components may be software, hardware, users, and the environment. All must be given consideration when developing software. All parts of the system must be safe. Functional and operational safety starts at the system level. Safety cannot be assured if efforts are focused only on software. The software can be totally free of 'bugs' and employ numerous safety features, yet the equipment can be unsafe because of how the software and all the other parts interact in the system (Swarup and Ramaiah,

2009)[27]. Thus safety and security are major issues in software engineering and their importance is even increasing (Rehn, 2009).

With the increasing attention to software safety, improving software safety has already become a more important issue, especially for safety-critical systems.

Software safety design process always starts with the system or platform hazards identified by preliminary hazard analysis.

Over the last few years, embedded systems have been increasingly used in safety critical applications where failure can have serious consequences. The design of these systems is a complex process, which is requiring the integration of common design methods both in hardware and software to fulfill functional and non-functional requirements for these safety-critical applications (Armoush, 2010).

Nowadays Software is used not only in offices and living-rooms but also in safety-critical environments and for tasks where sensitive data or huge amounts of money are involved. On the other hand software becomes more and more networked, distributed and ubiquitous. So the risk of failing or compromised software increases as well as the severity of the possible consequences. Thus safety and security are major issues in software engineering and their importance is even increasing (Rehn, 2009).

Software is an integral and increasingly complex part of modern safety critical systems. Therefore, it is essential to analyse software safety in a system context to gain a comprehensive understanding of the roles of software and to identify the software-related risks that can cause hazards in the system. Leveson (Leveson, 1991) noted that software by itself is not hazardous and cannot directly cause damage to human life or the environment; it can only contribute to hazards in a system context. Software can create hazardous system states through erroneous control of the system or by misleading the system operators when taking actions (Leveson, 2011).

2.2.1.5 The safety-based Development

Software safety assurance refers to a series of quality assurance activities during software development life cycle, which aims to eliminate the potential dangers.

The specification of safety constraints is the first step of the safety-constraint centered design approach (Swarup and Ramaiah, 2009).

To develop safe software, therefore, we first need to identify and analyse software-related hazards and the unsafe scenarios and develop the corresponding software safety requirements at the system level.

(Huang, 2013) The first step in the safety-constraint centered design approach is the specification of safety constraints (Bass, Klein and Bachmann, 2001). In hardware systems, redundancy and diversity are the most common ways to reduce hazards. Hardware detection and control includes mechanisms such as failsafe designs, self-tests, exception handling, warnings to operators or users, and reconfigurations. For software intensive safety-critical systems, software design must enforce safety constraints. Reviewers should be able to trace from requirements to code and vice versa. In addition to the specific safety constraints developed for the system being designed, the design should incorporate basic safety design principles. Safety, like any quality, must be built into the system design. The most effective way to ensure that a system will operate safely is to build safety in from the start, which means that system operation must not lead to a violation of the constraints on safe operation. System accidents result from interactions among components that lead to a violation of these constraints -- in other words, from a lack of appropriate enforcement of constraints on the interactions. Because software often acts as a controller in complex systems, it embodies or enforces the constraints by controlling the components and their interactions. Software, then, can contribute to an accident by not enforcing the appropriate constraints on behavior or by commanding behavior that violates the constraints. The requirement for software to be safe is not that it never "fails" but that it does not cause or contribute to a violation of any of the system constraints on safe behavior. This observation leads to the suggested approach to handling software in safety-critical systems, i.e., first identify the constraints on safe system behavior and then design the software to enforce those constraints. The software-specific analysis should provide specific mitigation approaches for each potential hazard identified.

2.2.2 Cyber-physical Systems

Cyber-Physical Systems (CPS) is an emerging paradigm (Paulo Leita˜o, Luis Ribeiro and Thomas Strasser, 2016). The term cyber-physical systems (CPS) refer to

a new generation of smart systems that integrate computational and physical components to implement a process in the real world. They are present in quite diverse areas, such as automotive electronics, aerospace systems, railways, telecommunication, health sector, security, fabrication equipment, smart buildings, robotics, and military applications (Shi *et al.*, 2011). They bring innovation to many industries, they have potential to integrate technologies from various sectors, transform traditional processes in several application areas, and enable new processes (Wan *et al.*, 2011).

Since the nature of CPSs is the interaction with the physical world, so they must operate dependably, safely, securely, and efficiently and in real-time (Peter, 2011). The safety is considered one of the most critical issues in such systems. These systems are directly connected to the physical environment and have an immediate impact on the environment (Amorim *et al.*, 2017).

In the other hand, CPSs contain computational (software), communication and physical components however these systems are at least partially controlled by software. The software has a major role and responsibility in such systems (Bures T. et al, 2017). The difficult design problems are often assumed to be readily solved using software; and the software must compensate for any deficiencies in hardware platforms (Bures T. et al, 2017) and (Sommerville, 2009).

2.2.3 Software Architectures Design

One of the critical steps in software development process is software architecture design (Sommerville, 2009). The output of this process is software architecture (Sommerville, 2009).

Importance of software architecture -“Software architecture is not only concerned with structure and behavior, but also with usage, functionality, performance, resilience, reuse, comprehensibility, economic and technology constraints and tradeoffs” - The Rational Unified Process, 2002.

Software architecture is the structure of the software system. "It describes the software elements, their characteristics and how interact with each other" (Len Bass and Paul Clements and Rick Kazman, 2003) and (Systems, 2000). Qualified software

architecture provides a blueprint for system construction and composition. It is a main factor to a successful software development (L Tan, Lin and Ye, 2012). There are many challenges in software architecture design for example, modeling the non-functional requirements, especially those requirements on the quality of the software.

Non-functional requirements and quality attributes (e.g. maintainability, performance, reliability, safety and product evolution) are important parameters of software products. Quality requirements of a system serve as a bridge between business goals and software architectures (L Tan, Lin and Ye, 2012). There is a major role of Software architecture in the determination of software quality (Medvidovic, Malek and Mikic-Rakic, 2003; Peter Wallin, 2012).

2.2.4 Software Product Line

Software product line (SPL) engineering is about developing a collection of systems which share great commonalities (Bayer, Flege and Gacek, 2000; L Tan, Lin and Ye, 2012).

Software product line is defined as "A set of software-intensive systems sharing a common managed set of features that satisfy the specific needs of a particular market segment or mission (Pär J Ågerfalk, 2006)". These systems are developed from a common core of assets (e.g. a common architecture) in a prescribed way.

The idea of SPL was initiated by Parnas (Parnas, 1976) and has been further developed by Kang et al (Kang *et al.*, 1998). The concept of SPL is to discover both commonalities and variabilities (CVs) among member products of the product family.

(Liliana Dobrica, Eila Niemela,2003) (Liliana Dobrica, 2000).Product-line (PL) and reusable software components are suitable approaches for embedded systems, which are often re-engineered from existing systems. Important issues in the development and maintenance of these software systems are functionality and quality. Although there are some similarities between embedded systems regarding quality attributes, there are also differences. If a quality attribute is important to one product-line domain, it does not necessarily mean it is important to another one.

2.2.5 The Complexity in Software Product Lines Development

In general, the complexity of systems makes the software development activities difficult in practice. There are number of issues or "practice areas" affect an organization's success in fielding a software product line (Len Bass and Paul Clements and Rick Kazman, 2003). These issues may also face the development of the single-system, but in the product line context these issues take on a new dimension (Len Bass and Paul Clements and Rick Kazman, 2003). Examples of these practice areas are architecture definition and configuration management (Andrade, 2013).

The large scale and complexity of today's software-intensive systems make the variability management become increasingly complex to conduct (Bashroush *et al.*, 2017).

The term architecture refers to the structure of a system, consisting of software elements, externally visible properties, and the relationships among elements (Bass et al., 2003a) (Bass, Klein and Bachmann, 2001). Developing a reference architecture which represents the base structure of the member products is the main task of the software product line architecture design. As we are dealing with families of products the architecture model contains the functionality for a whole family of variants.

The PLA is considered to be a key aspect in SPL engineering, through which the complexity of a variability-based environment can be managed.

Architecture design is an important activity for any project but, it needs to emphasize variation points in a software product line. This design process is more complex for a software product line that because of the variability existing in the product line.

The very high number of possible combinations emphasizes the high variability and the great complexity of the SPLs that can be managed with the effective approaches (Urli, Blay-Fornarino and Collet, 2014).

In the context of product line architecture model, Mannion and Camara in (Mannion, 2002) and (Mannion, M., Camara, 2004) argue that constructing and validating a product line model is difficult due to the size and linkage complexity of such models.

Variability models tend to be very large in size, in many cases comprising thousands of features, and complex in nature due to the myriad of relationships that could exist among the features (Bashroush *et al.*, 2017).

In fact, the creation of a model-based product line from a historically grown product family requires large effort and that due to size and complexity existing in the product family (Polzer *et al.*, 2012).

In the testing aspect, the complexity of software makes testing a challenging process because practically impossible to test all possible execution paths of software.

For that large effort is required to address this complexity. The variability management mechanism is one of these efforts.

2.2.6 Software Product Lines Architectures Design

Two main steps when the decision to initiate a software product line has been taken, the domain analysis as a first step to describe the variability in the requirements, the second important step is the definition of the product line software architecture (Heymans and Trigaux, 2003).

Developing a reference architecture which represents the base structure of the member products is the main task of the software product line architecture design (Systems, 2000).

The software product line architecture (SPLA) (L Tan, Lin and Ye, 2012) provides a coarse grain picture of structure in the software product family. It initiates the architecture design for the member product. In the architecture design of a product line, it must accommodate the variability and dependency of functionality in the components that is derived from the feature model (L Tan, Lin and Ye, 2012).

Recently, software product line architectures have been used successfully in industry for building families of systems of related products, maximizing reuse, and exploiting their variable and configurable options (Capilla *et al.*, 2014)(Behjati *et al.*, 2013)(White *et al.*, 2013).

The creation and validation of product line software architectures are inherently more complex than those of software architectures for single systems.

The importance of reference architectures increases by the increase of different domains in which embedded systems become a dominant part, and in which software becomes the most important component. This does not happen only in ‘‘classical’’ domains such as automotive industry, avionics or telecommunication, but also in new areas such as Internet of Things, e-health, environment, smart houses and smart cites, etc. The Reference Architectures improve the reusability of methods and artifacts, but also brings new research challenges such as finding the principles and methods for the Reference Architecture (RA) deployment with guarantees for domain-specific functional and extra-functional properties (Crnkovic and Stafford, 2013).

There are two main role of the software product line architecture as follow: first, it must describe the commonalities and variabilities of the products contained in the software product line and, secondly, it must provide a common overall structure (Heymans and Trigaux, 2003).

Figure 2.1 illustrates examples of product line architecture for embedded system (Ammar, 2013), which is the Product Line Architecture for a Microwave Oven.

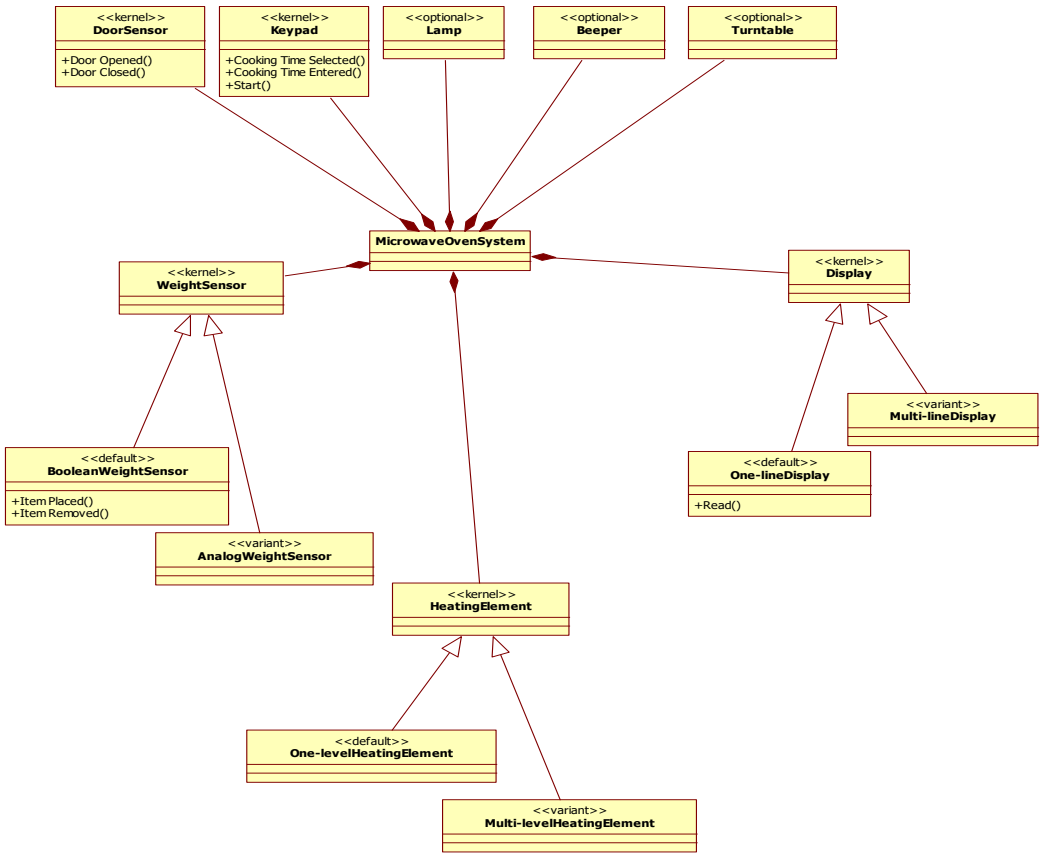


Figure 2.1: Product line architecture for a Microwave Oven (Ammar, 2013)

Definition of Terms Used in the Example Class Diagram:

Kernel: Kernel in product lines represents the mandatory features for the product line members. i.e.: they cannot be omitted in products.

- The stereotype <<kernel>> is used to specify Kernel in UML class diagrams.

Optional: Optionality in product lines means that some features are elective for the product line members, which means they can be omitted in some products and included in others.

- The stereotype <<optional>> is used to specify optionality in UML class diagrams.
- The optionality can concern classes, packages, attributes or operations. So the <<optional>> stereotype can be applied to Classifier, Package and Feature meta-classes.

Variant: Variant classes are modeled using UML inheritance and stereotypes. Each variation point will be defined by an abstract class and a set of subclasses.

- The abstract class will be defined with the stereotype <<variant>> and
- each subclass will be stereotyped <<variant>>, or <<optional>>, the default value being variant.

2.2.7 Safety–based SPLAs Design

"From a safety viewpoint, the software architecture is where the basic safety strategy is developed in the software." It is very significant to study how the non-functional attribute "safety" to be described, analyzed and verified during the architecture construction process (Capilla *et al.*, 2014).

Software safety assurance refers to a series of quality assurance activities during software development life cycle, which aims to eliminate the potential dangers.

The specification of safety constraints is the first step of the safety-constraint centered design approach (Ramakrishna and Satish et. al., 1996).

While modeling software safety it is important to note that no software works in isolation. The entire system must be designed to be safe. The system components may be software, hardware, users, and the environment. All must be given consideration when developing software. All parts of the system must be safe.

Functional and operational safety starts at the system level. Safety cannot be assured if efforts are focused only on software. The software can be totally free of 'bugs' and employ numerous safety features, yet the equipment can be unsafe because of how the software and all the other parts interact in the system (Swarup and Ramaiah, 2009).

Currently, the influence of architecture in assurance of software safety is being increasingly recognized. As product-line engineering becomes more widespread, more safety-critical software product lines are being built (Engström and Runeson, 2011).

As product-line engineering becomes more widespread, more safety-critical software product lines are being built (Feng and Lutz, 2005).

The study shown that, nowadays, there are various design methods available and each is focusing on certain perspective of architecture design. Especially, safety-based methods have received a lot of attentions and have been well developed for single system architecture design. However, the use of safety-based design methods is limited in software product line (SPL) because of the complexity and variabilities existing in SPL architecture. In the next lines we briefly present and discuss some of the related works.

(Donald Firesmith, 2004) (Firesmith, 2004). His work concerned with the Engineering Safety Requirements, Safety Constraints, and Safety-Critical Requirements. He used the concept of a quality model to define safety as a quality factor. Thus, safety (like security and survivability) is a kind of defensibility, which is a kind of dependability, which is a kind of quality. Next, he discussed the structure of quality requirements and showed how safety requirements can be engineered based on safety's numerous quality subfactors. Then, he defined and discussed safety constraints (i.e., mandated safeguards) and safety-critical requirements (i.e., functional, data, and interface requirements that can cause accidents if not implemented correctly).

However, no tasks or attentions related to how design the software product line architecture based on safety analysis.

(David C. Jensen, Irem Y. Tumer, 2013) (Jensen and Tumer, 2013), their work presented a method of explicit inclusion of safety into a model-based design (MBD) for cyber physical systems. This approach enables an analysis where component-

level failures can be mapped to potential system-level hazards. This work presented a method of representing the safety property of a system by the introduction of the concept termed "safety function". Further, the function of achieving safety is mapped to the performance functions of the system. They presented a process of concurrently developing a system concept from the safety and functional perspective. The end result of this process is a system architecture where components of the system are explicitly mapped to both the functions they perform and the role they play in ensuring safe system operation. The benefit of this approach is having a system representation that allows for analysis of critical events and off-nominal component behavior to identify potential losses in function and safety constraint violations. The perspective of these approaches is that safety should be a driver for design. Thus the objective of this work is to introduce a safety-centric method of developing a design based on the functional modeling paradigm.

However, this work does not address the design of software architecture. The proposed method focus on inclusion of safety into design level in general without focusing on a specific design activity.

(Yuling Huang, 2013) (Huang, 2013). Safety design at the architecture level can effectively improve software or system safety. This work addresses the problem of how to consider safety in software architecture design phase and proposed a safety-oriented software architecture design approach. Through the system hazard analysis, this design approach uses the selected combination of safety tactics to effectively improve the software or system safety, providing a new way of thinking for software safety architecture design.

However, this work does not take in account the concepts of a family of architectures, namely, Product line Architectures.

Although a considerable number of safety analysis techniques have been proposed to aid software design such as Software Hazard Analysis and Resolution in Design (SHARD) (Fenelon *et al.*, 1994), there is little analysis work focusing on an architectural level to aid software architecture design.

(Weihang Wu, Tim Kelly, July 2004) (Wu and Kelly, 2004) Safety design concerns the identification and management of hazards. Hazards are caused by failures. A distinction is often made between causes of failures in physical devices (e.g., random failures (RFs)) and failures in software. Software does not fail

randomly; its “failures” are due to its systematic nature (e.g., design faults (DFs)). In software systems, safety is thus achieved by avoiding, or protecting against, these failures. As a result, the focus of attention in their analytic model is the relationship between the safety attribute and software architecture with respect to failures (Wu and Kelly, 2004).

(Lei Tan, Yuqing Lin, Huilin Ye, 2012) (Lei Tan, Lin and Ye, 2012). Quality-driven Architecture Design and quality Analysis (QADA) is a traceable quality based method to design and evaluate software architecture. QADA contains scenario-based quality analysis to evaluate if the architecture design options meet the quality requirements. QADA consists of three viewpoints: structural view, behavior view, and deployment view at two levels of abstractions: conceptual level and concrete level. It contains several views at different levels to separate concerns and it provides a quality-driven link between software requirement and architecture. This work extended QADA method by adding an extra view to improve this quality based PLA design method.

In this framework, the quality attributes of a software system will be taken into account in the early stage of architecture design and the reference architecture of SPL will be elicited based on quality-related consideration.

However, their work is just extending to QADA method by adding an extra view to improve this quality based PLA design method. This work may be a direction of more open researches, especially in field of product line, that by focusing on a specific quality attribute or other architectural attributes.

2.2.8 Scenario-based Architecture Design for SPL

It is evidence that the scenario-based development is an effective mechanism which supports the overall the software development process. Numbers of scenario-based development methods for software architecture have been developed. The methods include different aspects of the development activities, e.g. method for architecture design, evaluation and reconstruction, (Lei Tan, Yuqing Lin, Huilin Ye, 2012) (Lei Tan, Lin and Ye, 2012). Quality-driven Architecture Design and quality Analysis (QADA) is a traceable quality based method to design and evaluate

software architecture. QADA contains scenario-based quality analysis to evaluate if the architecture design options meet the quality requirements.

This section describes how our proposed method presented in this thesis supports the scenario-base development. In this thesis, a safety-driven architectural design method for software product line architecture is presented (Chapter 4). It is a scenario-oriented method and it will done activities of development in an iterative manner of the design product line architecture. In this method the architecture is created in a number of iterations by stepwise application of scenarios and by using proven solutions to recurring problems such as architectural patterns. Iterations are performed until all scenarios have been applied and no problems arose from the final assessment of the architecture. We need to extract a limited number of scenarios that should be used in the iterations. Indeed, the scenarios describe the functional and safety requirements of the product family the architecture is designed for. In the context of product families, scenarios are generic that they do not only capture common but also variable requirements of the instances in the product family.

The architecture creation process, however, can only be finished once all scenarios have been applied successfully (i.e. no problems have been detected in the assessment) and all assessment criteria have been fulfilled. If the assessment of the architecture showed that at least one of the defined assessment criteria was not fulfilled, the underlying problem has to be examined in order to determine how the architecture creation process can continue. In the best case, changing just the last iteration may solve the problem. In the worst case, if a solution supporting all scenarios in all variants even exists it is necessary to track back to the first iteration. However, if the assessment doesn't have any problem, the next scenario in the priority list is entered into next cycle and reconstruction cycle continues.

The evaluation process is one of the critical steps in our proposed design method for the software product line architectures (Chapter 4). To facilitate the evaluation process of the architecture for safety-critical software product line systems a simplified safety assessment model is developed. This assessment model is also used to show the safety improvement in the design after using our work.

The safety assessment model (Chapter 7) is interconnected with the system model and potential attack and failure (risk) scenarios are described through the models. We use the statechart models to describe the system design or the risk scenarios. And

then we use these statechart models to define the Markov chain corresponding to each model (e.g. Fig. 8.15). These Markov models are then used in the mathematical calculations in the assessment process, see Sec. 8.4.

As we mentioned in the introduction, that this thesis aims to make the overall architecture development activities compatible and consistent with each other. In this context and in order to achieve this objective, numbers of efforts are made, (e.g. state and scenario-based method, state and scenario-based safety assessment model).

2.2.9 Safety-related Test Cases

Often the test cases are define early that because we need creating a document as a plan to asses or evaluate the architecture. The output of this step is a definition of architecture evaluation plan. This plan is used to evaluate the architecture in the end of the each iteration and in the last evolution of the architecture. Integration and system test cases should also be based on use cases. statecharts can also be used to depict the states and transitions for a state-dependent use case (Gomaa, 2011).

Because the proposed method process presented in this thesis is a safety-driven design, the most process steps (or activities) are based on the safety attribute. For instance, define test cases is defined based on the safety attribute, select a safety-driven architectural pattern(s), evaluate the architecture if met the safety requirements or not.

In a product line context, instance- and family-specific test cases have to be distinguished. The test plans, test cases or test data must consider variation points and multiple instances of the product line. The domain test cases for system tests must be defined. A set of test cases for the parts of the architecture should be designed based on number of artifacts in the domain. One of these artifacts is a requirements document. The variability in the domain artifacts should be preserved i.e. should be adequately introduced into the test case design (Pohl, Böckle and van der Linden, 2005).

In our method and in general, the given architecture is checked with respect to functional and quality requirements and the achievement of business goals. In case an architecture assessment (or evaluation) should be performed at the end of the iteration, assessment (or evaluation) criteria have to be defined according to the

business and quality goals. Defining assessment criteria before the actual design begins has several advantages such as a better understanding of the requirements and avoidance of specifying criteria that, due to an already influenced perspective, merely support what has been developed. For each group of scenarios, test cases are defined that will be used to evaluate the architecture at the end of each iteration. Here, the main assessment criterion, quality goal is the safety attribute. This safety attribute play a central, critical role for the appropriateness of reference architecture. Define test cases is based on safety. However, it is extremely difficult to assess the degree to which this attribute is achieved by a given architecture.

2.3 Literature Survey

This section presents a survey study which considered as a systematic literature review of the existing research on Software Product Line Architecture (SPLA) design based on quality attributes

The section 2.3.1 outlines the research method and the underlying protocol for the systematic literature review. The first contribution of this study, a taxonomy of architecture design approaches that has been derived from an iterative analysis of the existing research literature is presented in Section 2.3.2. The second contribution, a classification of existing architecture design approaches according to this taxonomy, is presented in Section 2.3.3. Finally, Section 2.3.4 provides a discussion and results of the survey.

2.3.1 Research Method

The stages involved in our literature review are structured into three phases: planning, conducting, and reporting the review, based on the guidelines proposed by

Kitchenham (Kitchenham, 2004).

A systematic mapping study is launched to find as much literature as possible, and the 22 papers found are classified with respect to focus, research type and contribution type.

Based on the guidelines, Kitchenham (Kitchenham, 2004), this section details the research questions, the performed research steps, and the protocol of the literature review. First, Section 2.3.1.1 describes the research questions underlying our survey.

Then, Section 2.3.1.2 derives the research tasks we conducted, and thus describes our procedure. Section 2.3.1.3 then details the literature search step and highlights the inclusion and exclusion criteria. Finally, Section 2.3.1.4 discusses threats to the validity of our study.

However, the reported results are fragmented over different research communities, multiple system domains, and multiple quality attributes. Based on this survey, a taxonomy has been created which is used to classify the existing research. Furthermore, the systematic analysis of the research literature provided in this review aims to help the research community in consolidating the existing research efforts and deriving a research agenda for future developments.

2.3.1.1 Research Questions

Based on the objectives described in the introduction, the following research questions have been derived, which form the basis for the literature review:

- ***RQ1***: How can the current research on software architecture design be classified?
- ***RQ2***: What is the current state of the art of software architecture design research with respect to this classification? And the SPLA design, Quality-driven SPLA design, Safety-driven SPLA design methods in the existing methods?
- ***RQ3***: What can be learned from the current research results that will lead to topics for further investigation?

2.3.1.2 Research Tasks

To answer the three research questions RQ1-3, numbers of tasks have been conducted: one task to set up the literature review, and others research tasks dedicated to the identified research questions.

2.3.1.3 Literature Search Process

The search strategy for the review was primarily directed toward finding published papers in journals and conference proceedings via the widely accepted literature search engines and databases Google Scholar, IEEE Explore, and Elsevier ScienceDirect. For the search we focused on selected keywords, based on the aimed scope of the literature review. Examples of the keywords are: Software Architecture, Quality attributes, Safety analysis, Architectural Design, Software Product Line Architectures, Safety-driven software product line architecture design.

The keywords were refined and extended during the search process. In the subsequent phase, we reviewed the abstracts (and keywords) of the collected papers with respect to the defined set of inclusion and exclusion criteria (Look the lines below), and further extended the collection with additional papers based on an analysis of the cited papers and the ones citing it (forward and backward citation search).

- ***Inclusion Criteria:***

The focus of this literature review is on software architecture quality attributes, safety analysis, architectural design, software product line architectures, and safety-driven software product line architecture design. A summary of the inclusion and criteria is: Peer reviewed publications with a clear focus on some aspect of software product line architecture design.

- ***Exclusion Criteria:***

We excluded papers that: (a) design a software with no relation to software architecture, (b) focus on an architecture-irrelevant problem, (c) focus on software architecture design for single program without considering any quality attribute, (d) focus on a product line-irrelevant problem.

We did not exclude papers for quality reasons, because the quality of the papers was generally acceptable. A summary of the exclusion criteria is: Publications where either architecture design focus or software product line focus is lacking.

2.3.1.4 Threats to Validity

One of the main threats to the validity of this literature review is the incompleteness. The risk of this threat highly depends on the selected list of keywords and the limitations of the employed search engines. To decrease the risk of an incomplete keyword list, we have used an iterative approach to keyword-list construction. A well-known set of papers was used to build the initial taxonomy which evolved over time. New keywords were added when the keyword list was not able to find the state-of-the-art in the respective area of study. Another important issue is whether our taxonomy is robust enough for the analysis and classification of the papers. To avoid the taxonomy with insufficient capability to classify the selected papers, we used an iterative content analysis method to continuously evolve the taxonomy for every new concept encountered in the papers. New concepts were introduced into the taxonomy and changes were made in the related taxonomy categories. Furthermore, in order to make the taxonomy a better foundation for analyzing the selected papers, we allowed multiple abstraction levels for selected taxonomy concepts.

2.3.2 Taxonomy and Classification

The quality of a literature review project highly depends on the selected taxonomy scheme, which influences the depth of knowledge recorded about each studied approach (Aleti *et al.*, 2012).This section, presents the identification of the taxonomy categories and provides an answer to the first research question (RQ1).

In this survey we provided a basic classification framework in form of taxonomy to classify existing architecture design approaches. As mentioned in the previous sections, here we present two schemes of broad classifications of software architecture design approaches, the two sections bellow are illustrate that.

2.3.2.1 First classification scheme

As the first step of our survey, we classify existing approaches for software architecture design into three broad categories depending on whether they attempt to

address the architecture of single product or product line or quality attributes of architecture.

Each of these categories contains one or more subcategories based on the high-level strategies used to realize its goal. Some of these sub-categories are further divided indicating the specific intention adopted. Fig. 2.2 and Fig. 2.3 illustrate this classification framework through which the results of the survey are presented.

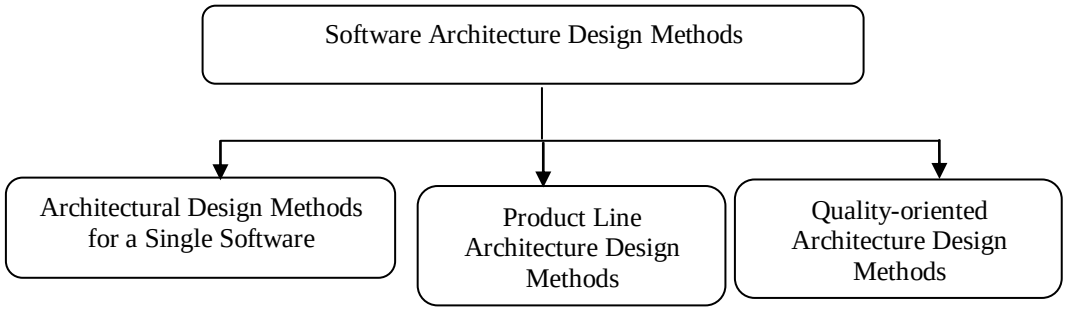


Figure 2.2: High level taxonomy of architecture design approaches

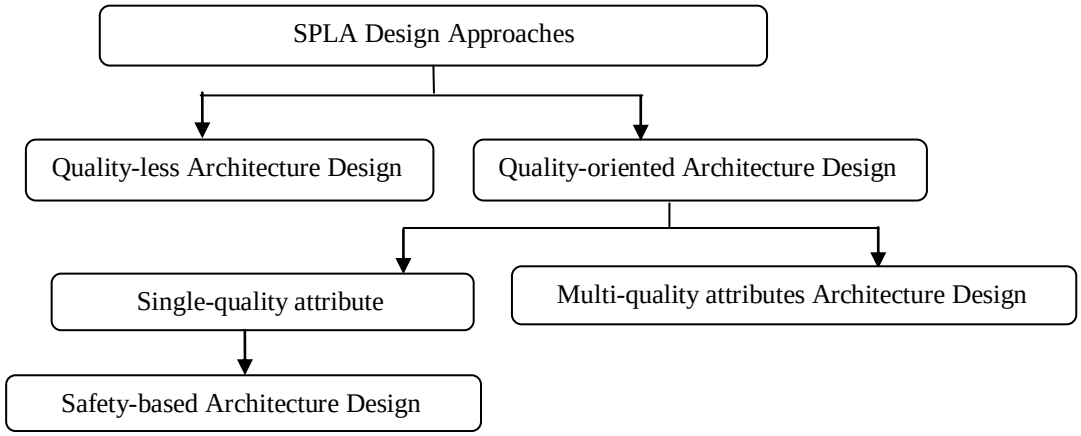


Figure 2.3: The taxonomy related to SPLA design approaches

The papers are published between 1998 and 2014, and summarized in Table 2.1, Tables 2.2 and 2.3. The total number of classification items in Table 2.1 is 29.

Table 2.1 lists all papers on term of architecture design approaches. Table 2.2 lists all papers on quality-oriented architecture design approaches. Table 2.3 lists all papers on software product line architecture design approaches.

2.3.2.2 The second classification scheme in term of SPLA design

Here, the publications are classified into categories in three different dimensions: research focus, type of contribution and research type, Table 2.4. This structure is presented by Petersen et al. (Petersen *et al.*, 2008), (Engström and Runeson, 2011). However we adopt different categories in our study. We established a scheme and mapped publications iteratively and added them as new primary studies. When the scheme was finally set, we reviewed all classifications again.

We identified Three categories of research focus: (i) SPLA design, (ii) quality-based SPLA design, (iii) safety-based SPLA design, Contribution type is classified into five categories: Tool, Method, Model, Metric, and Open Items (see Tables 2.5-2.7).

The classification of research types is based on a scheme proposed in (Engström and Runeson, 2011) (Wieringa *et al.*, 2006). And the research has been classified into six categories: (i) validation research, (ii) evaluation research, (iii) solution proposals, (iv) conceptual proposals, (v) opinion papers, and (vi) experience papers (see Tables 2.5-2.7).

Table 2.1: Papers on architecture design approaches (high level taxonomy/a broad classification)

No	Authors [Ref]	Paper Title	year	Architectural Design for a Single Software	PLA Design	Quality-oriented Architecture Design
1	David C et al. (Jensen and Tumer, 2013)	Modeling and Analysis of Safety in Early Design	2013	✓		✓
2	Lei Tan et al. (L Tan, Lin and Ye, 2012)	Modeling Quality Attributes in Software Product Line Architecture	2012		✓	✓
3	Yuling Huang (Huang, 2013)	Safety-Oriented Software Architecture Design Approach	2013	✓		✓
4	Len Bass et al. (Bass, Klein and Bachmann, 2001)	Quality Attribute Design Primitives and the Attribute Driven Design Method	2001	✓	✓	✓
5	Made Murwantara Tangerang, Indonesia (Murwantara, 2012)	Hybrid ANP: Quality Attributes Decision Modeling of a Product Line Architecture Design	2012		✓	✓
6	Qian Feng, Robyn R. Lutz (Feng and Lutz, 2005)	Bi-Directional Safety Analysis of Product Lines	2005		✓	✓
7	Joachim Bayer, Oliver Flege, and Cristina Gacek (Bayer, Flege and Gacek, 2000)	Creating Product Line Architectures	2000		✓	
8	Lei Tan, Yuqing Lin, Huilin Ye (Lei Tan, Lin and Ye,	Quality-Oriented Software Product Line	2012		✓	✓

	2012)	Architecture Design				
9	Weihang Wu, Tim Kelly (Wu and Kelly, 2004)	Safety Tactics for Software Architecture Design	2004	✓		✓
10	Liliana Dobrica, EILA Niemela (Liliana Dobrica, 2000)	Attribute-based product-line architecture development for embedded systems	2003		✓	✓
11	Bass, L.; Clements, P.; & Kazman, R. (Len Bass and Paul Clements and Rick Kazman, 2003)	Software Architecture in Practice. Reading , Attribute Driven Design method (ADD)	2003	✓		✓
12	John Ryan O'Farrell (John Ryan O'Farrell, 2009)	Development of A Software Architecture Method for Software Product Families and its Application to the AubieSat Satellite Program	2009		✓	
13	J¨urgen Meister, Ralf Reussner, Martin Rohde (Meister, Reussner and Rohde, 2004)	Applying Patterns to Develop a Product Line Architecture for Statistical Analysis Software	2004		✓	
14	P. America et al. (Obbink *et al.*, 2000)	COPA: A Component-Oriented Platform Architecting Method for Families of Software Intensive Electronic Products	2000		✓	
15	D. Weiss et al. (Weiss and Lai, 1999)	a family-based software development process.	1999		✓	

16	K. C. Kang et al. (Kang *et al.*, 1998)	FORM: A Feature-Oriented Reuse Method with Domain- Specific Reference Architectures	1998		✓	
17	C. Atkinson et al (Atkinson and Muthig, 2002)	Component-based product line engineering with UML	2002		✓	
18	Mikael Svahnberg et al. (Svahnberg *et al.*, 2003)	A Quality-Driven Decision-Support Method for Identifying Software Architecture Candidates	2003	✓		✓
19	M. Matinlassi et al. (Mari Matinlassi and Eila Nieme and Liliana Dobrica, 2002)	Quality-driven architecture design and quality analysis method	2002		✓	✓
20	F. Bachmann et al. (Bachmann *et al.*, 2000)	The Architecture Based Design Method	2000	✓	✓	
21	Hassan Gomaa (Gomaa, 2004)	Designing Software Product Lines with UML 2.0: From Use Cases to Pattern-Based Software Architectures	2006		✓	
22	Jianli Dong et al. (Dong *et al.*, 2008)	The Research of Software Product Line Engineering Process and Its Integrated Development Environment Model	2008		✓	
23	Jiayi Zhu et al. (Zhu *et al.*, 2011)	Improving Product Line Architecture Design and Customization by Raising the Level of Variability Modeling	2011		✓	
24	M.Sharafi, S.Dadollahi (M.Sharafi, 2013)	A Scenario-Based Approach for Architecture Reconstruction of Product Line	2013		✓	

25	Hataichanok Unphon (Unphon, 2009)	Making Use of Architecture throughout the Software Life Cycle – How the Build Hierarchy can Facilitate Product Line Development	2009		✓	
26	Jing Liu, Josh Dehlinger, Robyn Lutz (Liu, Dehlinger and Lutz, 2007)	Safety analysis of software product lines using state-based modeling	2007		✓	✓
27	Jan Bosch (Jan Bosch, 2001)	Software Product Lines and Software Architecture Design	2001		✓	✓
28	Thelma Elita Colanzi, Silvia Regina Vergilio (Colanzi and Vergilio, 2013)	Representation of Software Product Line Architectures for Search-Based Design	2013		✓	
29	Broerse, C et al. (Pinzger *et al.*, 2004)	Architecture Recovery for Product Family	2004		✓	

Table 2.2: Papers on quality-oriented architecture design approaches

No	Authors [Ref]	Paper	year	Single/Product line
1	M. Matinlassi, E. Niemel, and L. Dobrica (Mari Matinlassi and Eila Nieme and Liliana Dobrica, 2002)	Quality-driven architecture design and quality analysis method	2002	Product line
2	L. Bass, et al. (L. Bass, M. Klein, 2002)	Quality Attribute Primitives and the Attribute Driven Design Method	2002	Support all
3	David C. Jensen, Irem Y. Tumerb (Jensen and Tumer, 2013)	Modeling and Analysis of Safety in Early Design	2013	Single
4	Lei Tan, Yuqing Lin and Huilin Ye (L Tan, Lin and Ye, 2012)	Modeling Quality Attributes in Software Product Line Architecture	2012	Product line
5	Yuling Huang (Huang, 2013)	Safety-Oriented Software Architecture Design Approach	2013	Single
6	Len Bass, Mark Klein, and Felix Bachmann (Bass, Klein and Bachmann, 2001)	Quality Attribute Design Primitives and the Attribute Driven Design Method	2001	Single
7	Made Murwantara Tangerang, Indonesia (Murwantara, 2012)	Hybrid ANP: Quality Attributes Decision Modeling of a Product Line Architecture Design	2012	Product line
8	Qian Feng, Robyn R. Lutz (Feng and Lutz, 2005)	Bi-Directional Safety Analysis of Product Lines	2005	Product line
9	Lei Tan, Yuqing Lin, Huilin Ye (Lei Tan,	Quality-Oriented Software Product Line Architecture Design	2012	Product line

	Lin and Ye, 2012)			
10	Weihang Wu, Tim Kelly (Wu and Kelly, 2004)	Safety Tactics for Software Architecture Design	2004	Single
11	LILIANA DOBRICA, EILA NIEMELÄ (Liliana Dobrica, 2000)	Attribute-based product-line architecture development for embedded systems	2003	Product line
12	Mikael Svahnberg, Claes Wohlin, Lars Lundberg, Michael Mattsson (Svahnberg *et al.*, 2003)	A Quality-Driven Decision-Support Method for Identifying Software Architecture Candidates	2003	Single
13	Jing Liu, Josh Dehlinger, Robyn Lutz (Liu, Dehlinger and Lutz, 2007)	Safety analysis of software product lines using state-based modeling	2007	Product line
14	Jan Bosch (Jan Bosch, 2001)	Software Product Lines and Software Architecture Design	2001	Product line

Table 2.3: Papers on software product line architecture design approaches

No	Authors [Ref]	Paper Title	Date	Quality-less Architecture Design	Quality-oriented Architecture Design	Single-quality attribute	Multi-quality attributes Architecture Design
1	P. America et al. (Obbink *et al.*, 2000)	COPA: A Component-Oriented Platform Architecting Method for Families of Software Intensive Electronic Products	2000	✓			
2	D. Weiss, C. Lai, and R. Tau (Weiss and Lai, 1999)	Software product-line engineering: a family-based software development process.	1999	✓			
3	K. C. Kang et al. (Kang *et al.*, 1998)	FORM: A Feature-Oriented Reuse Method with Domain- Specific Reference Architectures	1998	✓			
4	C. Atkinson et al. (Atkinson and Muthig, 2002)	Component-based product line engineering with UML	2002	✓			
5	Lei Tan et al. (L Tan, Lin and Ye, 2012)	Modeling Quality Attributes in Software Product Line Architecture	2012		✓		✓
6	Len Bass, Mark Klein, and Felix Bachmann (Bass, Klein and Bachmann, 2001)	Quality Attribute Design Primitives and the Attribute Driven Design Method	2001		✓		✓
7	Made Murwantara Tangerang, Indonesia (Murwantara, 2012)	Hybrid ANP: Quality Attributes Decision Modeling of a Product Line Architecture Design	2012		✓		✓
8	Qian Feng, Robyn R. Lutz	Bi-Directional Safety Analysis of	2005		✓	✓	

	(Feng and Lutz, 2005)	Product Lines					
9	Joachim Bayer et al. (Bayer, Flege and Gacek, 2000)	Creating Product Line Architectures	2000	✓			
10	Lei Tan, Yuqing Lin, Huilin Ye (Lei Tan, Lin and Ye, 2012)	Quality-Oriented Software Product Line Architecture Design	2012		✓		✓
11	Liliana Dobrica, Eila NIEMELÄ (Liliana Dobrica, 2000)	Attribute-based product-line architecture development for embedded systems	2003		✓		✓
12	John Ryan O'Farrell (John Ryan O'Farrell, 2009)	Development of A Software Architecture Method for Software Product Families and its Application to the AubieSat Satellite Program	2009	✓			
13	J¨urgen et al. (Meister, Reussner and Rohde, 2004)	Applying Patterns to Develop a Product Line Architecture for Statistical Analysis Software	2004	✓			
14	Hassan Gomaa (Gomaa, 2004)	Designing Software Product Lines with UML 2.0: From Use Cases to Pattern-Based Software Architectures	2006	✓			
15	Jianli Dong et al. (Dong *et al.*, 2008)	The Research of Software Product Line Engineering Process and Its Integrated Development Environment Model	2008	✓			
16	Jiayi Zhu et al. (Zhu *et al.*, 2011)	Improving Product Line Architecture Design and Customization by Raising the Level of Variability Modeling	2011	✓			
17	M.Sharafi, S.Dadollahi	A Scenario-Based Approach for	2013	✓			

	(M.Sharafi, 2013)	Architecture Reconstruction of Product Line					
18	Hataichanok Unphon (Unphon, 2009)	Making Use of Architecture throughout the Software Life Cycle – How the Build Hierarchy can Facilitate Product Line Development	2009	✓			
19	M. Matinlassi, E. Niemel, and L. Dobrica (Mari Matinlassi and Eila Nieme and Liliana Dobrica, 2002)	Quality-driven architecture design and quality analysis method	2002		✓		✓
20	F. Bachmann et al. (Bachmann *et al.*, 2000)	The Architecture Based Design Method	2000	✓			
21	Jing Liu, Josh Dehlinger, Robyn Lutz (Liu, Dehlinger and Lutz, 2007)	Safety analysis of software product lines using state-based modeling	2007		✓	✓	
22	Jan Bosch (Jan Bosch, 2001)	Software Product Lines and Software Architecture Design	2001		✓		✓
23	Thelma Elita Colanzi, Silvia Regina Vergilio (Colanzi and Vergilio, 2013)	Representation of Software Product Line Architectures for Search-Based Design	2013	✓			
24	Broerse, C et al. (Pinzger *et al.*, 2004)	Architecture Recovery for Product Families	2004	✓			

Table 2.4: Distribution over research focus

Research focus	1998-2008	2009-2014	Total
SPLA design	10	5	15
Quality-based SPLA design	4	3	7
Safety-based SPLA design	2	-	2
Total	16	8	24

Table 2.5: Papers on SPLA design

Authors [Ref]	Title	Paper Type	Contribution type
John Ryan O'Farrell (John Ryan O'Farrell, 2009)	Development of A Software Architecture Method for Software Product Families and its Application to the AubieSat Satellite Program	Conceptual proposal	Method
Joachim Bayer et al. (Bayer, Flege and Gacek, 2000)	Creating Product Line Architectures	Solution proposal	Method
J¨urgen Meister, Ralf Reussner, Martin Rohde (Meister, Reussner and Rohde, 2004)	Applying Patterns to Develop a Product Line Architecture for Statistical Analysis Software	Experience report	Tool
Hassan Gomaa (Gomaa, 2004)	Designing Software Product Lines with UML 2.0: From Use Cases to Pattern-Based Software Architectures	Solution proposal	Model
Jianli Dong et al. (Dong *et al.*, 2008)	The Research of Software Product Line Engineering Process and Its Integrated Development Environment Model	Opinion paper	Model
Jiayi Zhu et al. (Zhu *et al.*, 2011)	Improving Product Line Architecture Design and Customization by Raising the Level of Variability Modeling	Conceptual proposal	Open items

M.Sharafi, S.Dadollahi (M.Sharafi, 2013)	A Scenario-Based Approach for Architecture Reconstruction of Product Line	Conceptual proposal	Method
Hataichanok Unphon (Unphon, 2009)	Making Use of Architecture throughout the Software Life Cycle – How the Build Hierarchy can Facilitate Product Line Development	Opinion paper	Open items
P. America et al. (Obbink *et al.*, 2000)	COPA: A Component-Oriented Platform Architecting Method for Families of Software Intensive Electronic Products	Conceptual proposal	Model
D. Weiss, C. Lai, and R. Tau (Weiss and Lai, 1999)	Software product-line engineering: a family-based software development process.	Conceptual proposal	Open items
K. C. Kang, et al. (Kang *et al.*, 1998)	FORM: A Feature-Oriented Reuse Method with Domain- Specific Reference Architectures	Conceptual proposal	Model
C. Atkinson et al. (Atkinson and Muthig, 2002)	Component-based product line engineering with UML	Conceptual proposal	Tool
F. Bachmann et al. (Bachmann *et al.*, 2000)	The Architecture Based Design Method	Solution proposal	Model
Thelma Elita Colanzi, Silvia Regina Vergilio (Colanzi and Vergilio, 2013)	Representation of Software Product Line Architectures for Search-Based Design	Experience report	Open items
Broerse, C et al. (Pinzger *et al.*, 2004)	Architecture Recovery for Product Family	Solution proposal	Method

Table 2.6: Papers on Quality-based SPLA design

Authors [Ref]	Title	Paper Type	Contribution Type
Lei Tan, Yuqing Lin and Huilin Ye (L Tan, Lin and Ye, 2012)	Modeling Quality Attributes in Software Product Line Architecture	Conceptual proposal	Method
I Made Murwantara Tangerang, Indonesia (Murwantara, 2012)	Hybrid ANP: Quality Attributes Decision Modeling of a Product Line Architecture Design	Conceptual proposal	Model
Lei Tan, Yuqing Lin, Huilin Ye (Lei Tan, Lin and Ye, 2012)	Quality-Oriented Software Product Line Architecture Design	Solution proposal	Method
Liliana Dobrica, Eila Niemela (Liliana Dobrica, 2000)	Attribute-based Product-line Architecture Development for Embedded Systems	Solution proposal	Method
Len Bass, Mark Klein, and Felix Bachmann (Bass, Klein and Bachmann, 2001)	Quality Attribute Design Primitives and the Attribute Driven Design Method	Opinion paper	Open items
M. Matinlassi, E. Niemel, and L. Dobrica (Mari Matinlassi and Eila Nieme and Liliana Dobrica, 2002)	Quality-driven architecture design and quality analysis method	Conceptual proposal	Method
Jan Bosch (Jan Bosch, 2001)	Software Product Lines and Software Architecture Design	Experience report	Method

Table 2.7: Papers on Safety-based SPLA design

Authors [Ref]	Title	Paper Type	Contribution type
Qian Feng, Robyn R. Lutz (Feng and Lutz, 2005)	Bi-Directional Safety Analysis of Product Lines	Conceptual proposal	Method
Jing Liu, Josh Dehlinger, Robyn Lutz (Liu, Dehlinger and Lutz, 2007)	Safety analysis of software product lines using state-based modeling	Solution proposal	Method

2.3.3 Mapping

In Fig. 2.4, we show, based on the second classification in section 2.3.2.2, a mapping between the research focus and the contribution type and the research type. The research focus items include SPLA design, Quality-based SPLA design, and Safety-based SPLA design. The contributions types include tools, methods, models, and metrics. The research types include experience reports, opinion papers, conceptual or solution proposals, and validation and evaluation research. The Research focus is on the Y axis, the contribution type is on the left side of the X axis, and research type on the right side of the X axis.

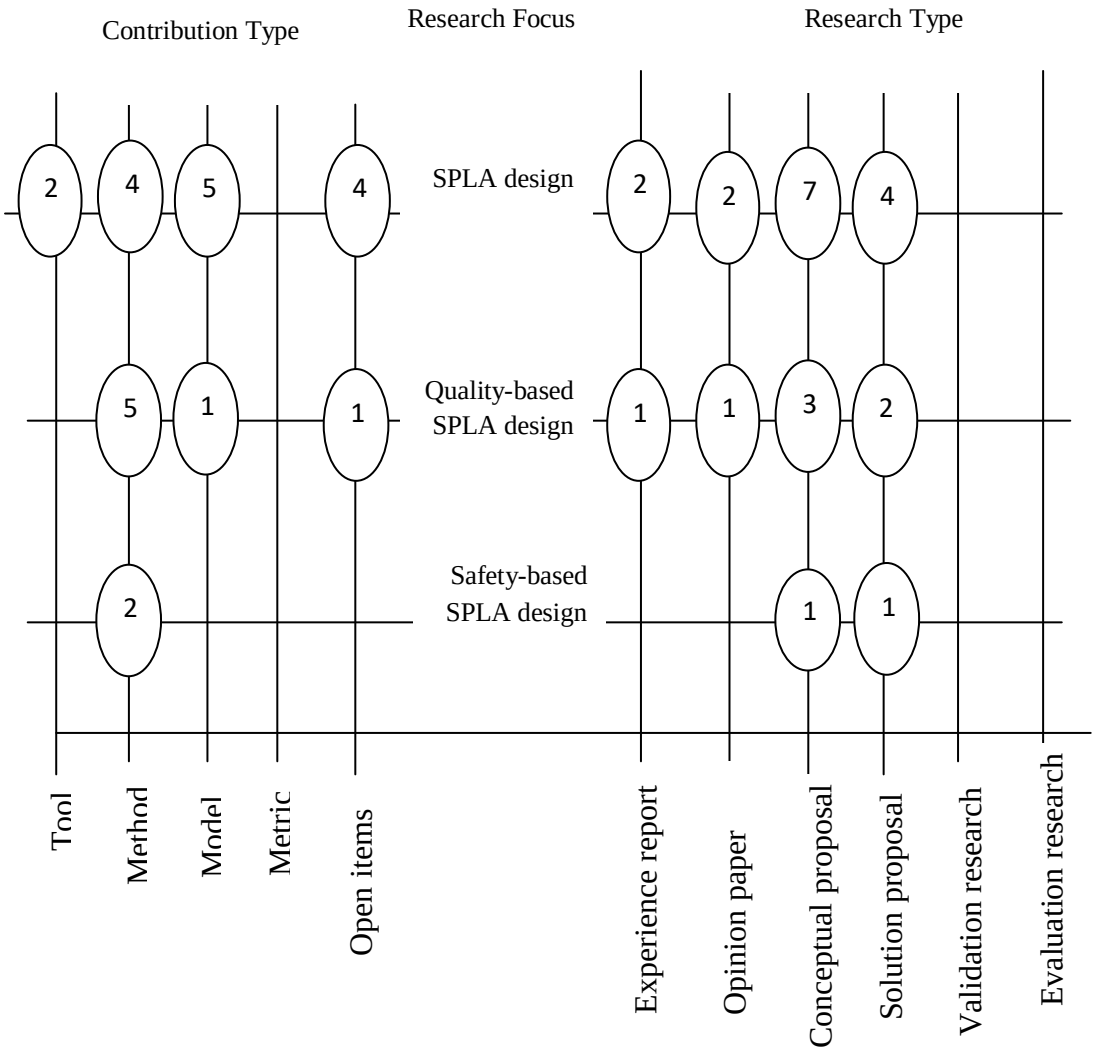


Figure 2.4: Map of research focus on software product line design. Research focus on the Y axis; contribution type on the left side of the X axis, and research type on the right side of the X axis.

2.3.4 Survey Results and Discussion

The surveyed research work indicates safety based software product line design being rather an immature area.

Safety-driven software product line architecture design seems to be a ‘‘discussion” topic. There is a well-established understanding about challenges. However, when looking for solutions to these challenges, we mostly find proposals. The mapping shows that 74% of the papers found include proposals, which contain ideas for solutions of the identified challenges.

The study shows that there are various design methods available and each is focusing on certain perspective of architecture design and there is a large number of SPLA design methods. Especially, quality-based methods have received a lot of attention. However, the use of safety-based design methods is limited in software product lines (SPL) due to the variability property that can potentially result in a large number of possible systems.

2.4 Architectural design patterns

The design patterns concepts are important in the systems design process. They are used to support and help designers and system architects choose a suitable solution for a recurring design problem among available collection of successful solutions.

Developing a reference architecture which represents the base structure of the member products is the main task of the software product line architecture design (Systems, 2000). It is evidence that a pattern based development of the reference architecture can support the development and application process of product lines.

2.4.1 Safety Design Pattern and Statechart Pattern

Design patterns can be used to enhance the design of systems in different application domains. Design patterns are popular in the field of software engineering and there are plenty of patterns (Rauhamäki, Vepsäläinen and Kuikka, 2012). Contradictorily, in the field of control, safety and security engineering patterns have

not been studied and published in such volumes. Therefore, one of the objectives of the work in this thesis is to answer the call, for more details see Chapter 4 and 5.

In the other hand, and because it is very efficiency to address the safety of the system considering the changing of the system from a state to another state, so we can monitor and control the safety of the system in each state. The challenge is how to combine the concepts of the traditional safety patterns with the concepts of statechart patterns.

2.4.2 Safety and Security Patterns

In (Amorim *et al.*, 2017) they argue that there is lack of experience with security concerns in context of safety engineering in general and in automotive safety departments in particular. To remediate this problem, they propose a pattern-based approach that provides guidance with respect to selection and combination of both types of patterns in context of system engineering. However this work focuses on the systems development engineering specifically, for developing a safety-critical systems with respect to the influence of security issues and not for patterns development. Therefore, extra work is needed to use the experience at the pattern design level which adds more generality and spread the using of the experience.

Ensuring safety and security of complex integrated systems requires coordinated approaches that involve different stakeholder groups going beyond safety and security experts and system developers. The authors in (Raspotnig, Karpati and Opdahl, 2018) have therefore proposed CHASSIS (Combined Harm Assessment of Safety and Security for Information Systems), a method for collaborative determination of requirements for safe and secure systems.

In the context of the statechart patterns, especially the safety patterns of statechart, it evidence that there is no explicit consideration of the influence of the security on the safety which is the most important thing. Therefore, in this work and in order to enhance the existing safety patterns or even develop new ones, to address the influence of the security issues on the safety, a pattern development approach that interlinks safety and security patterns is proposed.

To the best of our knowledge, there are various pattern approaches in the context of patterns composition, specifically at the architectural design level. In comparison

to our work, none of the aforementioned approaches show clearly the pattern composition process to develop the patterns. Pattern composition is considered as a challenge. In the other words, these approaches are limited at patterns development level, especially, for safety and security patterns development because this area still considered as an emergent research area.

2.5 Chapter Summary and Open Research Issues

Because a product line reference software architecture is the central artifact in product line engineering which provides the framework for developing and integrating shared assets [47], the creation and validation of such architecture are inherently much more complex. The study presented in this chapter aims at surveying existing research on Software Product Line Architecture (SPLA) based on quality attributes in order to identify useful approaches and needs for future research. We investigated safety analysis at the architectural level, and Safety-driven Software Product Line Architecture SSPLA design approaches.

The chapter shows that there are various design methods available and each is focusing on certain perspective of architecture design. The quality-based methods have received a lot of attentions and have been well developed for single system architecture design. However, the use of quality-based design methods is limited in software product line (SPL) because of the complexity and variability existing in the SPL reference architecture [22]. With the increasing attention to software safety, improving software safety has already become a much more important issue, especially for safety-critical systems [26]. We identify the following open research issues as related to specific key publications surveyed in this chapter:

- The methodology presented by Jensen and Tumer [25] focuses on the inclusion of safety into design level in general without focusing on a specific design activity such as the design of reference architectures.
- In [22], Lei Tan et al, presented a framework for Quality-Oriented Software Product Line Architecture Design. The framework is defined at a high level without specifying any modeling techniques or tools.
- The work of Huang in [26] provides a rather complex process for safety-oriented design using Fault Tree Analysis techniques and safety tactics

that would be difficult or intractable to use with the concept of variability in product lines where the space of possible faults is very large. It remains an open issue to research if the FTA techniques combined with safety tactics can be used with fault classification techniques for safety-driven design of reference architectures.

- In the context of the safety patterns, especially the safety patterns of statechart, it evidence that there is no explicit consideration of the influence of the security on the safety which is the most important thing. Therefore, this point needs more work to address the influence of the security issues on the safety.

The last point is still considered as an emergent research area.

CHAPTER III

RESEARCH METHODOLOGY

3.1 Introduction

This chapter describes the methodology according to which this research was completed. The chapter presents the methodology in terms of its phases, the languages, the notations and tool-support used for the development of the context. A brief description about the case studies conducted in this thesis is also presented in this chapter.

3.2 Research Methodology

This section describes the research method applied in this thesis. A number of conceptual research design frameworks were considered to put the methodology in context (e.g. the description of a constructive research process by Kasanen *et al*. (Kasanen, Lukka and Siitonen, 1993), building and evaluating system techniques and methods iteratively and incrementally based on cases by (Hevner et al. 2004) (Hevner *et al.*, 2004) and the software engineering paradigms identified by Mary Shaw (2001)) (Shaw, 2003). In this thesis the overall research design is basically based on the conceptual research design framework described by Robson in (Robson, 2002) and guidelines by Shaw (Shaw, 2001) (Shaw, 2003). The most important elements of the research design are depicted in Figure 1 and described in the following lines.

The phases of the research process are realized as follows:

1. Define the research problem:

The research problem and its significant were described in Chapter 1.

2. Understand the context of the problem.

To obtain a good understanding about the problem the previous results related to the research problem were searched for. Literature studies were needed to obtain a sufficient level of understanding. An amount of relevant literature was found, studied, and analysed. The analysis results were reported in a number of publications (Mozamil Elgodbe and Ammar, 2016), (Mozamil Ebnauf and Hany H. Ammar, 2017).

3. Innovate, i.e., construct a solution idea.

This is discussed in Chapter 4, 5 and 6

4. Demonstrate that the solution works, i.e., their applicability.

Hence in Shaw's terminology (Shaw, 2001), the form of validation is *implementation*. We motivate our work with the help of two application examples of two case studies. In Chapter 8 and in order to show the applicability of this work the developed safety and security design pattern is applied on the design process of two software product line architectures, a simplified Automated Electromechanical Braking Systems product line and the Smart Microwave Oven Control Systems Software Product Line.

To evaluate our work, a simplified safety assessment model (Chapter 7) is developed and applied on the two case studies.

5. Show theoretical connections and research contribution

The contribution of this thesis is present through its chapters (mainly, Chapter 4, 5, 6 and 7). The contribution is also summarized in the last chapter (Chapter 9).

The final step of our evaluation process is to compute the Relative Safety Improvement. Section 8.4.3 describes how all these works can improve the safety design of the software product line architectures. The section also shows the effectiveness of our work.

6. Examine the scope of applicability.

According to the evaluation results discussed in the fourth point, the applicability of the methods is discussed in Chapter 9.

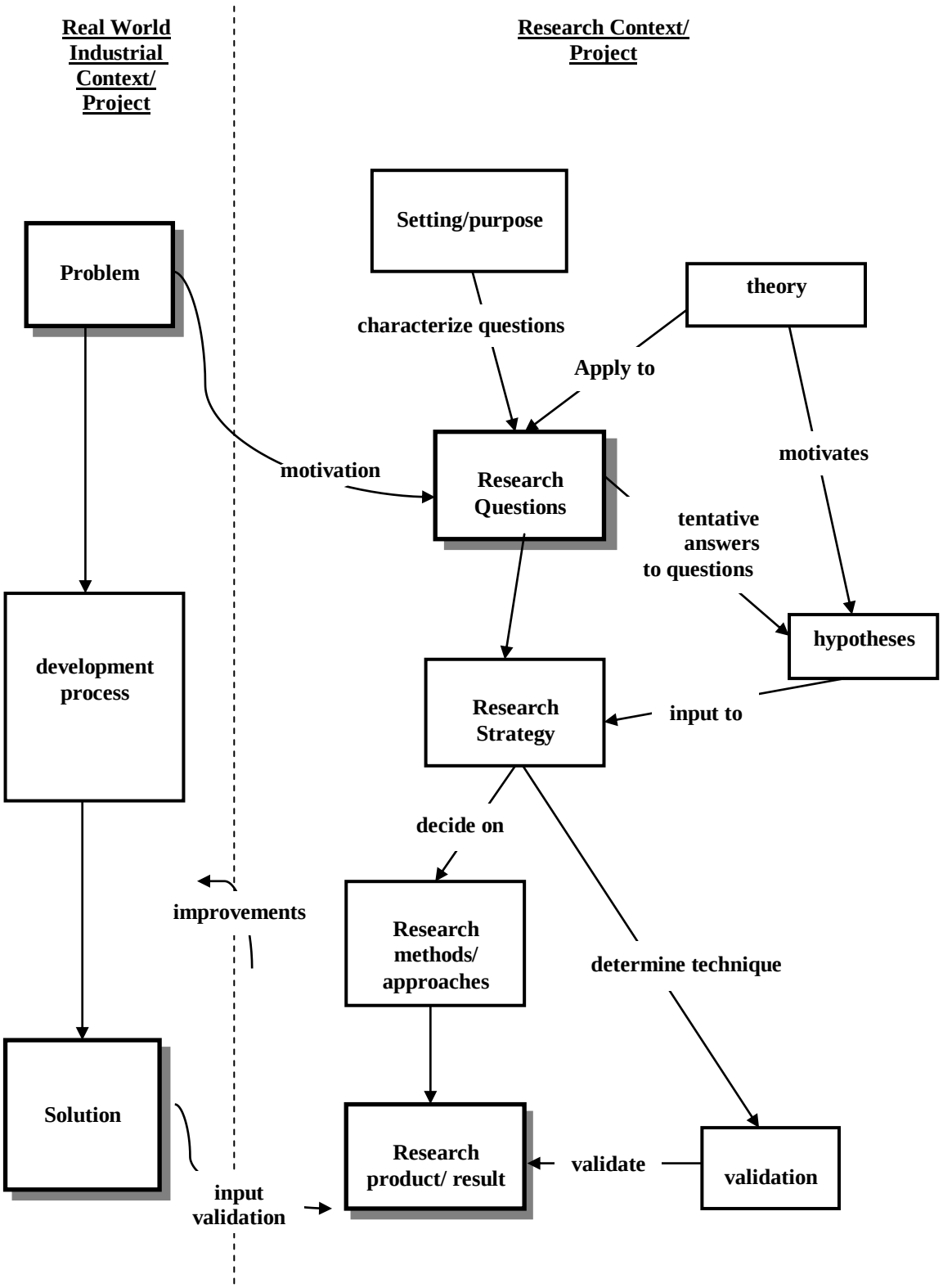


Figure 3.1: Elements of design of a "real world" research project (based on (Mustapi′c, 2004)(Robson, 2002))

3.3 Languages, Notations and Used Tools

To support the process of completing the research some languages and tools were needed.

In the other words, we need to add the appropriate support, in the development environment, e.g. tools, languages. The first required languages are modeling languages. The important language used in this thesis is the Unified Modeling Language (UML). Also the second required tool is a software capable of create and edit UML models.

Another example of such a tool is a tool that used in the evaluation process to analyze and present the results. Finally, we also need academic and scientific writing tools.

This subsection describes the UML language, notations and some used tools.

3.3.1 Language and Notations

3.3.1.1 UML Language

”According to the Object Modeling Group (OMG), “modeling is the designing of software applications before coding.” (Gomaa, 2011).

With the model-based software design and development, software modeling is used as an essential part of the software development process. The models can help understanding of the systems which specify the system from different perspective. The Unified Modeling Language (UML) is one of the graphical modeling languages. The UML helps in developing, understanding, and communicating the different views (Gomaa, 2011).

The Object Management Group (OMG) governs the UML, and Sparx Systems is an active member and contributor to the process of managing and improving the language (Sparx Systems, 2016).

The Unified Modeling Language (UML) highly supports the object-oriented modeling. It was developed to provide a standardized graphical language and notation for describing object-oriented models (Gomaa, 2011).

In context of the product line engineering the UML efficiently addresses the variability existing in the product line. For example: In UML, the use cases are labeled with the stereotype «kernel», «optional» or «alternative» [23]. In addition, variability can be inserted into a use case through variation points, which specify locations in the use case where variability can be introduced (Gomaa, 2004).

3.3.1.2 The Statechart Semantic

In the UML notation, a state transition diagram is referred to as a state machine diagram. The UML state machine diagram notation is based on Harel's statechart notation (Harel 1988; Harel and Politi 1998) (Harel, 1987)(Gomaa, 2011).

The UML provides two different kinds of state machine formalisms: statecharts and activity diagrams. They differ in the kinds of situations to which they are applied. Statecharts are used when the transition from state to state takes place primarily when an event of interest occurs. Activity diagrams are appropriate when the object (or operation) changes state primarily upon completion of the activities executed within the state rather than the asynchronous occurrence of events.

Statecharts, introduced by Harel in the late 1980s , have become a popular means for specifying the behavior of embedded, reactive systems (von Hanxleden *et al.*, 2014). The visual syntax of statecharts is intuitively understandable for application experts from different domains who are not necessarily computer scientists.

A statechart is an alternative means of system specification. This specification methodology is particularly oriented to "reactive systems" — that is, systems that respond to a series of events rather than transforming an input into an output (Niaz and Tanaka, 2003). Such systems may incorporate concurrent processing, and statecharts encompass this capability. The OO methodologies using statecharts describe in sufficient detail the steps to be followed for describing the behavior of objects (Niaz and Tanaka, 2003). In most OO methodologies, when people think of object behavior, they consider the functionality of the object. But in many real world applications this definition is insufficient — the internal state of an object and the quality attributes should also be considered (e.g. the safety in safety-critical systems). As (UML) statechart diagram is a powerful tool for specifying the dynamic behavior

of reactive objects, we can use this facility to describe the system behavior in term of safety.

Because it is very efficiency to address the safety of the system considering the changing of the system from a state to another state, so we can monitor and control the safety of the system in each state. Also the dynamic modeling – in the same time we can model the system with dynamic modeling. Many systems, such as real-time systems, are highly state-dependent; that is, their actions depend not only on their inputs but also on what has previously happened in the system.

We have already developed a new safety-driven design pattern, see Section 5.3. This design pattern support the design process in context of software architectures considering one of the software quality attribute which is safety attribute. This pattern extends capabilities of both the statecharts design patterns and safety patterns. The extension of the pattern is by addressing the security issues to develop an enhanced version of the pattern. The pattern addresses the safety attribute in the behavior of the objects to improve the safety of applications. The pattern allows an object to alter its behavior and change its internal state when there is a safety violation or even security violation that influence the safety, and to protect it from introducing in unsafe states. The result is an object-oriented design pattern which handles the safety attribute.

In the context of dynamic state machine modeling for software product lines - When variable classes are developed, there are two main approaches to consider, specialization or parameterization. In product line development, however, there can be a large degree of variability. Consider the issue of variability in state-dependent control classes, which are modeling using state machines and depicted on statecharts (Gomaa, 2011)(Gomaa, 2004). To capture state machine variability and evolution, it is necessary to specify optional states, events and state transitions, and actions. It is often more effective to design a parameterized state machine, in which there are feature-dependent states, events, and transitions. Optional state transitions are specified by having an event qualified by a Boolean feature condition, which guards entry into the state. Optional actions are also guarded by a Boolean feature condition, which is set to True if the feature is selected and False if the feature is not selected for a given SPL member See reference (Gomaa, 2011)(Gomaa, 2004).

3.4 Evaluation Notations, Metrics and Tools

3.4.1 The Relative Safety Improvement- a safety assessment metric

The final step of our evaluation process is to compute the Relative Safety Improvement (RSI). The Relative Safety Improvement (RSI) is a safety assessment metric proposed by Ashraf in (Armoush, 2010) which gives an indication about the safety improvement that can be achieved by the pattern. This metric (RSI) is defined as "the percentage improvement in safety (reduction in probability of unsafe failure) relative to the maximum possible improvement which can be achieved when the probability of unsafe failure is reduced to the minimum possible value (0)". Based on this definition, the relative safety improvement for a design pattern (or system design) can be expressed as shown in the following equation (Equation 1):

$$\left[RSI = \left(1 - \frac{P_{UF(new)}}{P_{UF(old)}}\right) \times 100\% \right] \qquad 1$$

- *RSI: Relative Safety Improvement.*
- *PUF(old): Probability of unsafe failure in the basic system.*
- *PUF(new): Probability of unsafe failure in the design pattern.*

We assume that the probability of the system to be in unsafe state is equivalent to the probability of the unsafe failure by considering that all failures are unsafe failures. By this assumption we do the calculation of the RSI for all the individual scenarios in the two examples of the two case studies used in this thesis, see Chapter 8 (Section 8.4.3, Fig. 8.21 and Fig. 8.22).

3.4.2 Markov Chain methods

As such domains include safety critical systems which exhibit probabilistic behavior, there is a major need for modeling and verification approaches dealing with probabilistic aspects of systems. In the other words, we need a probabilistic model

that captures statistical properties of system usage, and can make runtime probabilistic estimations.

State based formalisms are a more powerful alternative to combinatorial formalisms. Markov models and their underlying matrix algebra have been proposed as a means of evaluating usability at design-time (Kostakos *et al.*, 2016).

Markov chains (MCs) is one of the most common methods. They are effective tools that used for evaluating the safety and reliability of architectures (Varshosaz and Khosravi, 2013). After defining the Markov chains then they will be evaluated regarding the probabilities that the system is in a certain state at time *t*. As mentioned in Sec. 7.2.2.1 that in this thesis we used the steady state evaluation technique which calculates the probabilities for $t \rightarrow \infty$.

Because of the variabilities existing in the product line systems, the authors in (Varshosaz and Khosravi, 2013), introduce a mathematical model, Discrete Time Markov Chain Family (DTMCF), which compactly represents the probabilistic behavior of all the products in the product line. In (Dabrowski and Hunt, 2011) they describe how a Discrete Time Markov chain simulation and graph theory concepts can be used together to efficiently analyze behavior of complex distributed systems.

Note that, in the solution that used in this thesis, the calculation is based on some hypothesis and solutions of other works on Markov chain. We use the Discrete Time Markov Chain (DTMC) theory (Kassir, 2018) to efficiently analyze behavior and safety of the critical systems in term of software system architecture. The safety assessment method used in this work (See Sec. 7.2) is distinguishable from the well-known use of DTMCs to provide quantitative measures of system performance and reliability, which has reviewed in (Dabrowski and Hunt, 2011). Instead of measuring system reliability, we use DTMCs to examine safety in dynamic systems in order to identify the probability of system being in safe execution or in unsafe state.

3.5 Software Tools

3.5.1 UML Modeling Tools

UML is a Unified Modeling Language. UML is considered as an industry standard general-purpose modeling language for software engineering. It is used to

create meaningful, object-oriented models for a software application (International Organization for Standardization and International Electrotechnical Commission, 2005).

A UML tool is a software application that supports some or all of the notation and semantics associated with the Unified Modeling Language (UML). The UML tools are used broadly to include application programs which are not exclusively focused on UML, but which support some functions of the Unified Modeling Language, either as an add-on, as a component or as a part of their overall functionality (International Organization for Standardization and International Electrotechnical Commission, 2005). There are numerous tools available for designing UML diagrams. Some of these tools are commercial and the others are open-source.

Bellow is examples for these tools:

1. Magic Draw

Magic Draw is one of the most popular UML CASE tool. It is used to model UML diagrams, SysML, BPMN, and UPDM that supports the dynamic collaboration of the team. This tool is meant for business analysts, software analysts. It also facilitates analyzing and designing object-oriented systems and databases.

2. StarUML

StarUML is an open-source software modeling tool, which is provided by MKLab.

It is a sophisticated software modeler aimed to support agile and concise modeling.

It has come up with eleven different types of modeling diagrams. It also supports UML2.0 specified diagrams. If you use StarUML(tm), you can easily and quickly design exact software models which is based on UML standard. It will guarantee to maximize the productivity and quality because of generating numerous results automatically from it (Documentation.help, 2022).

The key features of StarUML are (javatpoint, 2022):

- It let you create Object, Use case, Deployment, Sequence, Collaboration, Activity, and Profile diagrams.

- It is a UML 2.x standard compliant.
- It offers multiplatform support (MacOS, Windows, and Linux).
https://www.javatpoint.com/uml-tools

3.5.2 Matlab

MATLAB (matrix laboratory) is a numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran. Matlab is a software package used primarily in the field of engineering for signal processing, numerical data analysis, modeling, programming, simulation, and computer graphic visualization (Dr. Ali Assi, 2011). It can be very useful when applied in the model-based design of dynamic systems, especially in the domain of power electronics systems (Dr. Ali Assi, 2011).

The authors in (Dr. Ali Assi, 2011) explain how MatLab/Simulink, with its toolboxes, is well adapted to solve the issues of defining the design requirements, at developing different components' models for the physical evolution processes and, through the combination of various sub-systems, how it enables engineers to verify that the overall performance satisfies the requirements.

The safety assessment model (Chapter 7) is interconnected with the system model and potential attack and failure (risk) scenarios are described through the models. We use the statechart models to describe the system design or the risk scenarios. And then we use these statechart models to define the Markov chain corresponding to each model (e.g. Fig. 8.15). These Markov models are then used in the mathematical calculations in the assessment process, see Sec. 8.4.

After defining the Markov chains then it can be evaluated regarding the probabilities that the system is in a certain state at time t. In this thesis we used the steady state evaluation technique which calculates the probabilities for $t \rightarrow \infty$. So the reliability and safety can be calculated by summing up the probabilities of the reliable respective safe states. One of the possible methods to do the analysis for both the

transient and the steady state is Monte Carlo (Dabrowski and Hunt, 2011). In this thesis we used the normal mathematical calculation of the steady state. So the results are not exact even with using Monte Carlo simulation since the accuracy depends on the number of simulation runs.

In this work we use Matlab in the following situations:

- **Perform the simulation processes**

Matlab can be used to perform the simulation processes that are partial step of the evaluation process. The evaluation process is one of the critical steps in our proposed design method for the software product line architectures (Chapter 4).

The critical point of the assessment solution proposed in this thesis is the method of defining or estimating the transition probabilities between the states in the Markov model. So we proposed a derived technique extracted from the previous works (Varshosaz and Khosravi, 2013) and (Dabrowski and Hunt, 2011) called (Maximum Likelihood Estimation for Markov Chains (MLE)). The following lines conclude the idea:

By using simulation technique, execute the Markov model for a period of time. And then we observe the execution to compute the number of transition between the states. State transition probabilities were derived as follows. Given states s_i, s_j, i,j= 1…n where n= the number of the system states, p_{ij}, is the probability of transitioning from state i to state j, written as $s_i \rightarrow s_j$. This probability is estimated by calculating the frequency of $s_i \rightarrow s_j$, or f_{ij}, and dividing by the sum of the frequencies of s_i to all other states s_k, as shown in equation (2), Chapter 7. Finally, we use the equation to calculate the transition probabilities on the Markov chain.

- ***Conducting the mathematical Calculations***

Conduct the calculation processes of the safety assessment process using mathematical methods upon the resulted Markov model. This step includes creation of the Transition Probability Matrix for each Markov model. When the transition probability matrix p is created, other Markov processes are used like steady state operation. Finally we calculate the probability of each state for the system states

before and after using the proposed design pattern. And finally we will observe the calculation results and then write the assessment results.

3.5.3 MS Excel

Among the computer programs which exist, Microsoft Excel is one of the most important because of the key role it plays in many sectors (Training.org, 2021).

Excel is a spreadsheet program from Microsoft and a component of its Office product group for business applications. Microsoft Excel enables users to format, organize and calculate data in a spreadsheet (TechTarget, 2021).

By organizing data using software like Excel, data analysts and other users can make information easier to view as data is added or changed (TechTarget, 2021). Microsoft Excel provides a grid interface to organize nearly any type of information. The power of Excel lies in its flexibility to define the layout and structure of the information you want to manage (Opengatesw.net, 2021).

Major uses for Excel are:

- to create budgets.
- to produce graphs and charts.
- Excel uses a large collection of cells formatted to organize and manipulate data and solve mathematical functions.
- Within business spreadsheet software is used to forecast future performance, calculate tax, completing basic payroll, producing charts and calculating revenues
- graphing, this package plays a very important role in graphing as it has the ability to produce a variety of different charts, which may be used to represent statistical data in more visual way.
- Predictions / Simulations
- Statistical analysis
- Explore and interpret data in order to draw conclusions for business

3.5.4 Academic and Scientific Writing Tools

3.5.4.1 Microsoft Word

One of the most widely used programs of Microsoft Office suite, MS Word is a word processor developed by Microsoft.

Microsoft Word is a power tool to create research and scientific documents. Many academics and scientists use Microsoft Word to write up their research and prepare it for publication in books, scholarly journals and online. Certainly the program is an excellent tool for authors (Tetzner, 2021).

3.5.4.2 References Management- Mendeley

Mendeley is a reference manager and academic social network that can help us organize our documents and references. It provides the free references manager for our publishing requirements.

Mendeley is free academic software that is available on all major platforms and in all modern browsers. This point means it can be used on Mac, PC, or in Linux. Mendeley offers us a desktop application that we run on our computer, a web library for when we are not at our own computer, and an iOS application, so (we can work on the go).

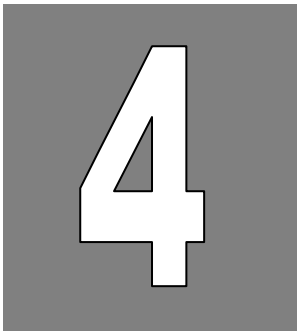
Mendeley helps us collaborate with our fellow researchers online by joining and working together in groups. It can provide us with readership statistics and recommendations. Using Mendeley allows you to collect, manage, store, share and use research papers and articles, as well as generate bibliographies in the citation style of your choice. It can be used with MS Word to add citations as you type as well as compile a reference list at the end of your assignment (Canterbury Christ Church University, 2019).

3.6 Brief Description of the Case Studies

We have motivated our work with the help of two case studies. These two case studies are presented to illustrate how all this work can improve the safety design of the SPLAs. In Chapter 8 and in order to show the applicability of our work the developed safety and security design pattern (Sec. 6.4) is applied on the design process of two software product line architectures, a simplified Automated Electromechanical Braking Systems product line and the Smart Microwave Oven Control Systems Software Product Line. Using our proposed safety and security pattern require developing a statechart to specify the entity's behavior without safety and security violation. Therefore, it is efficient to use a state-based architectural design approach in the overall SPLAs design lifecycle. For that the state-driven architecture design method for safety–critical software product lines (Sec. 4.4) is used in the architectural design process of these two SPLs. As we mentioned in the lines above that the two examples of the two case studies are presented to illustrate how this work can improve the safety design of the SPLAs.

Furthermore, to evaluate our work, a simplified safety assessment model proposed in Chapter 7 is applied on the running examples. The results show that by using our proposed design method and after using the proposed safety and security design pattern we can achieve a considerable improvement in the system safety design.

CHAPTER IV



ARCHITECTURAL DESIGN FOR SAFETY-CRITICAL SPLs

4.1 Introduction

Some of the content in this Chapter has been presented at 1st International Conference on Engineering and Applied Sciences, ICEAS2017,Red See, Sudan, and published at the Sea University Journal of Basic and Applied Science, 2(1) (Mozamil Ebnauf and Hany H. Ammar, 2017). And the contents of the two sections (Section 4.4 and 4.5 has been presented at 7th IEEE International Conference on Mechatronics Engineering, ICOM2019, Putrajaya, Malaysia (Ebnauf and Al., 2019).

This chapter introduces the proposed method for safety-driven software product line architecture design (SSPLA). This comprehensive method to the architectural design process of software product line systems is focusing on cyber-physical and embedded systems in safety-critical applications. The key aspect of this method is the use of the concept design patterns which improves the design process. In Chapter 5 we present our proposed safety-driven design pattern.

A modification to the traditional SPL architecture design method is proposed with the use of the statechart patterns and safety-based design concepts. In the other words, the method has been adapted to be a state-driven method. This adaptation means that most or all process steps should be based on or around the statechart semantic.

The remainder of this chapter is organized into four main sections. Section 4.2 describes some important topics related to the software architecture design. Section 4.2.1 shows the importance of safety-based design. A short overview of some

existing works in the field of software architecture design with a comparison between some of the more related works is presented in Section 4.2.2. Section 4.2.3 gives a brief overview of software product line architecture design methods. Section 4.2.4 describes the basic idea of safety pattern for product line. A detailed overview of the proposed ***Safety-driven Software Product Line Architecture Design Method*** (**SSPLA**) and its process steps with the interpretation of each step is presented in Section 4.3. Section 4.4 shows how the aforementioned safety-driven software product line architecture design method can be adapted to be statechart-centric architectural design method which is called "***State-driven Architectural Design Method for Software Product Lines*** **SAD**". The idea of how to address the variability in the statechart is explained in Section 4.5. Section 4.6 shows a simple, illustrative example to explain how the process steps of the proposed method has been implemented. Finally, Section 4.7 summarizes the chapter.

4.2 Overview of Software Architecture Design

Software architecture is the structure of the software system. It describes the software elements, their characteristics and their interactions with each other (Len Bass and Paul Clements and Rick Kazman, 2003; Systems, 2000). Qualified software architecture provides a blueprint for system construction and composition. It is a main factor to a successful software development (L Tan, Lin and Ye, 2012).

Software architecture design is a critical step of software development. There are many challenges in software architecture design for example, modeling the non-functional requirements, especially those requirements on the quality of the software. Non-functional requirements and quality attributes are important parameters of software products. Quality requirements of a system serve as a bridge between business goals and software architectures (L Tan, Lin and Ye, 2012).

4.2.1 Safety-based Architecture Design

The safety attribute is one of the important quality attributes. There are increasing in the attention of software safety, so how to improve software safety has already become a more important concerned issue, especially for the safety-critical systems

(Capilla *et al.*, 2014). Software safety assurance refers to a series of quality assurance activities during software development life cycle, which aims to eliminate the potential dangers. Currently, the influence of architecture in assurance of software safety is being increasingly recognized (Capilla *et al.*, 2014). Safety-based design at architecture level can effectively improve software or system safety.

4.2.2 A Comparison between Some of the More Related Works

There are a number of architectural design approaches or methods each one focuses on a specific perspective of the architecture design. Some of them are to design the architecture of single software where others for product line (M.Sharafi, 2013). In the other side, some of them do not address the quality attributes in the architecture and others address the single or multiple quality attributes (quality-oriented methods). Especially, quality-based methods have received a lot of attentions and have been well developed for single system architecture design. However, the use of quality-based design methods is limited in software product line (SPL) because of the complexity and variabilities existing in SPL architecture (Lei Tan, Lin and Ye, 2012) (Lei Tan, Lin and Ye, 2012). Our work is related to contributions from several other areas mainly architecture design, product line architecture and safety engineering. In the following table (Table 4.1) we will show in a briefly way the much related works which we built our method base on them. The summarized table presents some information about the works as well as advantages and limitations of each. Note that our method is mainly based on the work (Huang, 2013; Bayer, Flege and Gacek, 2000; M.Sharafi, 2013; Liu, Dehlinger and Lutz, 2007; Pinzger *et al.*, 2004; Jensen and Tumer, 2013), also the works (Yacoub, 1998) and (Rauhamäki, Vepsäläinen and Kuikka, 2012) considered as a basic for the next works of our research (e.g. developing a safety-oriented architectural pattern(s) for the SPLAs).

Our proposed process presented in this chapter is a safety-driven software product line architecture design method; the following points are the differences of this method in comparison with the other existing methods:

- In this method, the safety attribute of a software product line system will be taken into account in the early stage of architecture design and the reference architecture of SPL will be elicited based on safety-related consideration.

- The key aspect of this method is the using of the architectural patterns which are safety-driven architectural patterns to improve the design process of the software product line architectures.

Table 4.1: A comparison table between the more related works

REF	DATE	ADV+	LIMITATION
(Huang, 2013)	2013	- uses the selected combination of safety tactics to effectively improve the software or system safety, - providing a new way of thinking for software safety architecture design.	the weakness of this work is not take into account the family of architecture
(Bayer, Flege and Gacek, 2000)	2000	- It iss an integrated, iterative, and quality-centered method for the design and assessment of product family. - It is covers the whole product family development life cycle and can be introduced incrementally	It is difficult and time consuming. Also the approach considers multi quality attributes not focuses on specific one (e.g. the safety attribute)
) Pinzger *et al*., 2004(	2004	Easy method	- The architecture of all the existing products is reconstructed before the beginning of the reference architecture design, which requires considerable time and effort. - Don't address any quality attribute
(M.Sharafi, 2013)	2000	Saves time, effort and also helps product family architect to evaluate the architecture easily.	Not addressing the safety attribute as a key factor in the design process
(Liu, Dehlinger and Lutz, 2007)	2007	- More practical to check that safety properties for the product line hold in the presence of variations through scenario-guided execution, or animation, of the model. - allows safety engineers to discover faults early enough to design mitigation strategies before implementation and deployment.	Just technique to the safety analysis.

(Jensen and Tumer, 2013)	2013	They presented a process of concurrently developing a system concept from the safety and functional perspective.	the weakness of this work is not take into account the family of architecture
(Armoush, 2010)	2010	It is a good approach for the adoption of the design pattern concept for safety-critical embedded systems design.	It is just addresses the single architecture
(Yacoub, 1998)	1998	Using sate-based modeling (statecharts) which is a good way to address the safety attribute.	It is just addresses the single architecture

4.2.3 Safety-based Software Product line Architecture Design Methods

Software Product Line (SPL) engineering is about developing a collection of systems which share great commonalities ((L Tan, Lin and Ye, 2012; Bosch, 2000) and (Pleuss *et al.*, 2012)). (Liliana Dobrica, Eila Niemela, 2003; Liliana Dobrica, 2000). Product-line (PL) and reusable software components are suitable approaches for embedded systems, which are often re-engineered from existing systems.

Developing a reference architecture which represents the base structure of the member products is the main task of the software product line architecture design (Engström and Runeson, 2011). The software product line architecture (SPLA) (L Tan, Lin and Ye, 2012) provides a coarse grain picture of structure in the software product family. It initiates the architecture design for the member product. Important issues in the development and maintenance of these software systems are functionality and quality. As product-line engineering becomes more widespread, more safety-critical software product lines are being built (Feng and Lutz, 2005).

On the other hand, there are various design methods available and each is focusing on certain perspectives of architecture design. Safety-based methods have received increasing attention and have been well developed for single system architecture designs. However, the safety-based design methods are limited in SPLs because of the complexity and variabilities existing in SPL architectures.

There are two main questions we can address:

RQ1: Why the traditional quality-oriented architectural design methods (for single software) are not sufficient?

RQ2: How can we efficiently design safety-based product line architectures?

Based on the above, one of the main objectives of this research is to develop efficient and effective method that can be used into the design process of the safety-driven software product line architectures which enhances and manages the safety of software product line. This method characterizes the safety attribute as a central attribute in the design and captures the architectural patterns that are used to achieve this attribute.

4.2.4 Safety Patterns for product lines

Design patterns are popular in the field of software engineering and there are plenty of patterns. Contradictorily, in the field of safety engineering patterns have not been studied and published in such volumes (Rauhamäki, Vepsäläinen and Kuikka, 2012). And unfortunately, it is a complex task to build safe systems (Rehn, 2009). For safety the problem seems to be not that severe because fewer non-experts write software for safety-critical systems. But nevertheless building safe systems is evidently a complex task, too. In software-engineering reuse is a major means of reducing development effort and increasing quality by using existing solutions that are known to be well engineered. At the software architecture level this is done by so-called patterns and tactics (Rehn, 2009).

A statechart is an alternative means of system specification. This specification methodology is particularly oriented to “reactive systems” — that is, systems that respond to a series of events rather than transforming an input into an output (Niaz and Tanaka, 2003). Such systems may incorporate concurrent processing, and statecharts encompass this capability. The OO methodologies using statecharts describe in sufficient detail the steps to be followed for describing the behavior of objects (Niaz and Tanaka, 2003). In most OO methodologies, when people think of object behavior, they consider the functionality of the object. But in many real world applications this definition is insufficient — the internal state of an object and the quality attributes should also be considered (e.g. the safety in safety-critical systems). In other words, because the Unified Modeling Language (UML) statechart diagram is a powerful tool for specifying the dynamic behavior of reactive objects, we can use this facility to describe the system behavior in term of safety.

In this research we have developed a new Safety-driven design pattern (SDP). This pattern extends capabilities of both the statecharts design patterns and safety patterns. This pattern allows an object to alter its behavior and change its internal state when there is a safety violation, and to protect it from introducing in unsafe states. In this pattern we show how to solve recurring design problems in implementing safety-based statechart specification of an entity in object-oriented application mainly safety-critical applications.

This pattern is presented in more details in Chapter 5. In chapter 6 the assessment of this pattern is conducted by using a new safety assessment method to show the safety improvement after using this pattern. Then and because this pattern is for single software architecture we extended and adapted this pattern to be software product line design pattern and that in order to address the variability existing in the product line, See chapter 7.

4.3 The Proposed Safety-driven SPLA Design Method

In this research we have developed a new method to safety-driven software product line architectures design process (SSPLA). This method characterizes the safety attribute as a central attribute in the design and captures the architectural patterns that are used to achieve this attribute. The proposed process of our new method for safety-driven software product line architecture design is shown in Figure 4.1, which take a snapshot for the suggested method. The output of this method is a safety software product line architecture. Note that our method is based on a number of previous approaches or methods (Huang, 2013; Bayer, Flege and Gacek, 2000; M.Sharafi, 2013; Liu, Dehlinger and Lutz, 2007; Pinzger *et al.*, 2004; Jensen and Tumer, 2013) in context of architectures design in general, product line architecture design, safety-based design, and etc. In the following subsection we will describe the steps of the proposed process in our method.

4.3.1 The Method Process

Because this process is a safety-driven development, the most process operations or steps are based on the *safety attribute*. For instance, define test cases based on the

safety attribute, select a safety-driven architectural pattern(s), evaluate the architecture if met the safety requirements or not.

The architecture is created in a number of iterations by stepwise application of scenarios and by using proven solutions to recurring problems such as architectural patterns. Iterations are performed until all scenarios have been applied and no problems arose from the final assessment of the architecture. The input is a domain model and a scope definition, where the former defines the business case for the development of the product line and the latter describes commonalities and variations of applications within the product line. The Output of SSPLA is a safety product line architecture as defined in the introduction. Basis for the development of architecture are a prioritized list of business goals, functional and quality requirements (specifically, the safety). In the context of product families, the commonalities and variabilities among those goals and requirements also have to be known. Designing architectures requires software architecture experts to study a system and an active involvement of stakeholder representatives, such as testers, developers, manager, the business owning the system, and system users. The process steps are described in the following subsection.

4.3.2 The Process Steps

This method is based on a hierarchical system model. It is a process for creation and evaluation of product line architectures. The inputs of this process are the requirements or requirement specifications. It will be possible to define two types of the general requirements which are the domain model and scope definition. In general, we can say that the inputs of the process are domain model and scope definition of the product line.

Domain Model. Define the main requirements of the specific applications domain or family of products. And it is a requirements model. The system requirements are defined based on customer needs and or through development perspective from the developers. The requirements model is used to define the scenarios (scenarios of usages). So to build any architecture we must have a domain model which often created by modeling tools or languages (e.g. Use case model or Unified Modeling

Language). Example of domain model is a features model, it is an important modeling view for product line engineering because it addresses SPL variability.

The domain model is a domain model of a product line consists of two elements:

1) *generic workproducts*, for instance, products that describe the requirements in terms of common, variable features, like (the feature models).

2) *Decision model*, it is address the variability in the product line by determine the open decisions and possible solutions. In the other side, the domain decision determines the decisions about the common, variable features, non-functional requirements and etc. which must take into account. We should define the features structures that must select which define the features model, some of the structures can be right and others are not. Sometimes, the decisions is determine based on development view of the expected system and customer requirements.

Scope Definition. Define the commonality and variability of the deferent applications in product line. There are internal variable features between the components of the applications itself (Internal Variabilities) and others are variability features with respect to the environment (External Variabilities). So, some variabilities may be produced in the product line due to the differences in the environment, internal or external variabilites must be defined. Scope definition concerned with the process that searches to determine which of the system functions are essential in product line and which are optional. The scope shows the organization for the types of the developed products and for the others that will developed in future. The inputs for producing the scopes are come from organizational strategic plans, market staff and analysis and technology experts in the domain. The product scope is one of the main factors to determine the success to the product line. The major problem to define the scope is the definition of the similarity in the systems which decreases the development cost of the system for the organization.

Each step of the design process is briefly described in the following lines.

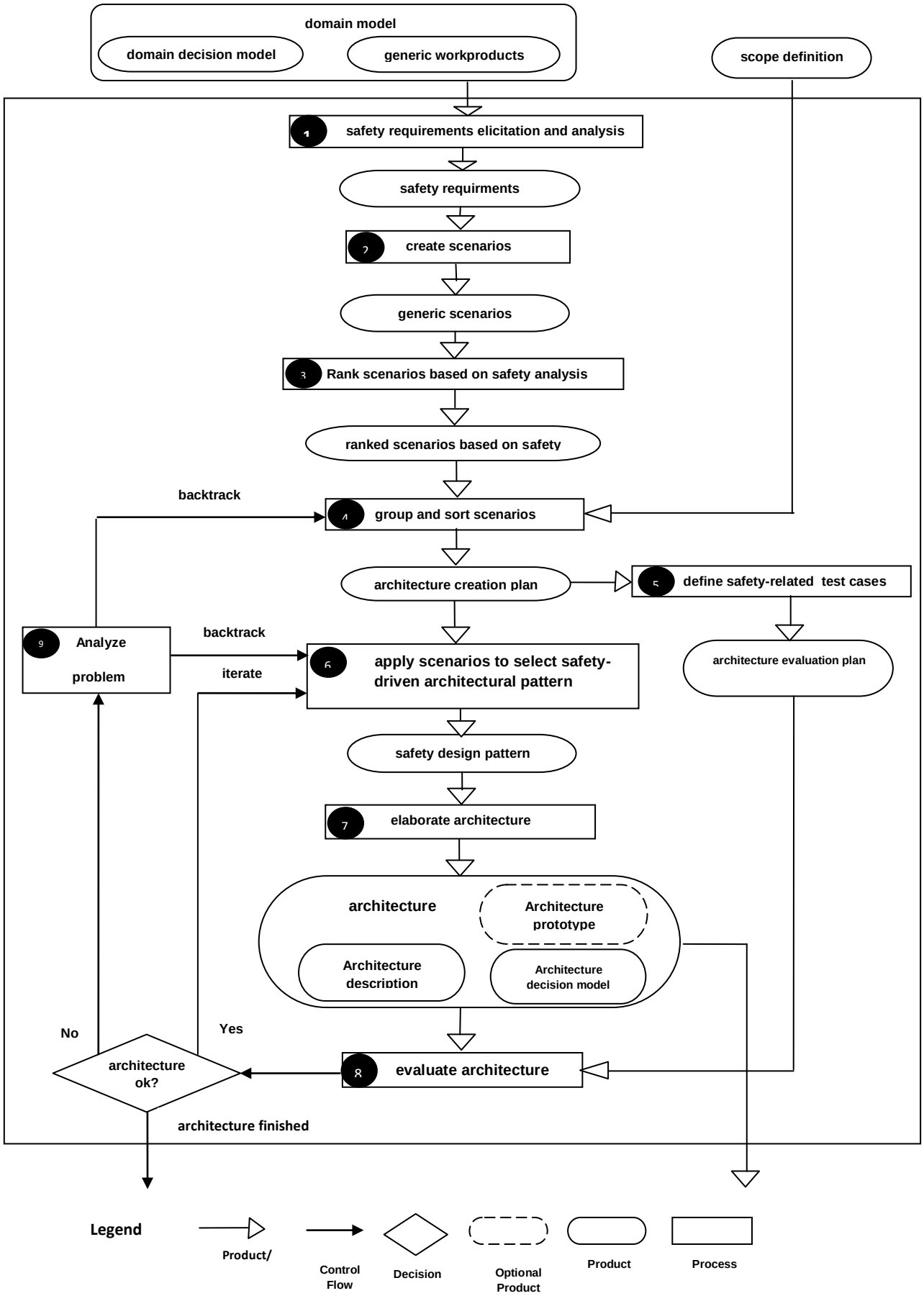


Figure 4.1: An overview of our proposed method (SSPLA)

Step 1: Safety requirements elicitation and analysis

This stage is one of the important stages in the process and all the next stages are depend on the success of this stage. The elicitation and analysis of the safety requirements for the product line is doing here. So, this step is to analyze the software requirement belong to one layer, and then according to the requirements specification, we should elicit functional requirements and safety requirements. However, not all the safety requirements can be individually extracted as some of them are often reflected in the functional requirements. There are many safety analysis methods or approaches in SPL domain (e.g. (Feng and Lutz, 2005) (Liu, Dehlinger and Lutz, 2007)). We can select a suitable and success analysis method from the available methods for safety analysis in product line domain. From product line point of view the safety requirements analysis is the commonality and variability, that means there is a commonalities and variabilities in the requirements of the safety attribute. For example the applications in product line share in some safety requirements (e.g. safety tactics or mechanisms) and differ in other. After completion of this step, the safety requirements of the software product line will have accomplished. And then go to the next stage (Step 2).

Step 2: Create Scenarios

As we mentioned above, that this process is a scenario-oriented process, so the architecture is created in iteration manner, by take the scenarios and then ranked and making it in a sorted groups (step 3 and 4 respectively).

The second step in the architecture creation is to determine the most important requirements. The input for this step would be a product line model consisting of generic workproducts (i.e., products describing requirements in terms of commonalities and variabilities) and a decision model, in addition to safety requirements. Now, the problem is analyzed, we should determine the architectural information required to solve the problem and the way to derive this information (Here the architecture information is driven by safety attribute).

We need to extract a limited number of scenarios that should be used in iterations. Indeed, the scenarios describe the functional and safety requirements of the product family the architecture is designed for. In the context of product families, scenarios

are generic that they do not only capture common but also variable requirements of the instances in the product family.

Step 3: Rank scenarios based on safety analysis

This step is to determine which scenarios are important than others based on safety requirements. By doing safety-based analysis of the scenarios we can rank the scenarios using any ranking technique. Prioritizing scenarios should follow a simple and basic rule: the bigger the impact of a scenario on the architecture safety, the higher the scenario's priority.

The next step is based on this step and that to group and sort the scenarios. Note that, sorting based on safety requirements - the scenarios which have more critical in term of safety than others. The output of this step is ranked scenarios, based on safety hazard.

Step 4: Group and sort scenarios

Note that, the order in which scenarios are addressed in the previous step is very important, those scenarios that are considered to have the highest significance for the architecture should be selected for the first iteration. In the next iteration, the second most important group is selected and so forth. This step yields the architecture creation plan that defines the iterations in which the architecture development is performed. The first iteration deals with the most important group of scenarios, the second one with the second most important group and so forth. The judgment of a scenario's importance cannot be based on its expected impact on the architecture alone, but mainly based on a safety. The sorting in which the scenarios addressed is very important, that because all the decisions of the iterations design impose constraints on the architecture which determine its next development. Resulted architecture creation plan from this step used to begin the creation of the architecture, and then over time we develop it and add other components or sub components of the other set of scenarios.

And then go to the next two steps (step 5 and 6) where often are executed concurrently, that means define an evaluation plan or method for any resulted architecture in each iteration.

Step 5: Define safety-related test cases

In case an architecture assessment (or evaluation) should be performed at the end of the iteration, assessment (or evaluation) criteria have to be defined according to the business and quality goals. Defining assessment criteria before the actual design begins has several advantages such as a better understanding of the requirements and avoidance of specifying criteria that, due to an already influenced perspective, merely support what has been developed. For each group of scenarios, test cases are defined that will be used to evaluate the architecture at the end of each iteration. Here, the main assessment criterion, quality goal is the safety attribute. This safety attribute play a central, critical role for the appropriateness of reference architecture. Define test cases is based on safety. However, it is extremely difficult to assess the degree to which this attribute is achieved by a given architecture.

Often the test cases are define early that because we need creating a document as a plan to asses or evaluate the architecture. The output of this step is a definition of architecture evaluation plan. This plan is used to evaluate the architecture in the end of the each iteration and in the last evolution of the architecture.

Step 6: Apply scenarios to select safety-driven architectural pattern

Scenarios based on their priorities are entered into the design cycle. In this step and based on the sorted scenarios we need to select appropriate safety-driven architectural pattern(s) that based on safety analysis. In other words, we should select an architectural pattern(s) which suit each scenario. And defining the decision model which includes the architectural decisions about the determination of the scenarios and the architectural patterns, for instance what are the selection criteria of the architectural pattern. Note that the main criterion is the safety. Note that one of the main contributions of our research is a developing of a safety-based architectural pattern (s), (see Chapter 5 and 6).

Step 7: Elaborate architecture

The group of scenarios associated with the current iteration and by using the selected architectural pattern(s) in the previous step is used to create the initial architecture, refine/extend an already existing or partial architecture. In this step, the architecture's elements or components and their relationships are defined, that for the selected group of scenarios. That result in either an initial architecture (initial architectural product) or a part of the architecture in the other iterations, or

improvement to the architecture. This process is continues until the completion of the architecture. In addition, this step consists in each iteration an architecture descriptions (all or part) and architectural decision model. The decision model includes the architectural decision about the definition and selection of resulted architecture.

Step 8: Evaluate architecture

In general, the given architecture is checked with respect to functional and quality requirements and the achievement of business goals. In this step, the architecture resulting from the previous step is evaluated according to the architecture evaluation plan which is based on safety. Evaluating architecture is based on test cases which based on the safety (Checking, if the safety requirements are met or not). If the evaluation is successful (i.e., all tests are passed), the architecture development continues with the next iteration or is finished once the last group of scenarios has been applied. If, however, at least some tests failed, the process continues with step 9 "Analyze Problem" then go back to do another iteration. The ease of deriving instance from the product line system is necessary and also refers to the efficiency of the architecture. So, the ease of the derivation must also be evaluated. In the other side, the evaluation process for the optimization of the architecture is hard because we need the optimization for all the products derived from the product line architecture. According to above, we can distinct three cases might be happened in the evaluation:

1. The architecture is ok. There are no problems in the resulted architecture. In this case a simple iteration is happens for taking another scenarios to add components or elements to the architecture which it started actually. Then go to step 6.

2. The evaluation shows that there is a need to improve the architecture, so we can go to step 6 too. This to repeat the architecture building process and then it could evaluate again.

3. There is a problem(s) in the resulted architecture. In this case the problem(s) will be analyzed in (step 9).

Step 9: Analyze problem

At least one of the tests for evaluating the current architecture failed. This step is to analyze the architectural problem that emerged. Examples of the problems are: the

problem in the attributes conflict or the required quality. But the main problem is the achieved level of safety attribute. In this step, the underlying problem is examined in order to determine how the architecture development process can be continued. The examination focuses on whether the current group of scenarios could be applied successfully to the architecture that resulted from the previous iterations. If this is deemed to be the case, only the current iteration needs to be reiterated. Otherwise, some design decisions from an earlier iteration are presumed to impose constraints that are too stringent for the current set of scenarios. Therefore, extended backtracking is needed, which may include reformulating, regrouping, and reordering of some scenarios and then reentering the process in the appropriate iteration. we can need going to the step 6 in case the problem is in the using or applying of the scenarios or in building of the architecture, or requiring to come back to step 4 if there is a possibility that there is a problem in the scenarios itself, or needing for some modifications in it.

4.4 State-driven Architecture Design for Safety-critical SPLs

As the Unified Modeling Language (UML) statechart diagram is a powerful tool for specifying the dynamic behavior of reactive objects, we can use this facility to describe the system behavior in term of safety.

In the previous section (Section 4.3) we describe our proposed safety-driven SPL architecture design method. In this method process the architecture is created in a number of iterations by stepwise application of scenarios and by using proven solutions to recurring problems such as architectural patterns. The output of this method is SPL architecture. Because this process is a safety-driven development, the most process operations or steps are based on safety attribute. This method is considered a generic safety-oriented software product lines architectures design method.

Since our work searches to define an effective and efficient method to enhance the development of the safety-critical software product lines the mentioned method has been adapted to be a statechart-centric method; which means most of method steps should be based on or around the statechart semantics.

For example: in step 1 we use the system statechart model to elicit and analysis the safety; in step 2 create the scenarios from the statechart; Also in step 6 and in the architecture elaboration step we can select safety-driven statechart pattern. And finally the architecture analysis and evaluation step can also be based on the statechart design model.

This resulted method is called State-driven Architectural Design for Software Product Lines.

In this context and in term of using architectural pattern, we present in Chapter 5 a proposed safety pattern of statechart. This pattern extends capabilities of both the statecharts design patterns and safety patterns. The pattern allows an object to alter its behavior and change its internal state when there is a safety violation, and to protect it from introducing in unsafe states. The critical aspect of the pattern is the ability to capture the dynamic nature of the safety attribute. By this pattern we can monitor and control the safety of the system in each state.

The critical issue of using the statechart modeling for the software product line is how to address the variability (Gomaa, 2011). This issue is addressed in some works (see (Gomaa, 2011)) and it is presented in Section 4.5 bellow.

4.5 Addressing the variability of the SPLs in the statechart.

As we mentioned above, that the critical issue of using the statechart modeling for the software product line is how to address the variability (Gomaa, 2011). This issue is addressed in some works, e.g. (Gomaa, 2011). We will give a brief overview about it in this section.

The works in (Gomaa, 2011) (Gomaa, 2004) show that there are two main approaches we can used when variable classes are developed, specialization or parameterization. In the case when there are small number of changes to be made then we can select specialization as we can manage the specialized classes (Gomaa, 2011) (Gomaa, 2004).

Based on these works and because the complexities exist in the PL we also propose using of parameterization approach. It is obvious that it is more effective to design a parameterized state machine, in which there are feature-dependent states, events, and transitions. Optional state transitions are specified by having an event

qualified by a Boolean feature condition, which guards entry into the state. Optional actions are also guarded by a Boolean feature condition, which is set to true if the feature is selected and false if the feature is not selected for a given SPL member, for more detail see reference (Gomaa, 2011)Gomaa, 2004).

4.6 Illustrative Example

This section present a simple application example to briefly show how to use our architectural design method.

In the other words, in order to illustrate our method for designing the software product line architectures, we use the Door Control Product Line System for a Smart Home. This is done to give a more comprehensive presentation of how the method can be used. We would like to point out that our example data have been taken from the (Feng and Lutz, 2005), “Qian Feng, Robyn R. Lutz, Bi-Directional Safety Analysis of Product Lines, 2005".

We have summarized the example by getting a general description for how to apply our methodology in a real system.

The Door Control System is a safety-critical product line. The software must function correctly to prevent intruders from entering and must respond correctly to life-threatening scenarios such as fires. A Smart Home system serves as an invisible housekeeper: it has sensors and agents to interact with humans and the environment to offer people convenience and safety. For example, the entrance doors can be opened only by inputting fingerprints or voiceprints. We restrict our discussion in this example to the Door Control System software product line with three products: a FrontDoor, a BedRoomDoor, and a SecurityDoor. This system does not address some of the more complex door features, such as maintenance modes, but is rich enough to be interesting.

4.6.1 The Process inputs:

As we mentioned in section 3.1, that the input of our method are the requirements or requirement specifications. It will be possible to define two types of the general requirements which are the domain model and scope definition. In general, we can

say the inputs of the process are domain model and scope definition of the product line.

4.6.1.1 General System Requirements

4.6.1.1.1 Commonality and Variability Analysis

The Commonality and Variability Analysis (CA) of a product line provides a requirements specification for the product line. The CA consists of the terminology used, the commonalities, the variabilities, and the dependencies among the variabilities. The dependencies are constraints that the choice of one features places on the choices of other features (Feng and Lutz, 2005). Note that we here exclude any non-behavioral commonalities and variabilities to focus on the software. The CA serves as a requirement specification for the product line and as an input to the product line's architecture design.

In this section we show the use case model and features model.

For single systems, use case modeling is the primary vehicle for describing software functional requirements. For SPLs, feature modeling is an additional important part of requirements modeling. The strength of feature modeling is in differentiating between the functionality provided by the different family members of the product line in terms of common functionality, optional functionality, and alternative functionality (Gomaa, 2011).

4.6.1.1.2 Use Case Modeling for the Door Control SPL

For a single system, all use cases are required. In a SPL, only some of the use cases, which are referred to as kernel use cases, are required by all members of the family. Other use cases are optional, in that they are required by some but not all members of the family. Some use cases might be alternatives to each other (i.e., different versions of the use case are required by different members of the family).

In UML, the use cases are labeled with the stereotype «kernel», «optional» or «alternative» (Gomaa, 2011). In addition, variability can be inserted into a use case through variation points, which specify locations in the use case where variability can be introduced (Gomaa, 2011).

The main use cases are:

1. *Registration*: registering the users' ID to the Door Control system (Commonality)

2. *Entry*: entering the house from the outside (Including recognition from outside, opening the door, closing the door after the people pass, also including the illegal entrant handling) (Commonalities)

3. *Exit*: exit the house from the inside (Including recognition from inside, opening the door, closing the door after the people pass, also including the illegal going out handling) (Commonalities)

4. *Fire alarm*: the door's response to the fire alarm (Commonality)

5. *Bolt*: lock door from inside (Variability)

The kernel and optional product line use cases for the Door Control SPL are given in Figure 4.2.

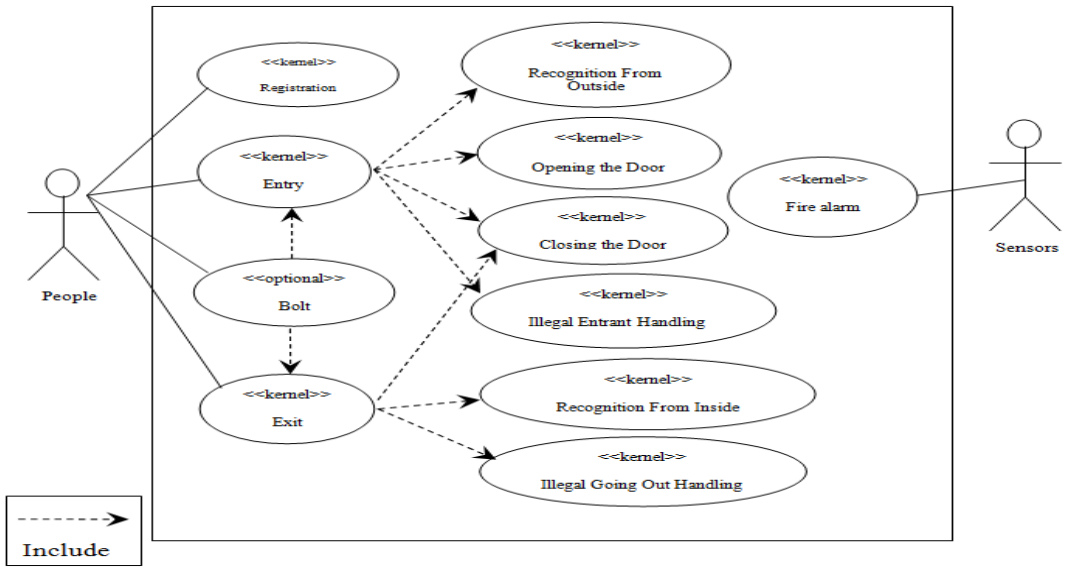


Figure 4.2: Door Control Software product line use cases

Variation points are provided for both the kernel and optional use cases. One variation point concerns the methods of registration: fingerprint or voiceprint. FrontDoor: fingerprint; BedRoomDoor: voiceprint; SecurityDoor: fingerprint and voiceprint. This variation point is of type mandatory alternative, which means that a selection among the alternative choices must be made.

……

Variation point in Identification use case:

Name: Type of Registration Method.

Type of functionality: Mandatory alternative.

Description of functionality: There is a choice of the type of method for identification. The alternatives are: fingerprint, voiceprint, fingerprint and voiceprint.

4.6.1.1.3 Feature Modeling

Feature modeling is an important modeling view for product line engineering, because it addresses SPL variability. Features are incorporated into UML in the Product Line UML-based Software (PLUS) method using the meta-class concept, in which features are modeled using the UML static modeling notation and given stereotypes to differentiate between «common feature», «optional feature», and «alternative feature» (Gomaa, 2011)(Gomaa, 2004).

The feature model for the Door Control SPL is shown in Figure 4.3.

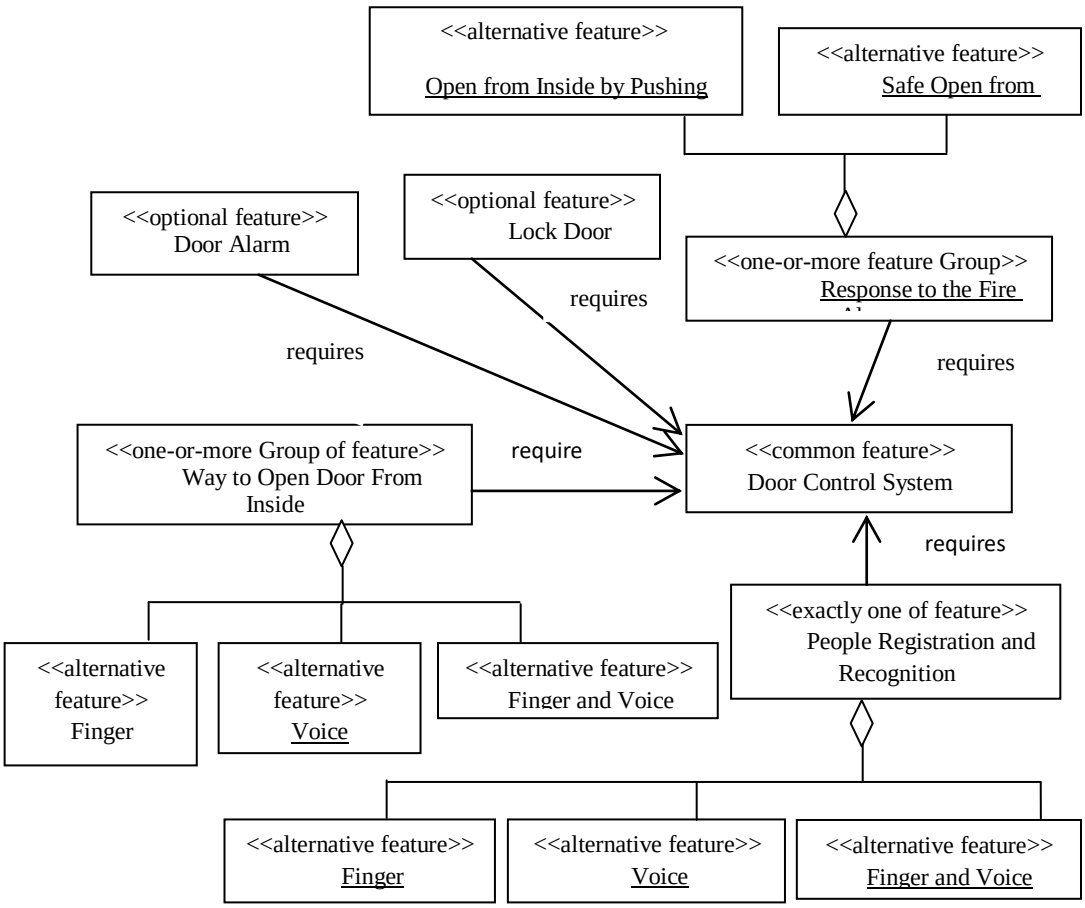


Figure 4.3: The Feature Model foe Door Control Software product line

4.6.1.2 Analysis Modeling for the Door Control SPL

As with single systems, analysis modeling consists of both static and dynamic modeling. However, both modeling approaches need to address modeling SPL variability.

4.6.1.3 Dynamic Modeling for the Door Control PL System-Using Statechart Modeling

When variable classes are developed, there are two main approaches to consider, specialization or parameterization (Gomaa, 2011). In product line development, however, there can be a large degree of variability. Consider the issue of variability in state-dependent control classes, which are modeling using state machines and depicted on statecharts. To capture state machine variability and evolution, it is necessary to specify optional states, events and state transitions, and actions. It is often more effective to design a parameterized state machine, in which there are feature-dependent states, events, and transitions. Optional state transitions are specified by having an event qualified by a Boolean feature condition, which guards entry into the state. Optional actions are also guarded by a Boolean feature condition, which is set to *True* if the feature is selected and *False* if the feature is not selected for a given SPL member See reference (Gomaa, 2011).

Figure 4.4 presents a part of the statechart specification for Door Control PL which depicts seven states (Waiting for ID, Door Opening from outside, Door Opening from inside, Door Opened, Door Closing, Door Closed, and Door Locked Inside). Locking the door from inside is an optional door control feature. In the statechart, Door locked is a feature-dependent state transition from Door Closed state to Door Locked Inside state. This state transition is guarded by the feature condition "an inside lock door button" which is True if the feature is selected.

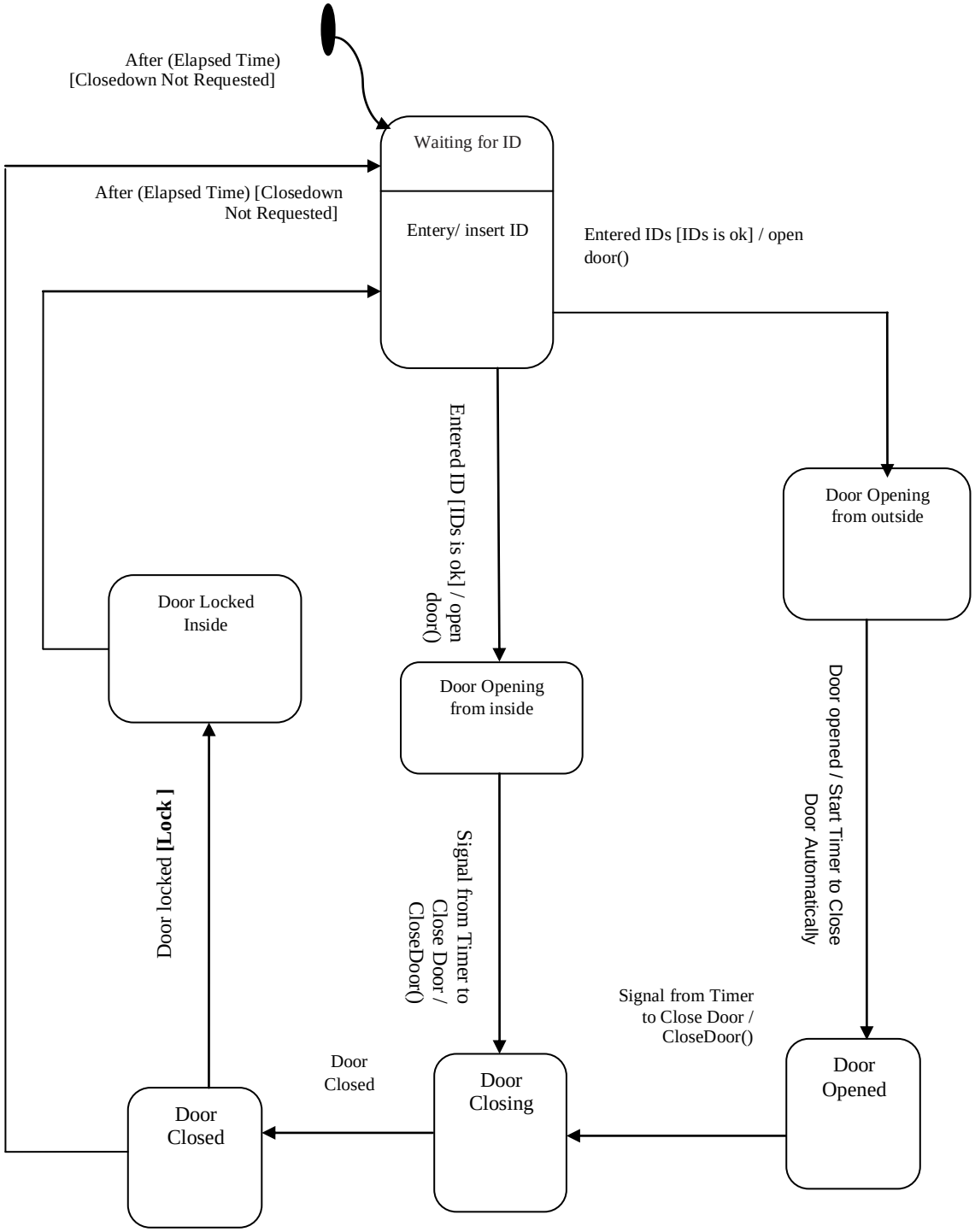


Figure 4.4: A part of the statechart specifications for Door Control PL System

All the data above (Sec. 4.6) are considered to be an input to our methodology. Below, we describe how each step of the methodology is applied in the study.

The following lines describe in general how to apply the process steps of our method for developing a product line architecture of a given system.

In step 1: (*Safety requirements elicitation and analysis*) we can use any safety analysis method(s) for product line to do the step of safety analysis in our method. Here in this example we used the "Bi-Directional Safety Analysis of Product Lines" proposed by [Qian Feng, Robyn R. Lutz] in (Feng and Lutz, 2005). It is a sufficient methodology and a bi-directional in that it combines a forward analysis (from failure modes to effects) with a backward analysis (from hazards to contributing causes). This methodology of the software safety analysis uses the Extended Commonality and Variability Analysis (XCA) and a hazards list to drive the bi-directional safety analysis (Feng and Lutz, 2005). Findings from application of the bi-directional safety-analysis method included new safety-related software requirements both for all the systems in the product line (commonalities) and for only some of the product-line systems (variabilities), as well as discovery of a new hazard, that people can be pinned by the door.

Step 2, 3, and 4: Second we can summarize the works that have done in step 2, 3, and 4 as follow:

Here and after the safety analysis in step1 we repeat the analysis process again in order to create revised scenarios. And as we mentioned above, that this process is a scenario-oriented process, so the architecture is created in iteration manner, by take the scenarios and then ranked and making it in a sorted groups. Figure 4.5 can present one of the final results for these steps. As we see, Figure 4.5 is a statechart specification and a safety-oriented solution. That leads to the question "why we use the statechart modeling?" The answer is "because we want to select a safety-driven design pattern of statechart which enhances the safety in the software architecture".

Step 5: Define safety-related test cases: The test cases are defined early that because we need creating a document as a plan to asses or evaluate the architecture. The output of this step is a definition of architecture evaluation plan. This plan is used to evaluate the architecture in the end of the each iteration and in the last evolution of the architecture.

Step 6: Apply scenarios to select safety-driven architectural pattern:

In this system we selected a safety-driven architectural pattern. We used our new proposed pattern see (section 5.3) which presented in Figure 5.2.

Step 7, 8, 9: After applying the steps 7, 8, and 9 the final architecture is produced. Figure 4.6 shows an abstract view of the final architecture of the Door Control PL which depicts it in an object-oriented design. Figure 4.7 shows an architecture of the Door Control PL developed by traditional methods. The reader can compare between this (the architecture in Figure 4.7) and the architecture developed by our method (the architecture in Figure 4.6).

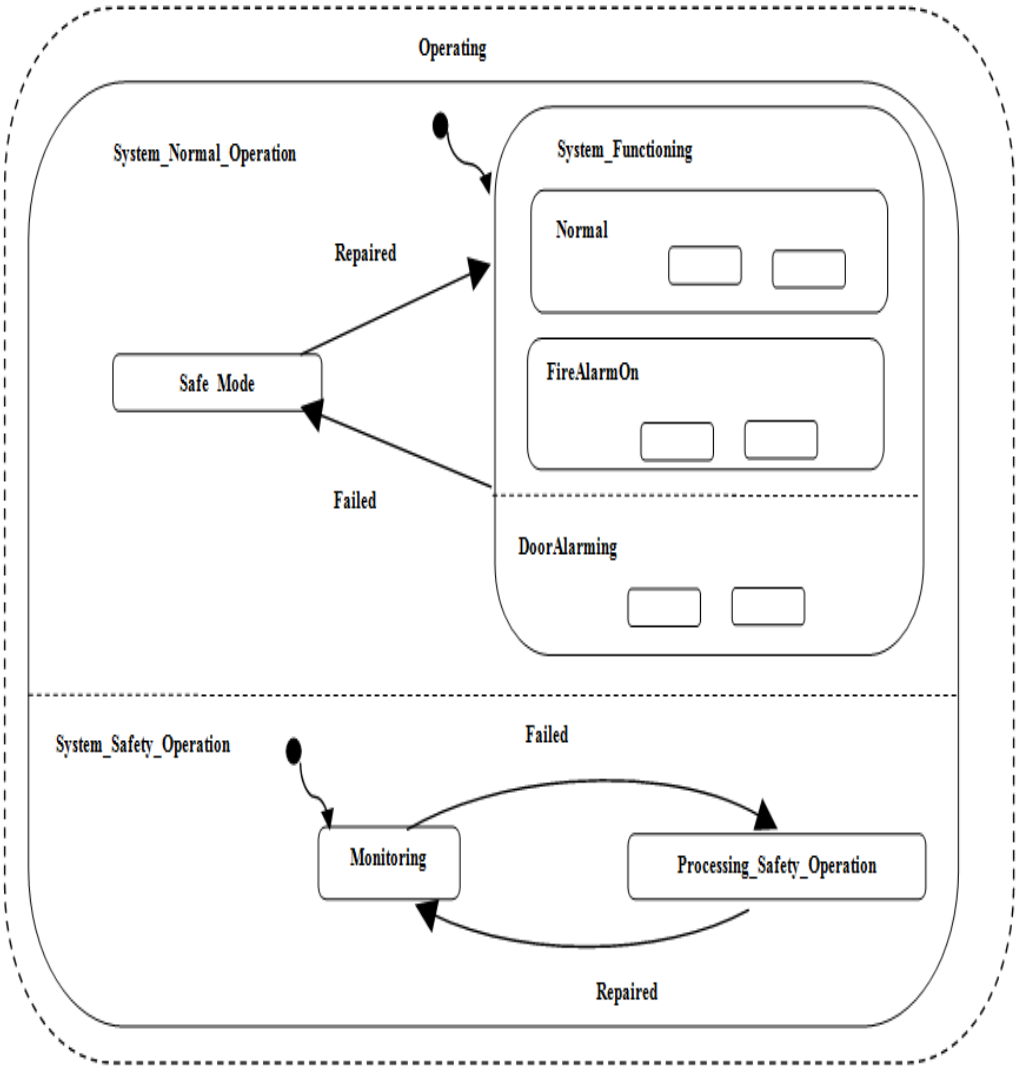


Figure 4.5: Abstract view of safety-driven statechart specification of the system

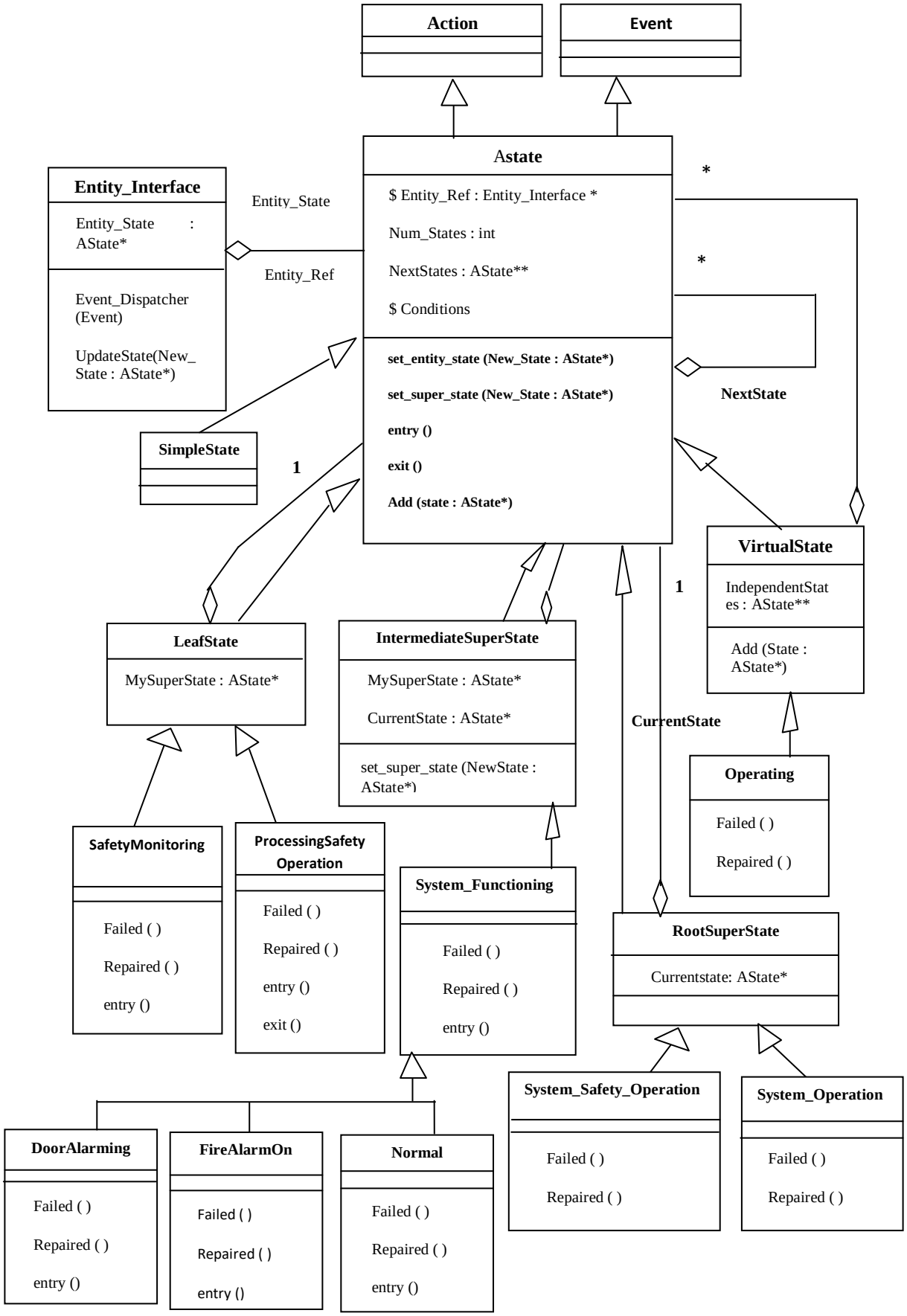


Figure 4.6: The architecture of the Door Control PL developed by our proposed method

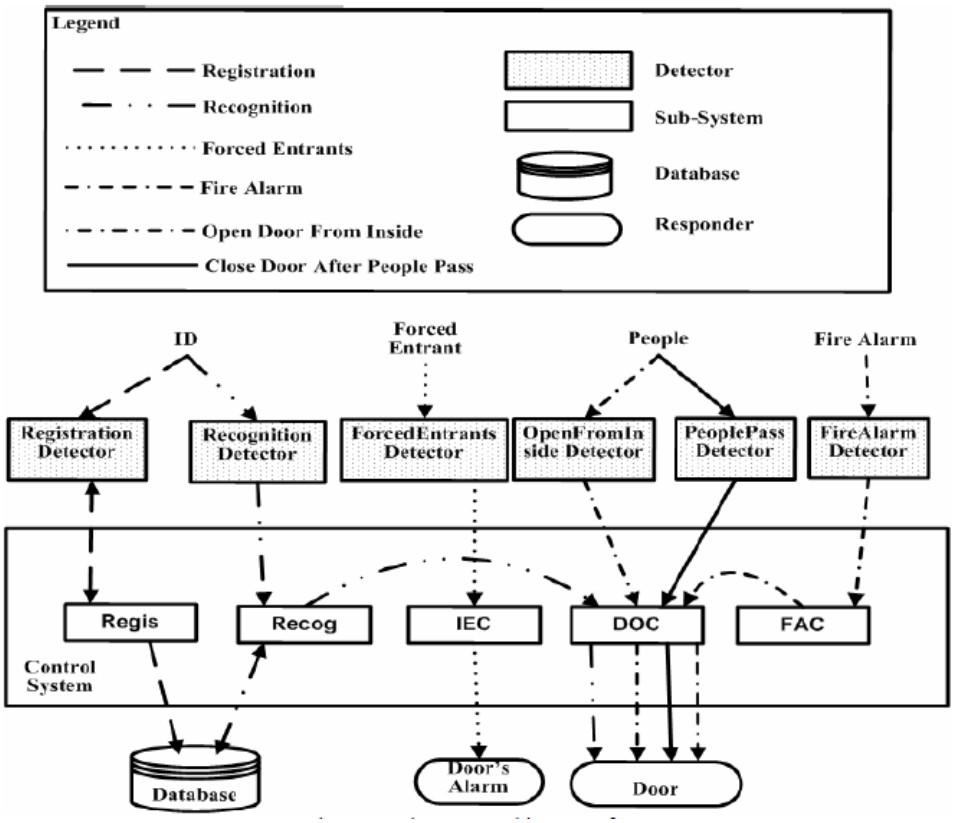


Figure 4.7: An architecture of the Door Control PL developed by traditional method (Feng and Lutz, 2005)

4.7 Chapter Summary

As mentioned in the previous chapters that the main objective of this research is to find an efficient and effective method that can be used into the design process of the safety-critical software product line architectures which enhances and manages the safety of the software product lines. In this chapter, a Safety-driven Architectural Design Method for Safety-critical Software Product Line has been proposed. This method characterizes the safety attribute as a central attribute in the design and captures the architectural patterns that are used to achieve this attribute. The method is based on a number of previous approaches or methods (Huang, 2013; Bayer, Flege and Gacek, 2000; M.Sharafi, 2013; Liu, Dehlinger and Lutz, 2007; Pinzger *et al.*, 2004; Jensen and Tumer, 2013) in context of architectures design in general, product line architecture design, safety-based design, and etc. A detailed overview of the proposed Safety-driven Software Product Line Architecture Design Method

(SSPLA) and its process steps as well as the interpretation of each step is also presented in this chapter. The key aspect of this method is the use of the concept design patterns which improves the design process. The next two chapters, Chapter 5 and 6, show the research contributions in the context of the design patterns. The adaptation of the safety-driven SPLA design method to be a State-driven, Statechart-centric Method has been also presented in this chapter. Finally, a simple application example to briefly show how to use our architectural design method is also presented in this chapter.

The implementation and evaluation processes to the overall research works are described in Chapter 8.

CHAPTER V

5

SAFETY PATTERN OF STATECHART FOR SPLAs

5.1 Introduction

Some of the content in this Chapter has been presented at 7th IEEE International Conference on Mechatronics Engineering, ICOM2019, Malaysia (Ebnauf and Al., 2019).

As mentioned in previous chapters that this research focuses on how to consider safety in software product line at the architecture design phase. It searches to define an effective and efficient method to enhance the development of safety-critical product lines systems. The research proposed a safety-driven SPL architecture design method, Chapter 4 (Sec. 4.3). And then this method has been configured and adapted to be state-driven architecture design method, Chapter 4. The main idea is to make this method a state-centric method, as described in Chapter 4 (Sec. 4.4). The key aspect of this method is the use of the concept design patterns which improves the design process.

In the other hand considering the design pattern paradigm, the research extends the capabilities of both the traditional safety patterns and statechart design patterns to develop safety-driven design pattern of statechart. This last point constitutes one of the main contributions of this thesis. The contribution is to develop a safety pattern based on the statechart which can be used to enhance the architectural design process for the safety-critical software product lines.

The developed safety design pattern—Safety-driven design pattern of statecharts constitutes one of the essential parts of this research. The pattern allows an object to alter its behavior and change its internal state when there is a safety violation, and to protect it from introducing in unsafe states. The result is an object-oriented design pattern which handles the safety attribute. The critical aspect of the pattern is the ability to capture the dynamic nature of the safety attribute. By this pattern we can monitor and control the safety of the system in each state.

To illustrate the effect of the design pattern in the PLA design, two case studies are used as running examples, Chapter 8. In Chapter 7 the thesis describes a new simplified safety assessment model which is used to evaluate the safety improvement in the design of the SPLA after using the proposed safety design pattern, specifically, the using the enhanced version of the pattern. As we will see in Chapter 8 that the results show that there is a considerable improvement in the system safety design after using the safety pattern.

The increased interactivity between cyber and physical systems and connectivity lead to a new safety and security challenges. And as there is a tight interplay between safety and security, combining safety and security in the engineering process for cyber-physical system has become a critical process. Regarding this criticality with the context of the safety patterns the research tried to enhance the proposed safety pattern (s) and that by proposing a new pattern development approach.

This chapter introduces the proposed safety design pattern and Chapter 6 describes how to improve this safety pattern to address the influence of the security issues on the safety. Chapter 6 also presents another version of the proposed safety pattern of statechart which contains addressing of the security that causes safety risks to the systems. The idea of how to extend the statechart pattern to capture the variability in the software product lines is also described in this chapter.

The remainder of this chapter is organized into four main sections. Section 5.2 presents overview of the architectural design patterns. Section 5.3 describes the proposed safety pattern of statechart. The section also describes the pattern description and gives a brief overview of the idea of the statechart pattern extension to capture the variability in the software product lines, Subsections 5.3.1 and 5.3.2 respectively. The chapter ends with a summary in Section 5.4.

5.2. Overview of the Architectural and Design Patterns

The pattern-based approach proposes patterns as a method of capturing expert solutions to many common software problems (Jamal and Eric, 2003). Patterns help you build on the collective experience of skilled software engineers (Buschmann and Maunier, 2001).

From the software engineering point of view, a design pattern is a description or template for how to solve a problem that can be used in many different situations (Buschmann and Maunier, 2001). It is a general reusable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code (Buschmann and Maunier, 2001).

A design pattern describes a design problem which repeatedly occurred in previous designs, and then describes the core of the solution to that problem.

Solutions are expressed in terms of classes of objects and interfaces (object-oriented design patterns).

Particularly, in the object-oriented community, patterns have been used as a methodology to document the best practices and experiences of object-oriented software systems (Hautamäki, 2005).

Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved (wikibooks).

Every pattern deals with a specific, recurring problem in the design or implementation of a software system (Buschmann and Maunier, 2001). Patterns can be used to construct software architectures with specific properties (Buschmann and Maunier, 2001).

Some Advantages of the design pattern:

- Design patterns can speed up the development process by providing tested, proven development paradigms.
- Design pattern support the effective software design process by addressing the issues which may not become visible until later in the implementation.
- Design patterns allow developers to communicate using well-known, well understood names for software interactions.

- Reusing design patterns helps to prevent subtle issues that can cause major problems.
- They capture existing, well-proven experience in software development and help to promote good design practice (Buschmann and Maunier, 2001).

The pattern can simply be described as follow: "a pattern is a named problem/solution pair that can be applied in new context, with advice on how to apply it in novel situations and discussion of its trade-offs.", Frank Buschmann et al. (Buschmann and Maunier, 2001).

Types of design patterns:

- *Structural patterns* address concerns related to the high level structure of an application being developed.
- *Computational patterns* address concerns related to the identification of key computations.
- *Algorithm strategy patterns* address concerns related to high level strategies that describe how to exploit application characteristic on a computation platform.
- *Implementation strategy patterns* address concerns related to the realization of the source code to support how the program itself is organized and the common data structures specific to parallel programming.
- *Execution patterns* address concerns related to the support of the execution of an application, including the strategies in executing streams of tasks and building blocks to support the synchronization between tasks.

Frank Buschmann et al. in (Buschmann and Maunier, 2001), and in order to refine the above classification, they have grouped patterns into three categories:

1. Architectural patterns
2. Design patterns
3. Idioms

Each category consists of patterns having a similar range of scale or abstraction.

The following sub sections present a summary of each category.

1. Architectural Patterns

The architectural patterns help in the architectural design process. Viable software architectures are built according to some overall structuring principle. These principles are described with architectural patterns.

> ***"An architectural pattern expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them."***, Frank Buschmann et al. (Buschmann and Maunier, 2001)

Architectural patterns are templates for concrete software architectures. They specify the system-wide structural properties of an application, and have an impact on the architecture of its subsystems. The selection of an architectural pattern is therefore a fundamental design decision when developing a software system.

Example:

The Model-View-Controller pattern is one of the best-known examples of an architectural pattern. It provides a structure for interactive software systems.

2. Design Patterns

> ***"A design pattern provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes a commonly-recumng structure of communicating components that solves a general design problem within a particular context [GHJV95],"*** Frank Buschmann et al. (Buschmann and Maunier, 2001)

Design patterns are medium-scale patterns. They are smaller in scale than architectural patterns, but tend to be independent of a particular programming language or programming paradigm. The application of a design pattern has no effect on the fundamental structure of a software system, but may have a strong influence on the architecture of a subsystem.

Many design patterns provide structures for decomposing more complex services or components. Others address the effective cooperation between them, such as the following pattern: Observer (Gama, Helm, Johnson, 1995) or Publisher-Subscriber (339).

3. **Idioms**

Idioms deal with the implementation of particular design issues.

> ***"An idiom is a low-level pattern specific to a programming language. An idiom describes how to implement particular aspects of components or the relationships between them using the features of the given language,"*** Frank Buschmann et al. (Buschmann and Maunier, 2001)

Patterns are normally not invented but discovered (Rauhamäki, Vepsäläinen and Kuikka, 2012). Someone realizes that a recurring problem was solved the same way several times. This already is the pattern, the recurring solution to the recurring problem. It can then be named, formalized and documented in order to make reuse possible (Rauhamäki, Vepsäläinen and Kuikka, 2012).

5.2.1 Pattern Composition

In general, most software systems cannot be structured according to a single architectural pattern and they must support several system requirements that can only be addressed by different architectural patterns (e.g. design for flexibility and adaptability) (Buschmann and Maunier, 2001). Therefore, we must combine several patterns to structure such systems. The selection of an architectural pattern, or a combination of several, is only the first step when designing the architecture of a software system (Buschmann and Maunier, 2001). Pattern composition has been shown as a challenge to applying design patterns in real software systems (Hallstrom, Soundarajan and Tyler, 1995).

5.2.1.1 Safety Patterns and Statechart Patterns

To the best of our knowledge, there are various pattern approaches in the context of patterns composition, specifically at the architectural design level. In comparison to our work, none of them approaches shows clearly the pattern composition process to develop the patterns. Pattern composition is considered as a challenge.

In the other side of our work, the statechart patterns, MOODS is one well-known work on Object-Oriented state machines which presents an alternative technique of

selecting the most optimal design among different state machine patterns (Ran, 1995). Yacoub in (Yacoub, 1998) has attempted to combine multiple state machine design patterns into a cohesive unit. His work is based on the concept of statecharts developed by Harel. His work is considered as a first attempting in this context. In (Adamczyk, 2003) The work shows how different design patterns solve different problems given a specific context and a set of expectations (e.g. flexibility of design, loose coupling between elements, performance, etc). The uniform format of presentation aims to help software designers select the FSM (Finite State Machine) most appropriate for their needs.

The object of the proposed design method (Sec. 4.3) is to enhance and manage the safety of software product line. In the other hand, considering the design pattern paradigm, our work search for extending the capabilities of both, the traditional safety patterns and statechart design patterns to develop a new safety-driven design pattern of statechart. The last point constitutes one of the essential parts of this work.

The developed safety design pattern—*Safety-driven design pattern of statecharts* (Sec. 5.3) constitutes an essential part of this chapter. This pattern extends capabilities of both the statecharts design patterns and safety patterns. The pattern allows an object to alter its behavior and change its internal state when there is a safety violation, and to protect it from introducing in unsafe states. The critical aspect of the pattern is the ability to capture the dynamic nature of the safety attribute. By this pattern we can monitor and control the safety of the system in each state.

5.2.1.2 Safety Pattern with Security Control

The modern systems cannot be reliable and safe if they are not secure. In the field of control, safety and security engineering patterns have not been studied and published in such volumes. In this context, the issue is how to effectively address the influence of the security in the safety design process using patterns. ***A Pattern Development Approach*** that interlinks safety and security patterns has been proposed, Chapter 6 (Sec. 6.3). The approach is then used to enhance the proposed safety design pattern of statechart described in this chapter that in order to address the security in the pattern (see Sec. 6.4). This developed version is considered as a new ***safety and security pattern***.

5.2.2 Pattern-based SPLAs Development

Developing a reference architecture which represents the base structure of the member products is the main task of the software product line architecture design (Systems, 2000). It is evidence that a pattern based development of the reference architecture can support the development and application process of product lines(Philippow, 2003).

5.3 Safety Design Pattern of Statechart- a Proposed Pattern

The design patterns concepts are important in the systems design process. They are used to support and help designers and system architects choose a suitable solution for a recurring design problem among available collection of successful solutions (Jamal and Eric, 2003). Design patterns are popular in the field of software engineering and there are several patterns. Contradictorily, in the field of safety engineering patterns have not been studied and published in such volumes. Therefore, we have now answered the call.

In the other hand the design pattern approach is widely used in object-oriented software design. It defines reusable mechanisms for collaboration and interaction among classes or among objects to solve common object-oriented problems in different domains (Niaz and Tanaka, 2003). In this context there is several design patterns have been proposed to implement statecharts.

Three major questions we can address here:

Q1: How can we develop an Object-oriented Design Pattern to address the safety attribute in the system?

Q2: Why we should use the statecharts semantics in modeling safety-critical software?

Q3: How the variability of product line addressed in the statechart?

This section presents a new design pattern — Safety-driven design pattern. This pattern extends capabilities of both the Statecharts design patterns and Safety patterns. This pattern allows an object to alter its behavior and change its internal state when there is a safety violation, and to protect it from introducing in unsafe states.

In this part, we show how to solve recurring design problems in implementing safety-based statechart specification of an entity in an object-oriented application mainly safety-critical application.

A statechart is an alternative means of system specification. This specification methodology is particularly oriented to "reactive systems" — that is, systems that respond to a series of events rather than transforming an input into an output (Niaz and Tanaka, 2003). Such systems may incorporate concurrent processing, and statecharts encompass this capability. The OO methodologies using statecharts describe in sufficient detail the steps to be followed for describing the behavior of objects (Niaz and Tanaka, 2003). In most OO methodologies, when people think of object behavior, they consider the functionality of the object. But in many real world applications this definition is insufficient — the internal state of an object and the quality attributes should also be considered (e.g. the safety in safety-critical systems). In other words, because the Unified Modeling Language (UML) statechart diagram is a powerful tool for specifying the dynamic behavior of reactive objects, we can use this facility to describe the system behavior in term of safety.

Because it is very efficiency to address the safety of the system considering the changing of the system from a state to another state, so we can monitor and control the safety of the system in each state. Also the dynamic modeling – in the same time we can model the system with dynamic modeling. Many systems, such as real-time systems, are highly state-dependent; that is, their actions depend not only on their inputs but also on what has previously happened in the system.

We have developed a new safety-driven design pattern. This work on safety pattern presented in this thesis is based on the works in (Yacoub, 1998; Armoush, 2010; Gomaa, 2011).

This design pattern support the design process in context of software architectures considering one of the software quality attribute which is safety attribute. This pattern extends capabilities of statechart design patterns to include the quality attribute (Safety). This pattern addresses the safety in the behavior of the objects to improve the safety of applications. The main idea of this new pattern is combining the concepts of the traditional safety patterns with the concepts of statechart patterns. The result is a new object-oriented design pattern which handles the safety attribute.

There are several benefits of this pattern, for example: the ability of this pattern to capture the dynamic nature of the safety attribute, so, by this pattern we can monitor and control the safety of the system in each state. Another important is the ability to capture the variability existing in the software product lines as it is an object-oriented design pattern. Also support the maintainability and performance attributes.

In the following sub sections (Section 5.3.1, 5.3.2) we will give a short description of the pattern and the idea of the statechart pattern extension to capture the variability in the SPLs.

5.3.1 Pattern Description

Pattern Name:

Safety-driven Design Pattern of Statechart.

Type:

Software Pattern.

Intent:

Allow an object to alter its behavior when there is a safety violation and enter into a safety state to take necessary actions.

Context:

Your system is a safety-critical system and contains an entity whose behavior depends on its state. You need to develop fault-tolerant software for a highly safety-critical system. The considered system has at least one fail-safe state, or an additional safety processing module has to be used to overtake necessary actions when there is a safety violation.

Problem:

The problem is how to address the safety in the system design in terms of Object-oriented design.

Solution:

Our solution is to combine the concept of a traditional safety patterns with the concepts of satechart patterns to develop a safety-driven design pattern (refer to the references (Yacoub, 1998)(Armoush, 2010)).

Using this pattern requires developing a safety-based statechart to specify the entity's behavior without safety violation. Figure 5.1 below describes the structure of the solution in term of statechart diagram (general structure of the pattern).

Define a *Virtual superstate* called *Operating state*. There are two concurrent regions of *Operating composite* state. In the other words, the *Operating* state is a collection of the two orthogonal state which process same events. The first one is *System_Operation* state which represents the normal operations of the system to do its functionality and if there is a safety problem the system changes to the *Safe_Mode* state. And the next one is a *System_Safety_Operation* state which includes the behavior of the system to handle the safety. In the composite *System_Safety_Operation* state initially the system is in the *monitoring* state that to monitor the safety in the system. If there is any safety violation the system changes to *Processing_Safety_Operation* state to overtake necessary actions to protect the system to enter in unsafe state or to handle the safety.

Pattern Architecture

Figure 5.2 shows the design solution structure in UML notation. Distinguish the events, conditions, actions, and entry and exit procedures in each state class.

A virtual class called "*Operating*" is created, which contains the two superstates "*System_Operation*" and "*System_Safety_Operation*", events received by the interface is dispatched to the " *Operating* " class which dispatches them to both the "*System_Operation*" and "*System_Safety_Operation*" classes.

As we see in Figure 5.2, *Operating* class becomes the context for the two concurrent regions *System_Operation* and *System_Safety_Operation*. *System_Operation* and *System_Safety_Operation* classes provide the interface for the two concurrent regions. *Safe_Mode*, *System_Functioning*, *Monitoring*, and *Processing_Safety_Operation* become the concrete substate or *IntermediateSuperState* classes for *Operating composite state*. The *Operating object* will keep the references of the current active substate within each concurrent region

in *System_Operation* and *System_Safety_Operation objects*. The concrete state objects will maintain two references for the two contexts *Operating* and *Entity_Interface*.

The *Entity_Interface* (context) object delegates all incoming events to its current state object (state). On receiving an event in the *Operating state*, the *Operating object* will delegate the request to the current active substate. The active substate will execute the corresponding action on the transition and then change the substate by calling the appropriate set substate method of the *Operating object*. On receiving a fault event from the *System_Operation* state, the operating object will delegate the request to the *safe_Mode substate* and *Processing_Safety_Operation substate* in *System_Safety_Operation* composite state. If the target of a transition is the composite state then it is executed by the composite state but if the target is the substate then the composite state object will delegate the request to the active substate. The active substate will execute the corresponding action on the transition and then executes the exit action and then sets the next substate by calling the *setSub() method* of the composite Operating object.

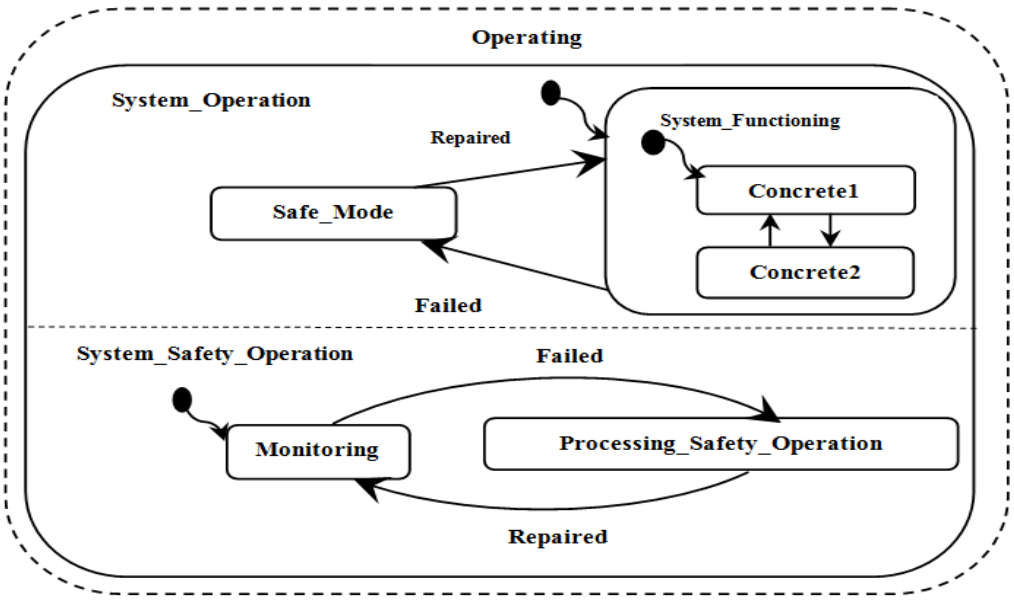


Figure 5.1: Safety-driven design pattern of statechart -the structure of the solution in term of statechart diagram

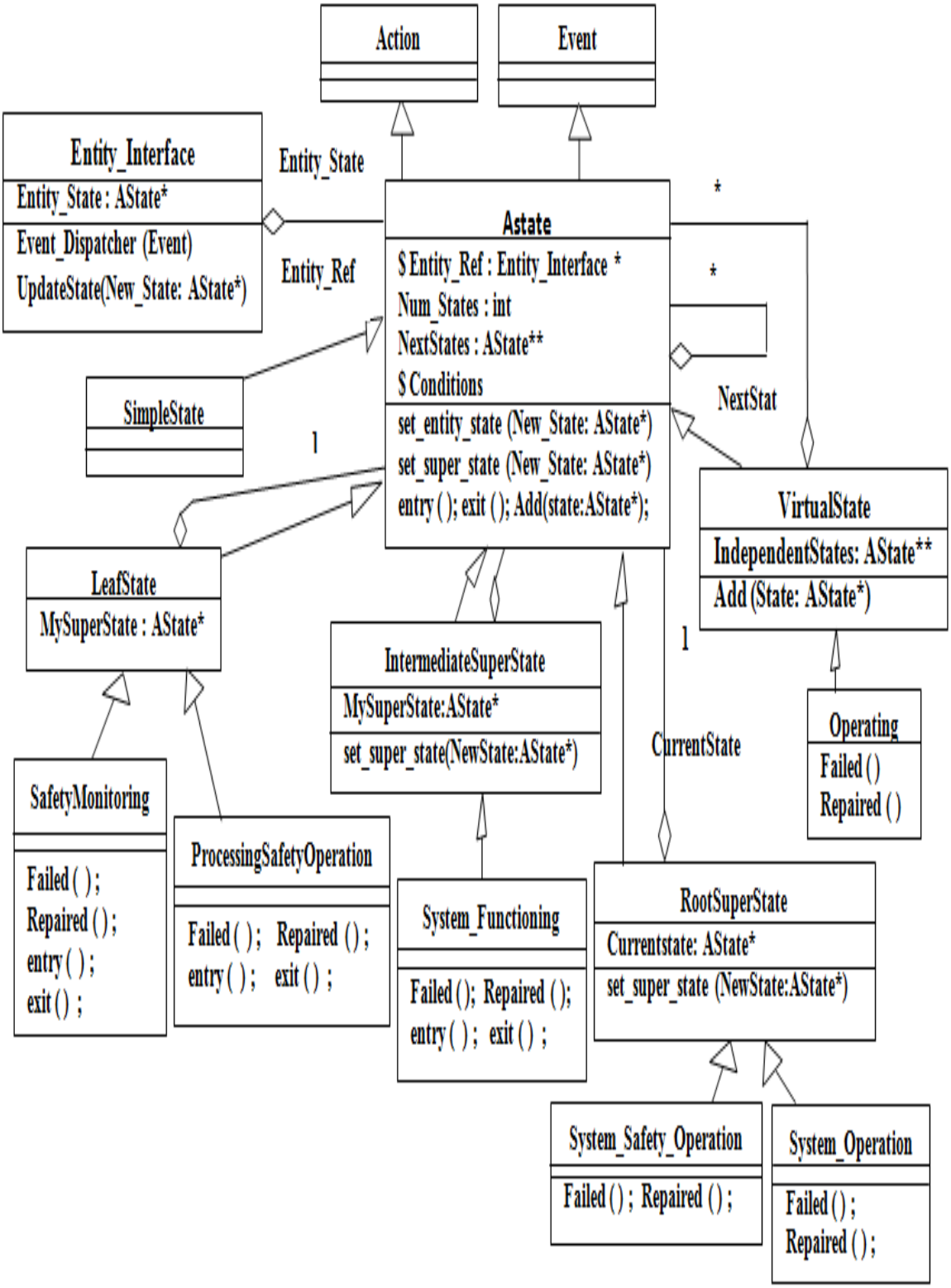


Figure 5.2: Safety-driven design pattern of statechart-the design solution structure in UML notation

5.3.2 The Statechart Pattern Extension to Capture the Variability in the SPLs.

As we mentioned in the previous chapter that the critical issue of using the statechart modeling for the software product line is how to address the variability. The works in (Gomaa, 2011) (Gomaa, 2004) proposed a method to address this issue. It shows that there are two main approaches we can use when variable classes are developed, specialization or parameterization. In the case when there are small number of changes to be made then we can select specialization as we can manage the specialized classes (Gomaa, 2011) (Gomaa, 2004). Based on this mentioned work and because the complexities exist in the PL we propose using of parameterization approach. It is obvious that it is more effective to design a parameterized state machine, in which there are feature-dependent states, events, and transitions. Optional state transitions are specified by having an event qualified by a Boolean feature condition, which guards entry into the state. Optional actions are also guarded by a Boolean feature condition, which is set to true if the feature is selected and False if the feature is not selected for a given SPL member, for more detail see reference (Gomaa, 2011) (Gomaa, 2004).

5.4 Chapter Summary

Chapter 4 mentioned that the key aspect of the proposed SSPLA design method is the use of the design patterns concept which improves the design process. A Safety-driven design pattern of statecharts has been presented in this chapter. The pattern extends capabilities of both the statecharts design patterns and safety patterns. This point constitutes one of the main contributions of this thesis.

This chapter also described the description of the developed pattern, Subsection 5.3.1, and gave a brief overview of the idea of the statechart pattern extension to capture the variability in the software product lines, 5.3.2. The next chapter (Chapter 6) describes our solution of how to address the influence of the security issues on the safety pattern. In Chapter 8, two application examples of two case studies are presented to illustrate the effect of the design pattern in the PLA design and to illustrate how all the research works can improve the safety design of the SPLAs.

CHAPTER VI

6

ENHANCING THE SAFETY DESIGN TO ADDRESS THE SECURITY ISSUES USING PATTERNs

6.1 Introduction

At the last decades, technological developments have enabled to be taken classic systems place by automatic and advanced systems (Karthika, Rahamtula and Anusha, 2018). Cyber-physical systems (CPS) and Internet of Things (IoT) have distinct origins but overlapping definitions and both combine the word embedded systems (Burns, 2019). These systems contain computational (software), communication and physical components. However such systems are at least partially controlled by software. The software has a major role and responsibility in such systems. The Difficult design problems are often assumed to be readily solved using software; and the software must compensate for any deficiencies in hardware platforms (Sommerville, 2009), (Bures T. et al, 2017). One of the critical steps in software development process is software architecture design (Sommerville, 2009), (Li, Safe and Université, 2018). Currently, the influence of architecture in assurance of software safety is being increasingly recognized.

As the high quality, short delivery time, and high productivity have become more and more important in developing embedded software for modern products (Nagamine, Nakajima and Kuno, 2016), product-line (PL) and reusable software components are suitable approaches for these embedded systems, which are often re-engineered from existing systems (Nagamine, Nakajima and Kuno, 2016). Developing a reference architecture which represents the base structure of the member products is the main task of the software product line architecture design (Systems, 2000). It is evidence that a pattern based development of the reference architecture can support the development and application process of product lines.

Most of the modern systems are considered to be safety-critical when failure events can lead to human lives losses or high valued asset losses (Vaccare Braga *et al.*, 2012). The security of these modern systems affects its safety. For example, an attacker who can send custom commands or modify the software of the system may change its behavior and send it into various unsafe situations (Li, Safe and Université, 2018). For that it is necessary to ensure security since failures in this type of systems can lead to catastrophic results.

Due to the tight interplay between safety and security, combining safety and security in the engineering process for Cyber-physical system (CPS) has become a new interesting research topic in recent years (Schmittner *et al.*, 2015).

The pattern-based approach proposes patterns as a method of capturing expert solutions to many common software problems (Jamal and Eric, 2003). In general, most software systems cannot be structured according to a single architectural pattern and they must support several system requirements that can only be addressed by different architectural patterns (e.g. design for flexibility and adaptability) (Buschmann and Maunier, 2001). Therefore, you must combine several patterns to structure such systems. The selection of an architectural pattern, or a combination of several, is only the first step when designing the architecture of a software system (Buschmann and Maunier, 2001). Pattern composition has been shown as a challenge to applying design patterns in real software systems (Hallstrom, Soundarajan and Tyler, 1995).

To the best of our knowledge, there are various pattern approaches in the context of patterns composition, specifically at the architectural design level. For example, In (Amorim *et al.*, 2017) their pattern-based approach is to provide guidance with respect to selection and combination of both types of patterns in context of system engineering, specifically, at the architectural design level. However, these approaches are limited at patterns development level, especially, for safety and security patterns development because this area still considered as an emergent research area.

In this thesis and in order to enhance the safety patterns to address the influence of the security issues on the safety, a pattern development approach that interlinks safety and security patterns is proposed, see Sec. 6.3. This approach is considered as one of the main contributions of this thesis.

The approach is then used to enhance the proposed safety design pattern of statechart (presented in Chapter 5) to address the security in the pattern (see Sec. 6.4). The work is highly supports the object-oriented software architectures development for the product lines.

The remainder of this chapter is organized as follows. Section 6.2 provides a short overview of some of the most related works. Section 6.3 presents a description of the proposed pattern development approach for safety and security. Using this approach to develop a safety and security patterns is presented in Section 6.4 which shows how the approach can be used to enhance a safety design pattern of statechart to address the security in the pattern. And finally, Section 6.5 concludes the chapter and identifies future research directions.

6.2 Related Works

Multiple studies and surveys showed that today's systems are vulnerable to security threats which can adversely affect the safety (Schmittner *et al.*, 2015). Due to the tight interplay between safety and security, combining safety and security in the engineering process for cyber-physical system (CPS) has become a new interesting research topic in recent years (Schmittner *et al.*, 2015). In the past, the safety and security communities developed quite differently and almost independently, and the resulting standards, guidelines and methods were also limited to safety or security (Schmittner *et al.*, 2015). In the context of software safety the software architecture is where the basic safety strategy is developed in the software.

Design patterns can be used to enhance the design of systems in different application domains. Design patterns are popular in the field of software engineering and there are plenty of patterns (Rauhamäki, Vepsäläinen and Kuikka, 2012). Contradictorily, in the field of control, safety and security engineering patterns have not been studied and published in such volumes. Therefore, we have now answered the call.

Patterns are normally not invented but discovered (Rauhamäki, Vepsäläinen and Kuikka, 2012). Someone realizes that a recurring problem was solved the same way several times. This already is the pattern, the recurring solution to the recurring

problem. It can then be named, formalized and documented in order to make reuse possible (Rauhamäki, Vepsäläinen and Kuikka, 2012).

Ensuring safety and security of complex integrated systems requires a coordinated approaches that involve different stakeholder groups going beyond safety and security experts and system developers. The authors in (Raspotnig, Karpati and Opdahl, 2018) have therefore proposed CHASSIS (Combined Harm Assessment of Safety and Security for Information Systems), a method for collaborative determination of requirements for safe and secure systems.

The work in (Schmittner *et al.*, 2015) investigates an integral part of the safety & security co-engineering approach called safety & security co-analysis, which aims to identify and analyze safety and security risk in a holistic approach. They focus on the methods that enable the assessment of safety effects from security threats and vice versa.

In (Amorim *et al.*, 2017) they argue that there is lack of experience with security concerns in context of safety engineering in general and in automotive safety departments in particular. To remediate this problem, they propose a pattern-based approach that provides guidance with respect to selection and combination of both types of patterns in context of system engineering. However this work focuses on the systems development engineering specifically, for developing a safety-critical systems with respect to the influence of security issues and not for patterns development. Therefore, our work is to use the experience at the pattern design level which adds more generality and spread the using of the experience.

In the other side of our work, the statechart patterns, MOODS is one well-known work on Object-Oriented state machines which presents an alternative technique of selecting the most optimal design among different state machine patterns (Ran, 1995). Yacoub in (Yacoub, 1998) has attempted to combine multiple state machine design patterns into a cohesive unit. His work is based on the concept of statecharts developed by Harel. His work is considered as a first attempting in this context. In (Adamczyk, 2003) The work shows how different design patterns solve different problems given a specific context and a set of expectations (e.g. flexibility of design, loose coupling between elements, performance, etc). The uniform format of presentation aims to help software designers select the FSM (Finite State Machine) most appropriate for their needs.

To the best of our knowledge, there are various pattern approaches in the context of patterns composition, specifically at the architectural design level. In comparison to our work, none of the aforementioned approaches show clearly the pattern composition process to develop the patterns. Pattern composition is considered as a challenge. In the other words, these approaches are limited at patterns development level, especially, for safety and security patterns development because this area still considered as an emergent research area. Our work is differ from the other existing works as it focuses on enhancing safety pattern itself in context of security issues to develop a new version of the pattern.

In the context of the safety patterns, especially the safety patterns of statechart, it evidence that there is no explicit consideration of the influence of the security on the safety which is the most important thing. Therefore, in this thesis and in order to enhance the existing safety patterns or even develop new ones, to address the influence of the security issues on the safety, a pattern development approach that interlinks safety and security patterns is proposed.

6.3 The Proposed Pattern Development Approach for Safety with Security Control

In this chapter we define a pattern development approach that interlinks safety and security patterns in order to enhance the safety patterns to address the influence of the security on the safety. This proposed approach is considered as a one of the main contributions of this thesis. It mainly focuses on enhancing safety pattern itself in context of security to develop a new version of the pattern. The enhanced safety pattern is considered as a new safety and security pattern.

The pattern approach uses the selected combination of safety and security patterns to effectively improve the safety pattern (s). It provides a new way of thinking for safety patterns development with respect to security. The approach supports the variability of product line in the point of selecting of an appropriate security pattern (s). Also by using this pattern approach we can develop a new safety security patterns.

Apart from the systematic interlinking of safety and security patterns, we elaborate how these patterns can be combined in order to enhance the safety patterns.

In the subsection (Sec. 6.3.1) bellow and for this approach we propose a safety and security pattern engineering lifecycle that aims at combining the two engineering processes for safety pattern development and allows for the necessary interaction and focusing on safety with respect to the influence of security issues.

In order to show the applicability of this approach we have used it to enhance the proposed statechart-based safety pattern and we have developed a new safety and security pattern.

6.3.1 A Proposed Pattern Development Engineering Lifecycle

The Pattern Development Engineering Lifecycle is a development model of the pattern development approach which defined to help engineers developing effective safety patterns that address the safety with presence of security issues influence the safety.

In general the lifecycle contains three main processes: Safety engineering process which comes before Security Engineering, the second process. The rationale for this is that the approach explicitly focuses on “security for safety” (i.e., safety concerns are the main engineering drivers). However, security measures can influence pattern architecture properties that can alter safety. For this reason, the Safety and Security Co-Engineering Loop is introduce, the third process of the lifecycle.

6.3.2 The process steps

We proposed a process model that guides the overall process. Figure 6.1 bellow shows the model steps. Notice that the development process is done in iterative manner. Each step contains some engineering activities. The input of this model is the description of the safety pattern that needs to improve, or to use to develop a new safety and security pattern.

Step 1: Safety and Security Pattern Co-analysis:

The main purposes of the safety and security co-analysis step are: to deeply understand and analyze the safety pattern under consideration in order to identify the

security issues which cause safety risks to the pattern. And based on this we have to find an appropriate security pattern(s) that can be used to interlink with the safety pattern to achieve the safety requirements for the safety pattern considering the security problems.

This step includes two main activities. The first one is the safety and security co-analysis which includes safety pattern analysis process and security patterns analysis for the safety pattern. The other main activity is a definition and selection of a suitable security pattern(s).

In step 1 and after determine the specific safety pattern that needs to improve, we analyze this pattern. This step is one of the important steps in the pattern development process and all the next stages are depend on the success of this step. The analysis also includes definition of how the security factors affect the safety. After these two activities and at the end of this step a suitable security pattern(s) is defined. This appropriate security pattern is used latter to deal with the influencing of the security on the safety. After completion of this step we move to step 2 (the design phase).

Step 2: Safety and Security Pattern Co-design

There are two main activities in this step; Applying/Instantiating the security pattern(s) and Designing and modeling the new version of the safety pattern. The construction of the new version of the safety pattern is done in this step. The two engineering aspects, safety and security are considered in this step.

The specific safety pattern and the selected security pattern defined in the previous step are combined to create the initial architecture of new version of the safety pattern or to refine the existing one. In this step also, the architecture's elements or components and their relationships are defined, that for the produced pattern. That results in either an initial architecture model (initial version) or the final one. This process is continues until the completion of the pattern architecture. Also in this step the architecture descriptions of the produced pattern are defined.

Step 3: Evaluate and Assess the Produced Version of the Safety Pattern

This is an engineering process usually conducted by using assessment or evaluation methods.

In this step, and in general, the resulted pattern architecture is checked with respect to functional and quality requirements. In addition, the pattern architecture resulting from the previous step is evaluated according to the architecture evaluation plan which is based on safety and security. Here, in this process model, evaluating architecture is based on test cases which based on safety and security, if the safety and security requirements are met or not.

We can distinct three cases might be happened in the evaluation:

1. The architecture of the produced version of the safety pattern is ok. There are no problems in the resulted pattern architecture. In this case documentation for this new pattern is created.

2. The evaluation shows that there is a need to improve the pattern architecture, so we can go to step 2. This to repeat the pattern architecture building process and then it could evaluate again.

3. There is a problem(s) in the resulted pattern architecture. In this case the problem(s) will be analyzed in (step1).

In this context we developed a new safety assessment model proposed to show the relative safety improvement in the design after using a specific safety pattern, see Chapter 7. The implementation of this safety assessment model is illustrated using case studies, see Chapter 7. The results show that there is a considerable improvement in the design of the system architecture after using this pattern.

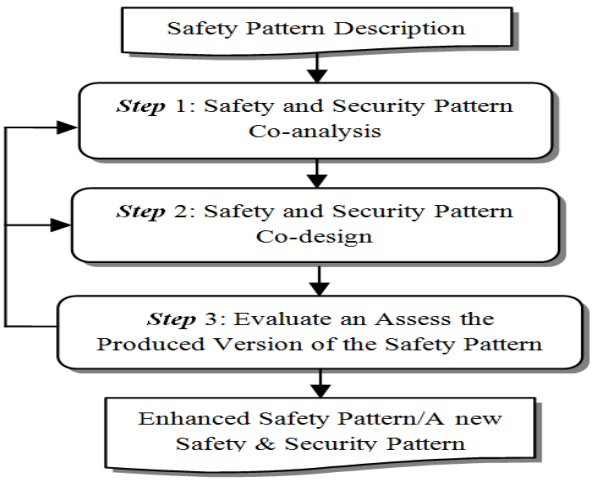


Figure 6.1: Software Architecture Pattern development model for Safety with Security Control

6.4 Developing a Safety and Security Pattern

In this section we describe how the proposed pattern development approach (mentioned in Section 6.3 above) helps to develop an effective safety and security patterns that address the safety with presence of security issues influence the safety.

The approach is applied on our proposed safety pattern of statechart mentioned in chapter 5, and it is published in (Ebnauf and Al., 2019). In the following sub sections we present in briefly way how the pattern development process is applied as well as a description of the enhanced version of the safety pattern of statechart. Note that the developed version is considered as a new safety and security pattern.

6.4.1 The Safety Design Pattern of Statechart

In this section we apply the proposed pattern development approach on our proposed safety-driven design pattern mentioned in chapter 5 (it is published in (Ebnauf and Al., 2019)).This pattern extends capabilities of both the statecharts design patterns and safety patterns. The pattern allows an object to alter its behavior and change its internal state when there is a safety violation, and to protect it from introducing in unsafe states. The critical aspect of the pattern is the ability to capture the dynamic nature of the safety attribute. By this pattern we can monitor and control the safety of the system in each state. The result is an object-oriented design pattern which handles the safety attribute. In the context of the product lines, there is an idea of the statechart pattern extension to capture the variability in the SPLs (See reference (Gomaa, 2011)).

6.4.2 Applying the Proposed Pattern Development Approach

The previous pattern (presented in Sec. 6.4.1 and published in (Ebnauf and Al., 2019)) is a general safety pattern which means that there is no consideration of the impacts of the other specific non-functional requirements. As there is a high impact of the security on the safety, especially in the smart environments (e.g. Cyber-physical Systems and Internet of Things), we need an improved safety patterns to address this issue. In this chapter and in order to achieve that, we have applied the

pattern approach mentioned above. As mentioned above, by applying this approach an enhanced version of the safety pattern under consideration is produced. The developed version is considered as a new safety and security pattern.

The following lines show the development process of a safety and security pattern using this pattern development approach.

6.4.3 The Development Process:

The following lines describe briefly how to conduct the development process with the proposed safety pattern (Sec. 6.4.1):

After analysis of the safety pattern (***Step 1: Safety and Security Pattern Co-analysis***) we observed that we need a security monitoring technique. This monitoring technique is to prevent the system against attacks and check the internal state of the system in context of security.

The security analysis shows that the Gateway Pattern (GP) (Han, Weimerskirch and Shin, 2014) is a suitable pattern to achieve the security monitoring and it is a light pattern also. Here, we proposed using the Security Gateway Pattern (SGP) with some modifications for checking the system internal state and notifications. This last point leads to the end of the step 1.

- Security Gateway Pattern

The security gateway is a security pattern that is placed between an unprotected internal network and un-trusted external entities when communication to the outside is inevitable. As a repeatable solution, the security gateway is not limited to the specific interface. In (Han, Weimerskirch and Shin, 2014) and (Schmittner *et al.*, 2015) they proposed a three-step authentication protocol that provides secure communication between the external device and the ECUs in the vehicle (See Fig. 6.2). Adding the Security Gateway (SG) as an additional component supports the maintainability of the security solution. In the other hand, updates to the gateway do not impact the safety pattern directly.

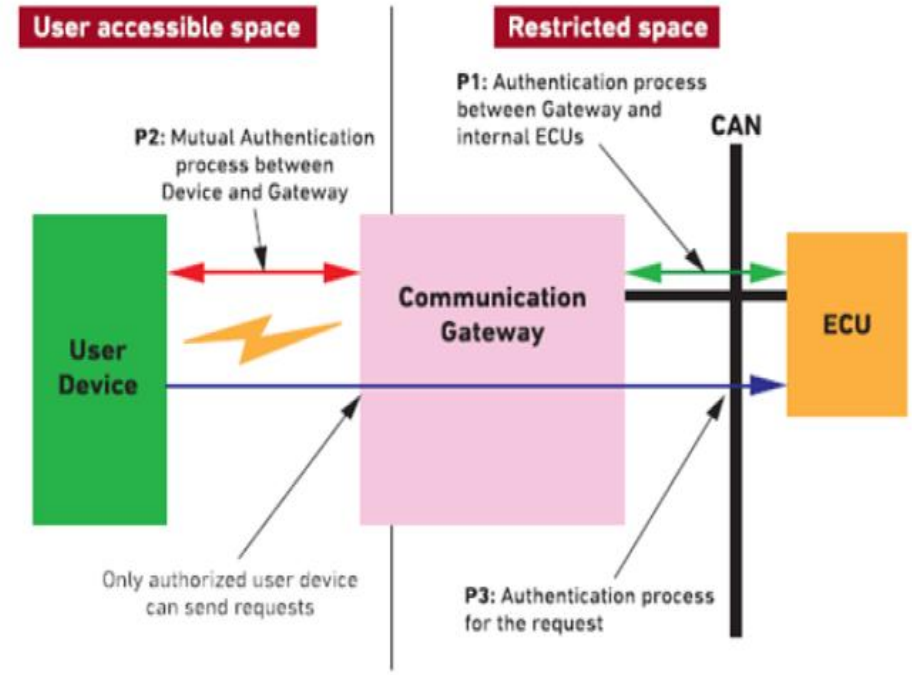


Figure 6.2: Three-step authentication for secure connection between external entities (user devices) and ECUs (CAN) (Han, Weimerskirch and Shin, 2014)

And then in ***Step 2***: (***Safety and Security Pattern Co-design***) The selected Gateway Security Pattern is instantiated and incorporated into the existing design of our safety pattern architecture. In this process we have needed to adapt the Gateway Security pattern in order to integrate it with our safety pattern (presented in Sec. 6.4.1 and published in (Ebnauf and Al., 2019)). This integration has leaded us to performing some changes and configuration on the architecture of our safety pattern as shown in Fig. 6.3. Depending on the analysis result of the safety pattern under consideration, we decided to modify the *System_Safety_Operation Super state* in the pattern structure to be a *System_SafetyandSecurity_Control Super state*. The modification is by adding *SecurityMonitoring Sub_Super state*.

Fig. 6.3 below describes the structure of the solution in term of statechart diagram (general structure of the enhanced safety pattern). This figure (Fig. 6.3) also shows the new version of the pattern after applying step 3 (Evaluate an Assess the Produced Version of the Safety Pattern). In addition, this last model is used to develop the model of the design solution structure in UML notation, see Fig. 6.4. The model describes the architectural view of the developed pattern in object-oriented design. In

the following lines we present the architectural models of the pattern and the description of the developed pattern. For more details about how to develop a UML class diagram from a statechart model, refer to the work in (Ebnauf and Al., 2019) or Sec. 5.3.1.

Finally, in ***Step 3*** and for the evaluation of the produced pattern, a safety assessment method developed in (Ebnauf and Al., 2019), Sect. 7.2, is used. As this development process is done in iterative manner, we continue until there are no problems in the resulted pattern architecture or there is no need to improve it more.

To illustrate the effect of this improved safety pattern in the PLA design, a Smart Microwave Oven product line and a simplified Automated Electromechanical Braking System (EBS) are used as running examples, see chapter 8. We used our safety assessment method presented in Sect. 7.2 to show the relative safety improvement after using the developed safety and security pattern. The results show that there is a considerable improvement in the system safety design after using the developed safety and security pattern.

6.4.4 The Developed Version of our Safety Pattern of Statechart

- Safety and Security Design Pattern of Statechart:

Other Names:

Safety and Security Pattern, Safety and Security Pattern of Statechart

Type:

Software Pattern.

Abstract:

The Safety and Security Pattern is considered as an enhanced version of the previous Safety Pattern of Statechart mentioned in Section 6.4.1 which is used to address the safety with presence of security issues influence the safety in the pattern.

A security pattern is configured and adapted to the safety pattern context and requirements that in order to integrate both safety and security patterns. The major configurations which have done in the previous version of the safety pattern are described briefly as follow:

"As shown in Fig. 6.3, the *System_Safety_Operation* supper state in the previous version (look Fig. 5.1) is configured to be *System_SafetyandSecurity_Control* supper state which includes a new sub supper state called *Security_Monitoring* state added to address the influence of security issues and to add some security control. Notice that this point explains how the instantiated security pattern is incorporated into the safety pattern. So, as we mentioned that with our safety pattern we have proposed a security monitoring technique. This monitoring technique is to prevent the system against attacks and check the internal state of the system in context of security. According to this, the proposed *Gateway Security Pattern* is used with some modifications for checking the system internal state and notifications processes. This technique with the overall safety control prevents the system against the security risks regarding to the safety and protects it to inter in an unsafe state."

Context:

Developing an effective safety architecture for the safety-critical system which addresses the safety with security control.

Problem:

The problem is how to address the safety in the system design with presence of security issues influence the safety and that in terms of Object-oriented design.

Pattern Structure:

Fig. 6.3 shows the structure of the new solution of the enhanced version in term of statechart diagram (general structure of the pattern). The model describes the architectural view of the developed pattern in object-oriented design is shown in Fig. 6.4. The model shows the design solution structure in UML notation.

The other details about the enhanced version of the safety pattern are similar to the description of the previous version, see Sec. 5.3.1.

Implication:

It is observed that the performance is a little bit improved as there is a security monitoring and notifications happen (in advance) which lead to take appropriate actions early. This last point is the other benefit of this version.

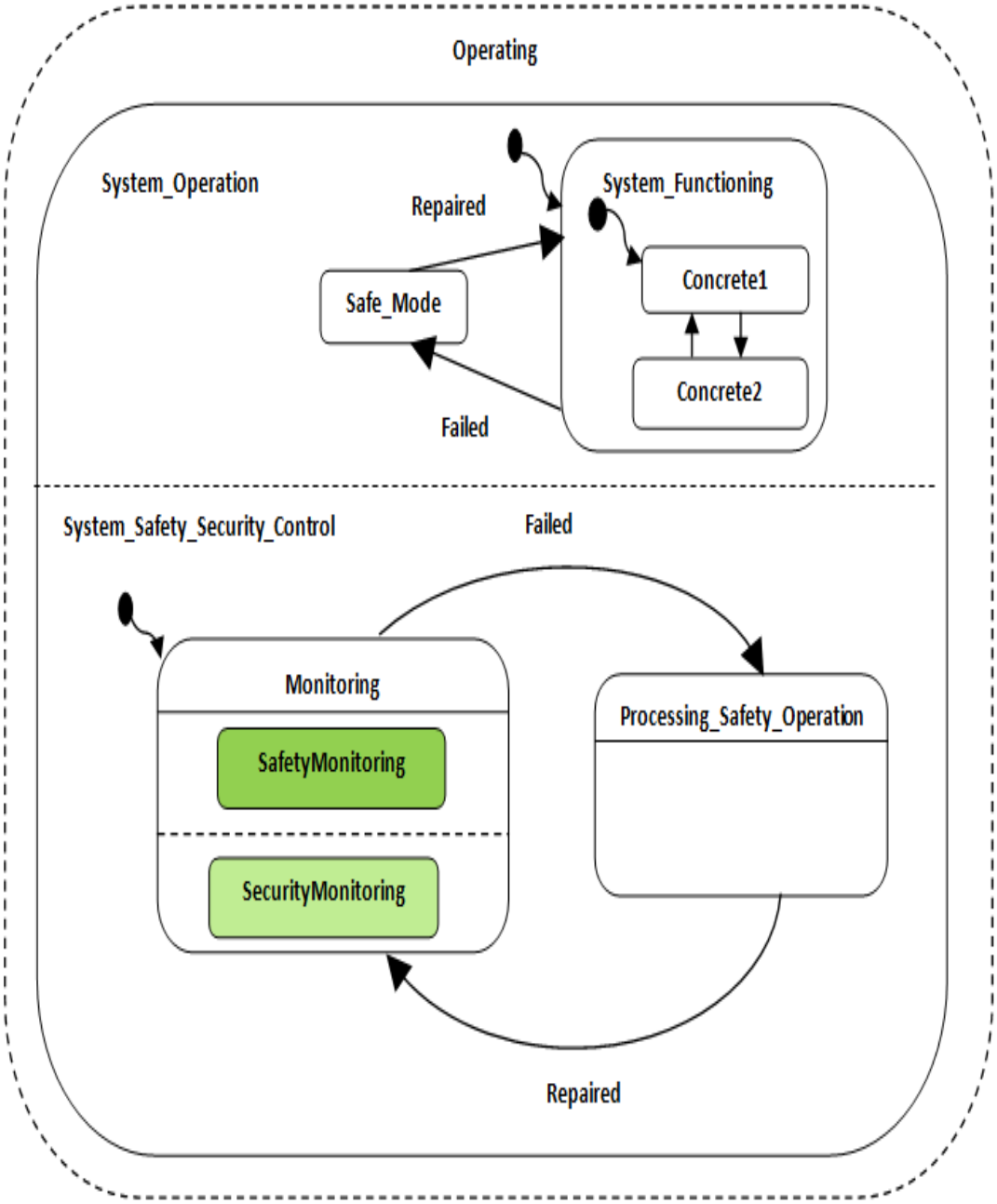


Figure 6.3: The Safety and Security Design Pattern-the structure of the solution in term of statechart diagram

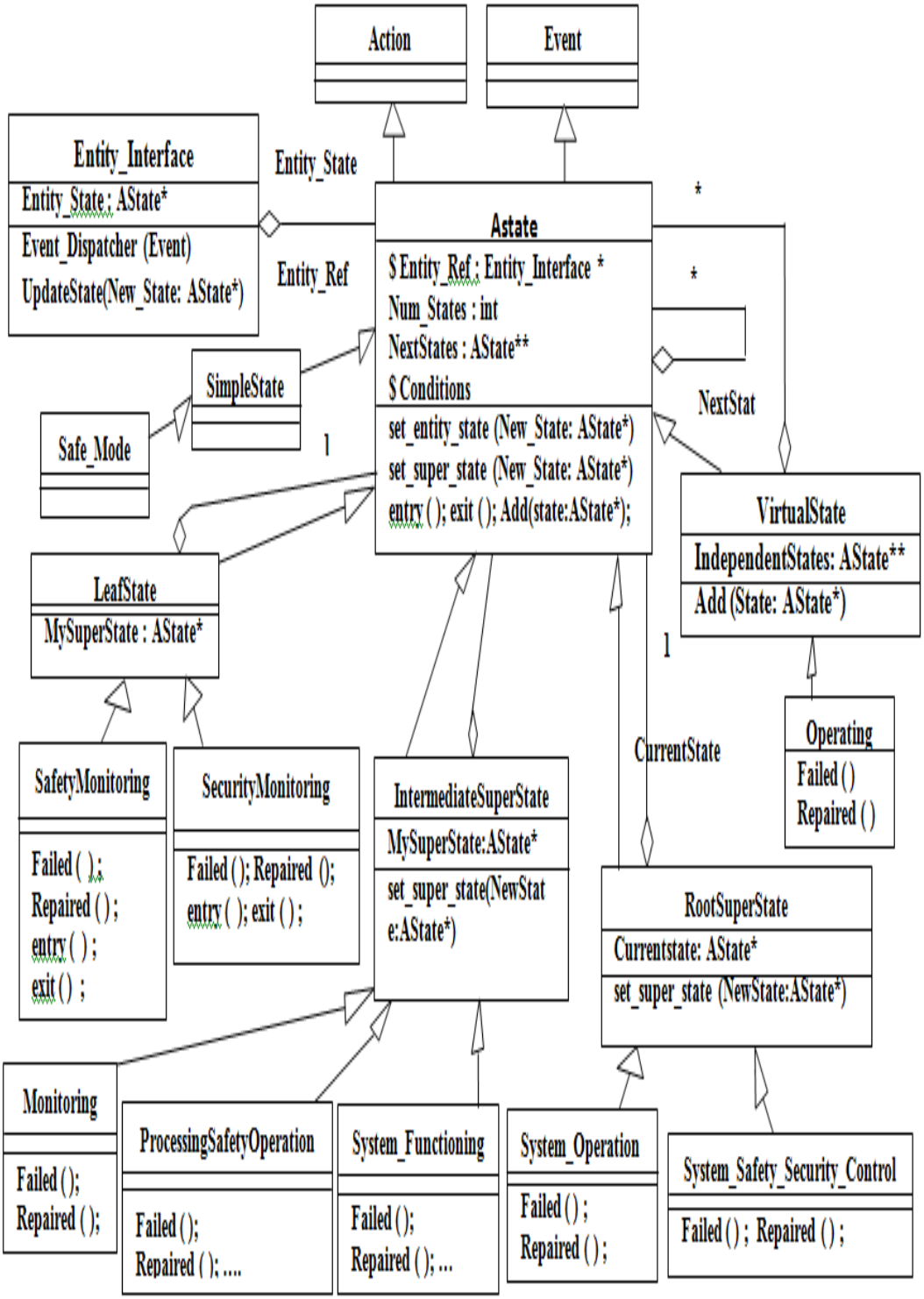


Figure 6.4: The Safety and Security Design Pattern-the design solution structure in UML notation

6.5 Chapter Summary

In this chapter, our work is concerned with the addressing of the security in the safety design patterns in order to improve the safety design using patterns, and its applicability is on the SPL architectures design. In order to address the influence of the security issues on the safety, a pattern development approach that interlinks safety and security patterns is proposed, Section 6.3. This pattern approach can be used to enhance the existing safety patterns or even develop new ones. It is evidence that the pattern approach can support the variability of product line in the point of selecting of the appropriate security pattern(s).

The pattern development approach is applied on the proposed safety pattern of statechart which described in Section 5.3. By applying the pattern approach, an enhanced version of the safety pattern is produced, Section 6.4. This version is developed to address the security issues in the safety pattern. The developed version is considered as a new safety and security pattern. The enhanced pattern version developed by this approach is called ***safety and security pattern of statechart***. Finally, to illustrate the effect of this enhanced pattern in the PLA design, two application examples of two case studies are used, Chapter 8. For future work, this research can be applied to other patterns domains.

CHAPTER VII

7

SAFETY ASSESSMENT FOR SSPLAs

"Achieving reliability and safety is hard, but what is even tougher is assessing those qualities". J M Voas (Thane, 1999).

7.1 Introduction

Some of the content in this Chapter has been presented at 7th IEEE International Conference on Mechatronics Engineering, ICOM2019, Putrajaya, Malaysia.

Safety is not a software issue; rather, it is a system issue (Place and Kang, 1993). It is an abstract concept. We mean by "the system is safe" that it will not cause harm either to people or property (Place and Kang, 1993). All parts of the system must be safe. Functional and operational safety starts at the system level. Safety cannot be assured if efforts are focused only on software (Huang, 2013). After the designers have applied measures to mitigate mishap risk to a basic system, they must determine if the modified system design meets an acceptable level of mishap safety risk (W.R. Dunn, 2002).

[John McDermid] in (McDermid, 2002) argues that the software safety analyses are a special portion of the overall system safety analyses and are not conducted in isolation. In essence there are four safety-relevant parts of a system development process (McDermid, 2002):

1) Identifying hazards and associated safety requirements.

2) Designing the system to meet its safety requirements.

3) Analyzing the system to show that it meets its safety requirements.

4) Demonstrating the safety of the system by producing a safety case.

Software safety analysis does not look at functional and safety requirements simultaneously. For example the functional specifications can modeled with a UML state diagram and other tool can used to delineate the causes of hazards like a Fault Tree Analysis (FTA). So, there is a need for an integration approach. Creating a model that shows how a system is designed to work, while also describing where issues can occur, leads to a better, safer system.

It is evidence that the definition of safety becomes related to risk. Risk may be defined as (Place and Kang, 1993):

$$Risk = \sum_{hazard} \varepsilon(hazard) \times P(hazard) \qquad (2)$$

where ε (hazard) is a measure of the effects that may be caused by a particular mishap and P(hazard) is the probability that the mishap will occur (Place and Kang, 1993).

The authors in (Goseva-Popstojanova *et al.*, 2003) developed a methodology for risk assessment of software architectures based on the Unified Modeling Language (UML). In the methodology the probability of software components/connectors failures is estimated by measuring the complexity/coupling of the UML dynamic specifications. Severity is estimated using the classical technique of Failure Mode and Effect Analysis (FMEA). While the authors in (Hassan, Goseva-Popstojanova and Ammar, 2005) propose a severity assessment methodology which is performed combining three different hazard analysis techniques: Functional Failure Analysis (FFA), Failure Mode and Effect Analysis (FMEA), and Fault Tree Analysis (FTA) (Hassan, Goseva-Popstojanova and Ammar, 2005).

Statecharts is a specification tool derived from finite-state machines (Bogdanov and Holcombe, 2001). It is a power tool to specify the dynamic behavior of the system. Due to the graphical nature and a variety of constructs, statecharts have been widely used in projects (Bogdanov and Holcombe, 2001).

As (UML) statechart diagram is a powerful tool for specifying the dynamic behavior of reactive objects, we can use this facility to describe the system behavior in term of safety.

In (Leveson *et al.*, 1991) The authors has described some of the lessons learned and issues raised while building a model using statecharts of a real aircraft collision avoidance system. Once the system requirements specification is completed, safety analysis procedures will be derived for the modeling language and evaluated using a specific test case tool called (TCAS tesbed) (Leveson *et al.*, 1991).

As such domains include safety critical systems which exhibit probabilistic behavior, there is a major need for modeling and verification approaches dealing with probabilistic aspects of systems in the presence of variabilities. The authors In (Varshosaz and Khosravi, 2013), introduce a mathematical model, Discrete Time Markov Chain Family (DTMCF), which compactly represents the probabilistic behavior of all the products in the product line.

Although a considerable number of safety analysis techniques have been proposed to aid software design such as Software Hazard Analysis and Resolution in Design (SHARD) (Fenelon *et al.*, 1994), there is little analysis work focusing on an architectural level to aid software architecture design. In the other side, while several assessment methods have been used to evaluate safety-critical systems, most of these methods cannot be used to assess safety-critical design software product lines due to the complexity and variability of these systems. We thus need a suitable safety assessment method that is able to characterize the architectural elements at different architectural levels.

The objective of this part of our thesis is to provide a safety assessment model that can be used to facilitate the assessment process of the architecture for safety-critical software product line systems and also to show the safety improvement in the design after using our work. The assessment method to be described here is particularly suited for testing an architecture design against a detailed specification.

The remainder of this chapter is organized as follows. Section 7.2 presents a proposed safety assessment model. It is a simplified mathematical model for safety assessment of the product lines architectures. Adapting this assessment model to be a scenario-based assessment method and adding a metric (or the concept) of Relative Safety Improvement RSI is presented in Section 7.3. The final step of the evaluation

process is to compute the Relative Safety Improvement (RSI), this is described in Section 7.4. The illustration of the safety assessment model using a simplified Electromechanical Break System software product line as an application example is presented in Section 7.5. And finally, Section 7.6 concludes the chapter and identifies future research directions.

7.2 The proposed Safety Assessment Model for SPLAs

As we mentioned in the above sections that the thesis aims to make the overall architecture development activities compatible and consistent. For that a new safety assessment model is proposed. This model is considered as a state-based model. The effectiveness of this model is that it is a dynamic model which means it assesses the system safety at the runtime.

The proposed *Safety Assessment Model* presented in this section is a simplified mathematical model for safety assessment of the product lines architectures which has been developed based on some of the previous approaches (Dabrowski and Hunt, 2011; Nunes *et al.*, 2012; Varshosaz and Khosravi, 2013). It is for using in *the evaluation process step* in our proposed design method (presented in Chapter 4). And it is also used to evaluate our work to show the improvement in the safety design of the product line architecture after using our work, see Chapter 8. The following sub sections mention and describe the model steps and the mathematical calculation in the model. Figure 7.1 bellow shows the safety assessment model steps.

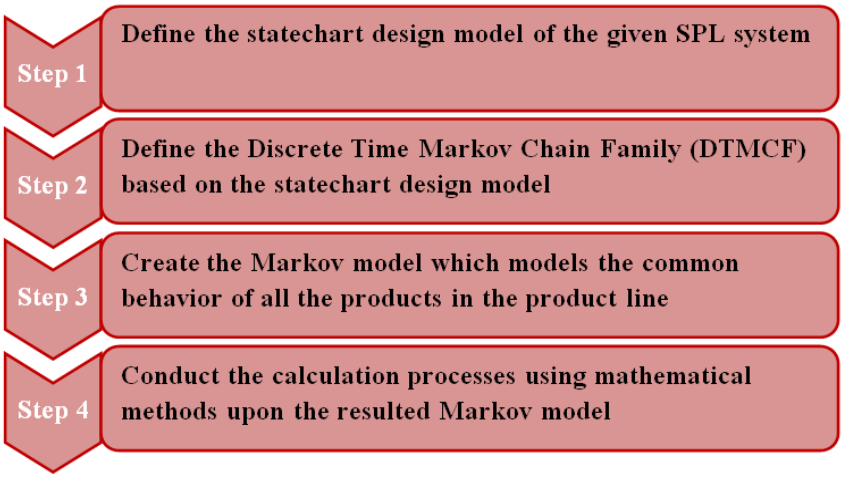


Figure 7.1: The Safety Assessment Model Steps.

7.2.1 The Model Steps:

Step 1: Define the statechart design model of the given software product line system.

Step 2: Define the Discrete Time Markov Model Family (DTMCF) based on the statechart design model of the PLA which is defined in the previous step. The resulted DTMCF compactly represents the probabilistic behavior of all the products in the product line.

Step 3: By using a variability-aware approach which takes into account the common behavior between products in a SPL, create the Markov model which models the common behavior of all the products in the product line.

Step 4: Conduct the calculation processes of the safety evaluation using mathematical methods upon the resulted Markov model.

7.2.2 The Mathematical Calculations of the assessment process

In this solution, the calculation is based on some hypothesis and solutions of other works on Markov chain. We use the Discrete Time Markov Chain (DTMC) theory (Kassir, 2018) to efficiently analyze behavior and safety of the critical systems in term of software system architecture. The safety assessment method used in this work is distinguishable from the well-known use of DTMCs to provide quantitative measures of system performance and reliability, which we review in (Dabrowski and Hunt, 2011). Instead of measuring system reliability, we use DTMCs to examine safety in dynamic systems in order to identify the probability of system being in safe execution or in unsafe state.

7.2.2.1 Specifying a Markov Chain Process

State based formalisms are a more powerful alternative to combinatorial formalisms. Markov chains (MCs) is one of the most common methods. Markov chains are effective tools that used for evaluating the safety and reliability of architectures (Varshosaz and Khosravi, 2013).

After defining the Markov chains then it can be evaluated regarding the probabilities that the system is in a certain state at time *t*. In this thesis we used the steady state evaluation technique which calculates the probabilities for $t \rightarrow \infty$. So the

reliability and safety can be calculated by summing up the probabilities of the reliable respective safe states. One of the possible methods to do the analysis for both the transient and the steady state is Monte Carlo (Dabrowski and Hunt, 2011). In this thesis we used the normal mathematical calculation of the steady state. So the results are not exact even with using Monte Carlo simulation since the accuracy depends on the number of simulation runs.

7.2.2.2 Maximum Likelihood Estimation for Markov Chains (MLE)

The critical point of this solution is the method of define or estimating the transition probabilities between the states in the Markov model and form the transitions probabilities matrix TPM . So we proposed a derived technique extracted from the previous works (Varshosaz and Khosravi, 2013) and (Dabrowski and Hunt, 2011) called (Maximum Likelihood Estimation for Markov Chains (MLE)). The following lines conclude the idea:

By using simulation technique, execute the Markov model for a period of time. And then we observe the execution to compute the number of transition between the states. State transition probabilities were derived as follows. Given states s_i, s_j, i,j= 1…n where n= the number of the system states, p_{ij}, is the probability of transitioning from state i to state j, written as $s_i \rightarrow s_j$. This probability is estimated by calculating the frequency of $s_i \rightarrow s_j$, or f_{ij}, and dividing by the sum of the frequencies of s_i to all other states s_k, as shown in equation (3)

$$p_{ij} = \frac{f_{ij}}{\sum_{k=1}^{n} f_{ik}} \qquad (3)$$

Here i and j may be equal, to allow for self transitions, which are counted if the task process remained in a state longer than a discrete time step.

In this thesis and to simplify the process, we suppose that the frequencies of the transitions from any state i to any of its adjacent states (states connected directly) are equals. By this suggestion the probability is estimated by dividing the number of direct links state i connect directly to state j (N_{ij}) to the total of all direct transition links from state i to other adjacent states $\sum_j N_{ij}$. By this suggestion we can derive

equation 4. This can lead to a low accuracy, but for more accuracy we can use a simulation technique.

$$p_{ij} = \frac{N_{ij}}{\sum_{j} N_{ij}} \qquad (4)$$

7.3 A Scenarios-based Assessment

As we mentioned in the previous chapters that a number of efforts have been made to make the method effective and efficient. One of the critical efforts is making the design process activities of the method compatible and consistent (e.g. state-based design, scenario-based design and assessment). So, this section describes how the safety assessment model (described in Section 7.2 above) can be adapted and configured to be a scenario-based assessment model.

To accomplish the evaluation process of the safety and security pattern (our enhanced pattern version) using two application examples of the two case studies (see Chapter 8), we have adapted and configured the safety assessment model mentioned above (Sec. 7.2) to be a scenario-based assessment method as well as adding the metric or the concept of Relative Safety Improvement RSI). We have used the safety and security risk scenarios of the two software product lines, Smart Microwave Oven and the simplified Automated Electromechanical Braking System (See Chapter 8).

The safety assessment process model presented in Section 7.2 is a simplified mathematical model for safety assessment of the product lines architectures. The following lines mention and describe in briefly way the major steps of the model after adapting it to be a scenario-based assessment model:

- *Define the statechart models for each defined scenario in the given software product line system.*

- *Developing the Markov chain models before and after using the proposed pattern for each scenario- These models can elicit by using the statechart model defined in the above step.*

- *Conduct the calculation processes of the safety assessment process using mathematical methods upon the resulted Markov model. This step includes Creation of the Transition Probability Matrix for each Markov model.*

Markov chains are effective tools that used for evaluating the safety and reliability of architectures (Varshosaz and Khosravi, 2013). After defining the Markov chains then they will be evaluated regarding the probabilities that the system is in a certain state at time *t*. As mentioned in Sec. 7.2.2.1 that in this thesis we used the steady state evaluation technique which calculates the probabilities for $t \rightarrow \infty$.

Note that in our work and for both examples (presented in Chapter 8), we applied this assessment process by using the kernel system, an advanced product of the Software Product Lines as well as the individual risk scenarios. In this chapter and in Section 7.5 we applied the assessment model using the kernel system of the EBS product line as illustrative example of the model. In Chapter 8 we will present in more details the implementation and evaluation of our work, with the using of individual risk scenarios.

7.4 Defining the Final Results- The Relative Safety Improvement

The final step of our evaluation process is to compute the Relative Safety Improvement (RSI). The Relative Safety Improvement (RSI) is a safety assessment metric proposed by Ashraf in (Armoush, 2010) which gives an indication about the safety improvement that can be achieved by the pattern. This metric (RSI) is defined as "the percentage improvement in safety (reduction in probability of unsafe failure) relative to the maximum possible improvement which can be achieved when the probability of unsafe failure is reduced to the minimum possible value (0)". Based on this definition, the relative safety improvement for a design pattern (or system design) can be expressed as shown in the following equation (Equation 1) (Armoush, 2010):

$$\left[RSI = \left(1 - \frac{P_{UF(new)}}{P_{UF(old)}}\right) \times 100\% \right] \qquad (1)$$

- *RSI: Relative Safety Improvement.*
- *PUF(old): Probability of unsafe failure in the basic system.*
- *PUF(new): Probability of unsafe failure in the design pattern.*

We assume that the probability of the system to be in unsafe state is equivalent to the probability of the unsafe failure, by considering that all failures are unsafe failures. By this assumption we do the calculation of the RSI for all the individual scenarios in the two examples of the two case studies, see Chapter 8 (Section 8.4.3, Fig. 8.21 and Fig. 8.22).

A scenario-based assessment model can be used for assessing the quality attributes which are safety and security. In this context the produced scenarios are based on the safety and security requirements. The evaluation process includes analysis of how well software architecture satisfies safety and security requirements. In Chapter 8, the sections (Sec. 8.4.1 and 8.4.2) present the analysis results of the individual risk scenarios for the two case studies. These results are shown in Fig. 8.17 and Fig. 8.20 (for example 1 and example 2 respectively). For all the specified scenarios of the two examples, the Fig. 8.17 and Fig. 8.20 show a comparison between the probabilities of the systems to be in the Unsafe states, and that before and after using the proposed pattern in the case of executing the systems for a long time.

7.5 Illustrative Example

To illustrate the applicability of the safety assessment model for product line architecture, we have chosen a very simplified automated Electromechanical Braking System (EBS) product line (Varshosaz and Khosravi, 2013), as an example. In this section we applied this assessment process by using the kernel system of the EBS product line.

7.5.1 The EBS SPL System

The EBS system is considered one of the subsystems of Cruise Control System (C.C). The main function of EBS system is to automatically stop the car or the

vehicle and in safely way when there is an obstacle in front of the vehicle. In such braking systems, sensors, communication media, and actuators replace mechanical devices (Varshosaz and Khosravi, 2013). In reality, there could be malfunctions with nonzero probabilities. For example, failure of the sensors to detect obstacles in an admissible interval, possible message loss, and (Control Unit) CU failure are some examples of undesirable but possible characteristics of such systems.

From applying our proposed design method (presented in Chapter 4), the dynamic analysis step processes results in dynamic models such as the communication and statechart models. Figures 7.1 and 7.2 are two statechart design models of the EBS SPL system which show the system design structure without safety control, and after using the new safety design pattern respectively. For more details see Chapter 8, the application example 1 of the first case study.

The next section (Sec. 7.5.2) presents the implementation of the safety assessment model using the EBS product line mentioned above. The result shows the improvement in the product line design after using the proposed safety design pattern.

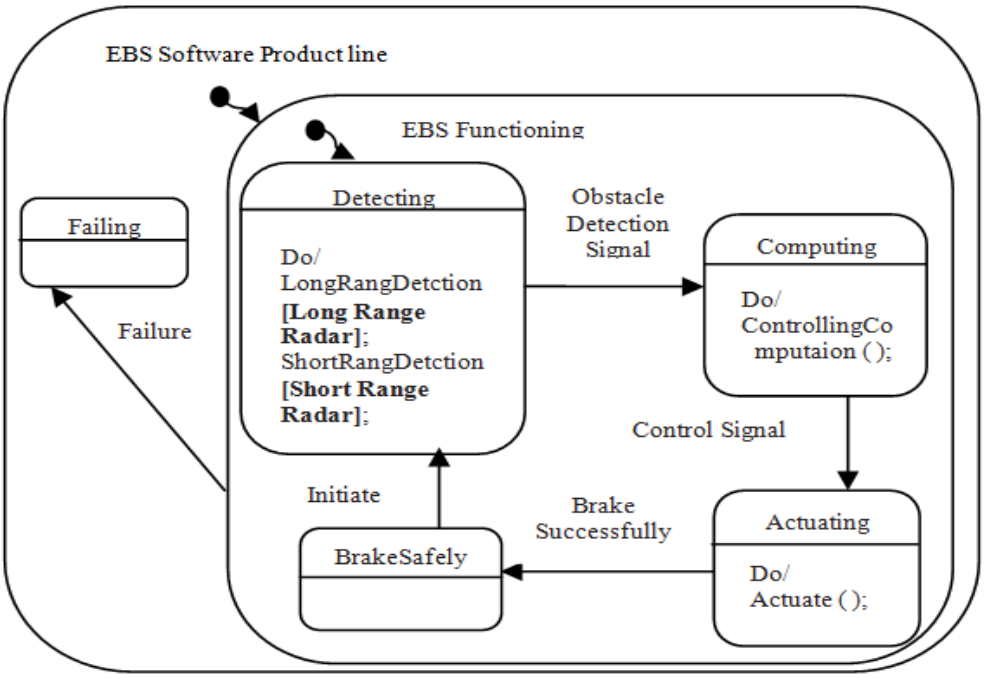


Figure 7.2: The Statechart Design Model of Automated Electromechanical Braking System (EBS) Software Product Line after Using the Pattern.

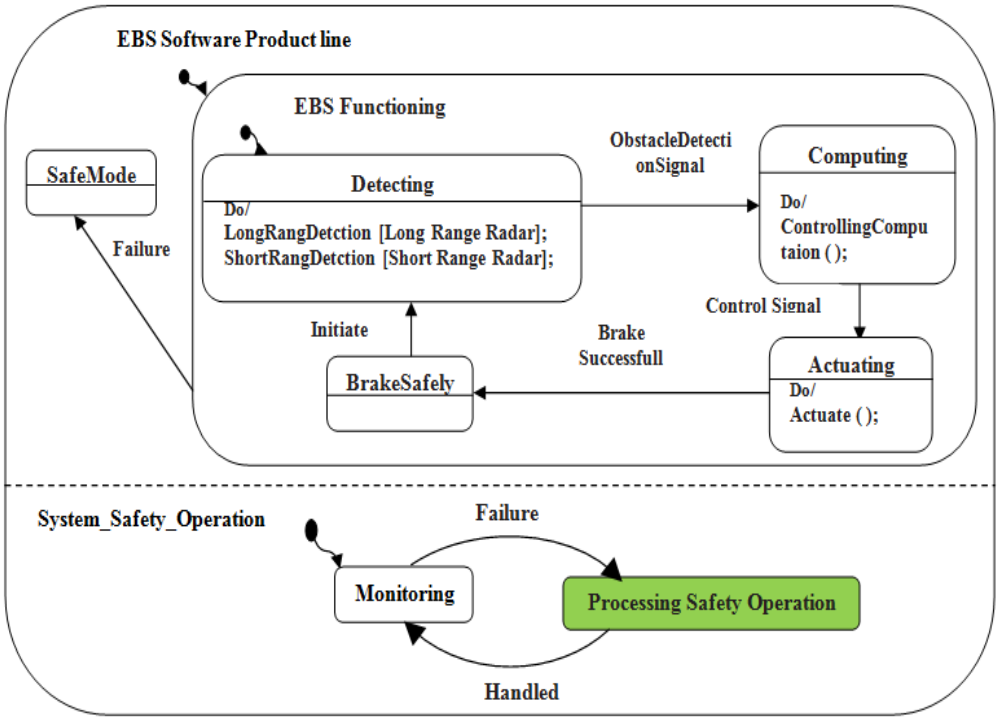


Figure 7.3: The Statechart Design Model of Automated Electromechanical Braking System (EBS) Software Product Line after Using the Pattern.

7.5.2 The implementation of the Assessment Model

This section presents the implementation of our proposed safety assessment model that mentioned above. To illustrate that we apply the assessment model on a simplified automated Electromechanical Braking System (EBS) product line. The section also shows briefly the safety improvement on the product line system after using our proposed safety design pattern presented in Chapter 5. Figure 7.3 below is a compact Discrete Time Markov Chain Family (DTMCF) of the EBS product line, designed normally without safety control and before using the proposed safety design pattern. The compact Discrete Time Markov Chain Family of the EBS product line after using the proposed safety design pattern is shown in Figure 7.4. Figure 7.5 shows the Markov model of the common behavior (kernel system) of EBS product line without using the new safety design pattern. The Markov model of the common behavior (kernel system) of EBS product line after using the proposed safety design pattern is shown in Fig. 7.6.

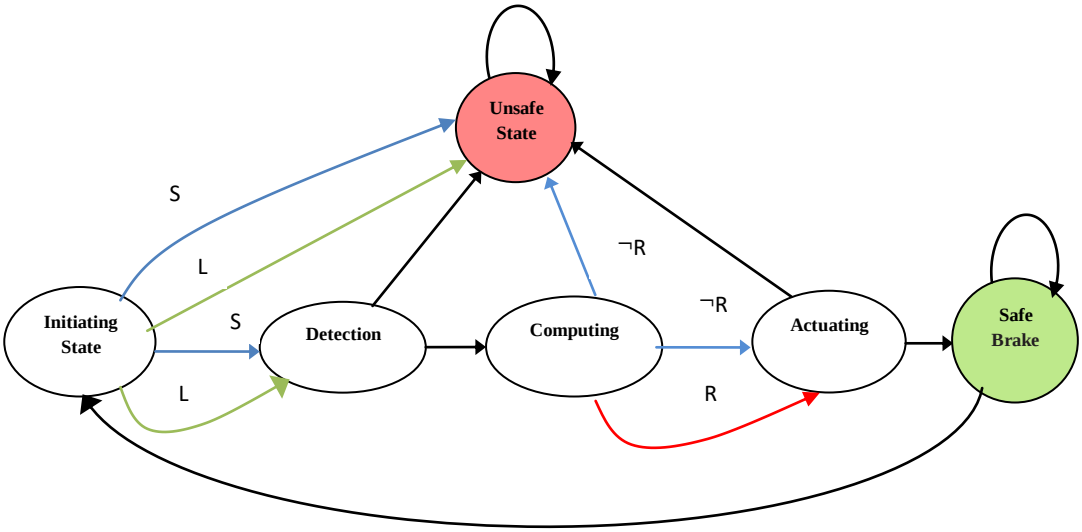


Figure 7.4: A compact Discrete Time Markov Chain Family of the EBS product line, designed normally without safety control and before using our safety design pattern.

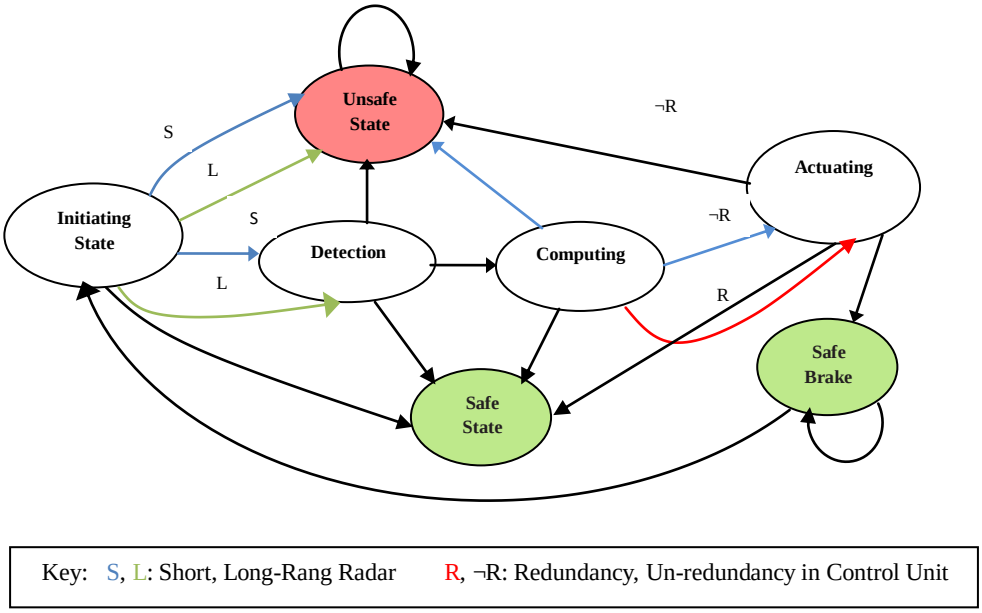


Figure 7.5: A compact Discrete Time Markov Chain Family of the EBS product line, designed with safety control or after using our safety design pattern.

7.5.3 The calculations of the Safety Assessment Process

We used above equation (Equation 3) to calculate the transition probabilities on the Markov chain. And then we created the transition probability matrix p. Other Markov processes are used like steady state operation. Finally we calculate the probability of each state for the system states before and after using the proposed design pattern. And finally we will observe the calculation results and then write the assessment results.

***Firstly**: The calculation results of the system safety design before using the proposed pattern*

Here we need to calculate the probability of each state for the system states before using the proposed design pattern. The process is conducted in the steps as follows:

Step 1: Calculate and define the transitions probabilities for the given Markov chain. We use the maximum likelihood estimation method (MLE) mentioned above to do that, See Fig. 7.5. We can use the states abbreviations in Table 7.1.

Table 7.1: The abbreviations of the Markov states

The State	Initial State	Detection state	Computing state	Actuating state	Safe Brake	Unsafe State	Safe State
Abbreviations	S_0	S_1	S_2	S_3	S_4	S_5	S_6

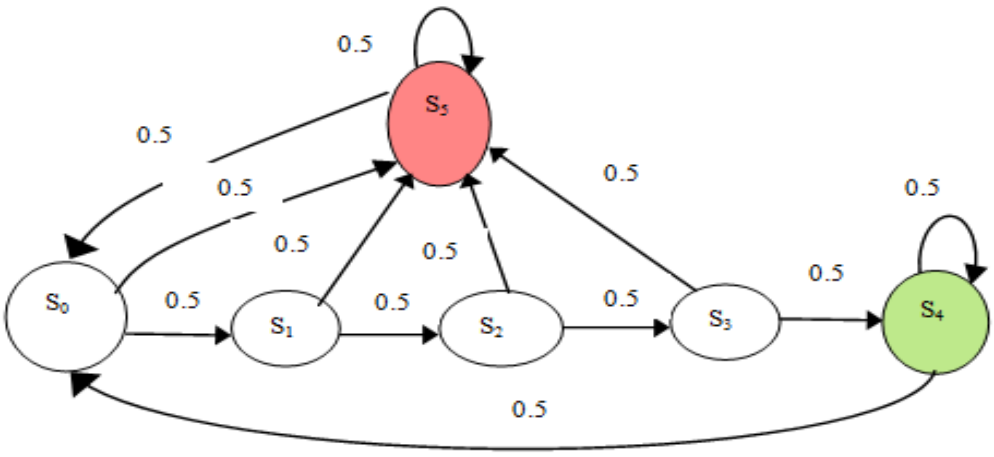


Figure 7.6: The Markov model of the common behavior (kernel system) of EBS product line without using the proposed safety design pattern.

Step 2: Conduct the transitions probability matrix TPM. Below is the transition probabilities matrix P.

$$P = \begin{array}{c} \\ S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \end{array} \begin{array}{c} \begin{array}{cccccc} S_0 & S_1 & S_2 & S_3 & S_4 & S_5 \end{array} \\ \begin{pmatrix} 0 & 0.5 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0 & 0 & 0.5 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0.5 \end{pmatrix} \end{array}$$

Step 3: Define the Markov process. By using the concept steady state we define the probability of each state of the Markov model which present the abstract state of the system.

Step 4: Solving the given Markov chain to obtain the steady state probability vector.

$$[S_0 \quad S_1 \quad S_2 \quad S_3 \quad S_4 \quad S_5] \cdot \begin{array}{c} \\ S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \end{array} \begin{array}{c} \begin{array}{cccccc} S_0 & S_1 & S_2 & S_3 & S_4 & S_5 \end{array} \\ \begin{pmatrix} 0 & 0.5 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0 & 0 & 0.5 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0.5 \end{pmatrix} \end{array} = [S_0 \quad S_1 \quad S_2 \quad S_3 \quad S_4 \quad S_5]$$

So:

$0.5\ S_4 + 0.5\ S_5 = S_0$ ---------- (1)

$0.5\ S_0 = S_1$ ---------- (2)

$0.5\ S_1 = S_2$ ---------- (3)

$0.5\ S_2 = S_3$ ---------- (4)

$0.5\ S_3 + 0.5\ S_4 = S_4$ ---------- (5)

$0.5\ S_0 + 0.5\ S_1 + 0.5\ S_2 + 0.5\ S_3 + 0.5\ S_5 = S_5$ ---------- (6)

$S_0 + S_1 + S_2 + S_3 + S_4 + S_5 = 1$ ---------- (7)

After solving the resulted equations system, the steady state probabilities obtained as follows:

$S_0 = 0.258$, $S_1 = 0.129$, $S_2 = 0.065$, $S_3 = 0.033$, $S_4 = 0.033$, and $S_5 = 0.484$

That shows that the probability the Unsafe State is significant at 0.484 and the system safety is not achieved in this design.

***Secondly**: The calculation results of the system safety design after using the proposed pattern*

we calculate the probability of each state for the system states after using the proposed design pattern. Calculate and define the transitions probabilities for the given Markov chain, see Fig. 7.6. We use the maximum likelihood estimation method (MLE) as before.

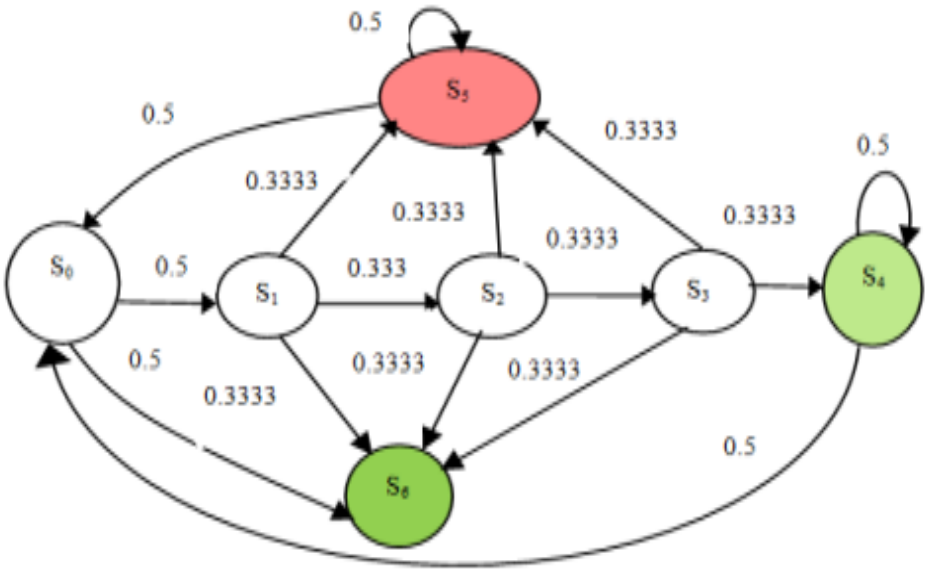


Figure 7.7: The Markov model of the common behavior (kernel system) of EBS product line after using the proposed safety design pattern

Solving the given Markov chain (Fig. 7.6) to obtain the steady state probability vector.

The new results are shown as follows:

$S_0 = 0.2686$, $S_1 = 0.1343$, $S_2 = 0.045$, $S_3 = 0.015$, $S_4 = 0.0025$, $S_5 = 0.266$ and $S_6 = 0.2686$

7.5.4 Results Discussion:

The results above show that the probability of the system being in unsafe state before using the pattern is 0.484, and after using the safety pattern is 0.266. So, the different between these two values of the probabilities is 0.218, which means, approximately, the improvement on the safety is more than or equal 21%. Finally, and with considering the relative safety improvement factor it obvious there is a considerable improvement in the safety of the system after using the safety pattern.

7.6 Chapter Summary

In this chapter, a new Safety Assessment Model (SAM) is presented. This model is considered as a state-based safety assessment model. It is a simplified mathematical model for safety assessment of the product lines architectures. Adapting this assessment model to be a scenario-based assessment method and adding a metric or the concept of Relative Safety Improvement (RSI) are also presented in this chapter.

This model has been proposed to use in the evaluation process step in the proposed design method (SSPLA) (presented in Chapter 4). It is also used to evaluate our research. In addition, the model is used to show the improvement in the safety design of the product line architecture before and after applying our work, see Chapter 8.

Finally, the chapter described the illustration of the safety assessment model using a simplified Electromechanical Break System software product line as an application example. For future work, further research is needed to assess other product line issues such as the complexity and reusability.

CHAPTER VIII

8

IMPLEMENTATION AND EVALUATION

8.1 Introduction

In the last decades, technological developments have enabled to be taken classic systems place by automatic and advanced systems (Karthika, Rahamtula and Anusha, 2018). The term cyber-physical systems (CPSs) refers to a new generation of smart systems with integrated computational and physical capabilities that can interact with humans through many new modalities (Sadiku *et al.*, 2017). The Internet of Things (IoT) and Smart Environments (SE) have attracted a lot of research and development activities during the last decade (Ray, 2018). In this context we aim at controlling physical entities through the Internet of Things.

As shown in multiple studies and surveys (Burns, 2019) (Li, Safe and Université, 2018) (Surkovi, 2018) (Ray, 2018) today's systems are vulnerable to security threats which can adversely affect the safety. Since the nature of CPSs is the interaction with the physical world, so they must operate dependably, safely, securely, and efficiently and in real-time.

We motivate our approach with the help of two application examples of two case studies. In this chapter and in order to show the applicability of this work the developed safety and security design pattern (Sec. 6.4) is applied on the design process of two software product line architectures, a simplified Automated Electromechanical Braking Systems product line and the Smart Microwave Oven Control Systems Software Product Line. Using our proposed safety and security pattern requires developing a statechart to specify the entity's behavior without safety and security violation. Therefore, it is efficient to use a state-based architectural

design approach in the overall SPLAs design lifecycle. For that the proposed state-driven architecture design method for safety–critical software product lines (Sec. 4.4) is used in the architectural design process of these two SPLs. The two examples are presented to illustrate how this work can improve the safety design of the SPLAs with the influence of the security issues.

Furthermore, to evaluate our work, a simplified safety assessment model proposed in Chapter 7 is applied on the running examples. The results show that there is a considerable improvement in the system safety design after using our proposed method and by using the proposed safety and security design pattern, as described in the following sub sections.

The remainder of this chapter is organized into four main sections. The following successive sections (Sec. 8.2 and Sec. 8.3) illustrate the applicability of our work with the help of two application examples of two case studies. Section 8.4, describes the evaluation process using the application examples as well as the final results and discussion. Finally, Section 8.5 summarizes the chapter.

8.2 The Case Study 1

In order to show the applicability of our work as well as evaluate it we have used a simplified automated Electromechanical Braking Systems (EBS) Software Product Line as a first case study.

The EBS system is considered as one of the subsystems of Cruise Control System (C.C). The main function of EBS system is to automatically stop the car or the vehicle and in safely way when there is an obstacle in front of the vehicle. In such braking systems, sensors, communication media, and actuators replace mechanical devices (Varshosaz and Khosravi, 2013). The common functions (Kernel features) that the perfect kernel system in this SPL generally does are: (Detection) the first function is the detection of an obstacle which can be done alternatively by the short range or long range radar sensors. The range of detection obviously differs in these two kinds of sensors. (Computing) When an obstacle is detected, a signal will be sent to the CU via communication media. After receiving the signal, the CU computes some required parameters aiming to control the speed of the vehicle and braking safely. (Actuating) Then, the CU sends the necessary commands to the actuators via

communication media which eventually result in the required actuation. This is the functionality in the case that every component performs perfectly. As known in reality, there could be malfunctions with nonzero probabilities. For example, failure of the sensors to detect obstacles in an admissible interval, possible message loss, and CU failure are some examples of undesirable but possible characteristics of such systems.

In the following sub sections we will present some important details of the case study specifically, how to implement our work. The main UML development models that have been developed through the case study are also presented. The main details include the safety and security issues, presenting of some individual risk scenarios, as well as presenting of some development models produced through the design process such as the statechart models and the statechart design model in UML class diagram. The statechart design model in UML class diagram describes the architecture of EBS software product line.

8.2.1 The EBS SPL Architecture Development Life Cycle

This section is to generally illustrate the implementation of the adapted design method for the SPLAs (Chapter 4) and the Safety & Security design pattern (Chapter 6). It also gives a general understanding of the implementation process. The following lines describe the development steps and show the main models with a short description for each one.

8.2.1.1 The Method Process

As we mentioned in Section 4.3.2 that the method is based on a hierarchical system model. It is a process for creation and evaluation of product line architectures. The inputs of this process are the requirements or requirement specifications. It will be possible to define two types of the general requirements which are the domain model and scope definition (see Sec. 4.3.2).

8.2.1.2 The Inputs of the Process:

As we mentioned in Section 4.3, that the input of our method are the requirements or requirement specifications. It will be possible to define two types of the general requirements which are the domain model and scope definition. In general, we can say that the inputs of the process are domain model and scope definition of the product line (for more details see Section. 4.3).

- Domain Model:

The domain model defines the main requirements of the specific applications domain or family of products. It is a requirements model. These requirements model is later used to define the scenarios (scenarios of usages).

- Scope Definition:

The scope definition defines the commonality and variability of the deferent applications in product line. There are internal variable features between the components of the applications itself (Internal Variabilities) and others are variability features with respect to the environment (External Variabilities). The scope shows the organization for the types of the developed products and for the others that will have been developed in future.

In the following lines, the section briefly presents a descriptions of the requirements engineering, dynamic modeling (Dynamic Analysis results) for the EBS PL System using statechart modeling and some other models:

1) The Requirements Modeling:

A. Commonality and Variability Analysis

The Commonality and Variability Analysis (CVA) of a product line provides a requirements specification for the product line. The CVA consists of the terminology used, the commonalities, the variabilities, and the dependencies among the variabilities. The dependencies are constraints that the choice of one features places on the choices of other features (Feng and Lutz, 2005). Note that we exclude any non-behavioral commonalities and variabilities to focus on the software. The CVA serves as a requirement specification for the product line and as an input to the

product line's architecture design. In this section we show the use case model and features model.

For single systems, use case modeling is the primary vehicle for describing software functional requirements. For SPLs, feature modeling is an additional important part of requirements modeling. The strength of feature modeling is in differentiating between the functionality provided by the different family members of the product line in terms of common functionality, optional functionality, and alternative functionality (Gomaa, 2011).

B. Use Case Modeling for the EBS SPL

For a single system, all use cases are required. In a SPL, only some of the use cases, which are referred to as kernel use cases, are required by all members of the family. Other use cases are optional, in that they are required by some but not all members of the family. Some use cases might be alternatives to each other (i.e., different versions of the use case are required by different members of the family).

In UML, the use cases are labeled with the stereotype «kernel», «optional» or «alternative» (Gomaa, 2011). In addition, variability can be inserted into a use case through variation points, which specify locations in the use case where variability can be introduced (Gomaa, 2011) (Gomaa, 2004).

The kernel and optional product line use cases for the EBS SPL are given in Figure 8.1.

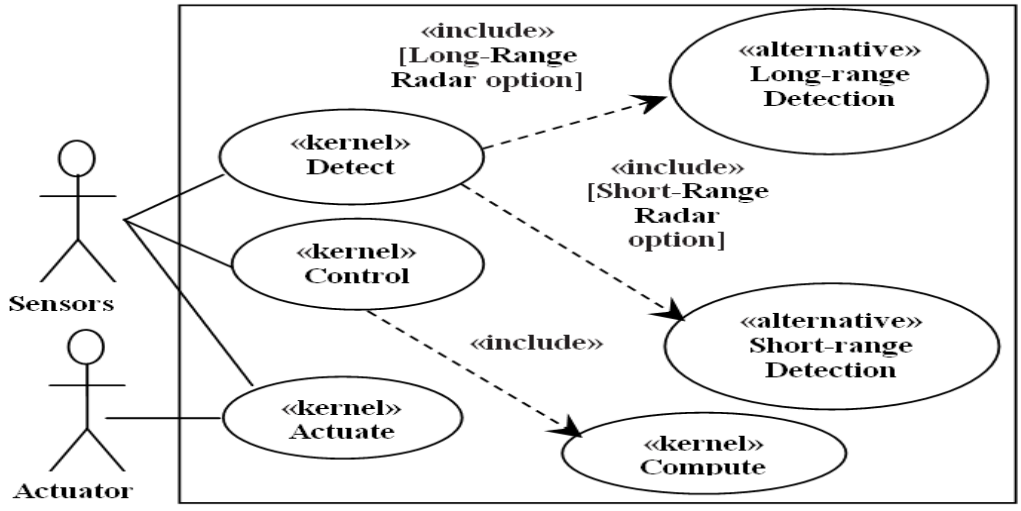


Figure 8.1: EBS Product Line Use Cases Model

Variation points are provided for both the kernel and optional use cases. One variation point concerns the methods of detection: which can be done alternatively by the short range or long range radar sensors. The range of detection obviously differs in these two kinds of sensors. This variation point is of type *mandatory alternative,* which means that a selection among the alternative choices must be made.

……

Variation point in Detect use case:

Name: Type of Detection Method.

Type of functionality: Mandatory alternative.

Description of functionality: the first function is the detection of an obstacle which can be done alternatively by the short range or long range radar sensors. The range of detection obviously differs in these two kinds of sensors.

C. Feature Modeling

The feature model is used to model the product line requirements in addition to the use case model. Feature modeling is an important modeling view for product line engineering, because it addresses SPL variability. Features are incorporated into UML in the PLUS method using the meta-class concept, in which features are modeled using the UML static modeling notation and given stereotypes to differentiate between «common feature», «optional feature», and «alternative feature» [23]. Feature dependencies are depicted as associations with the name requires (Gomaa, 2011)(Gomaa, 2004).

Fig. 8.2 below shows the feature mode of EBS Software Product Line. Table 8.1 presents the Feature/Use Case Dependencies.

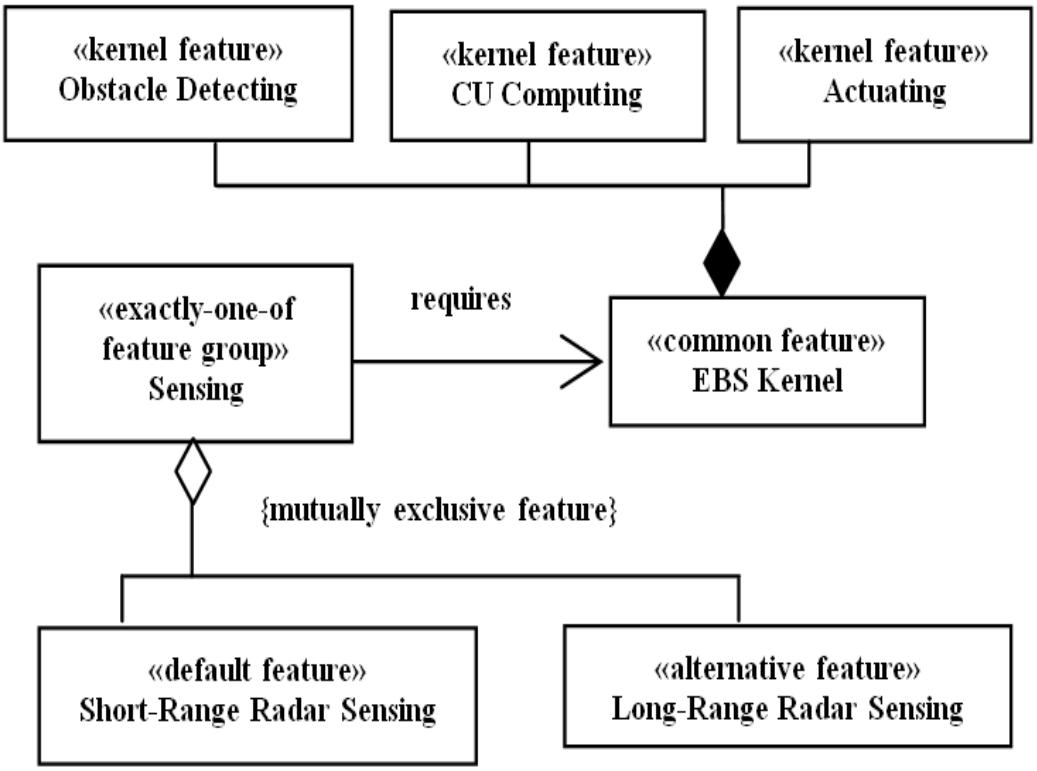


Figure 8.2: The feature model of EBS product line Using UML Notations

Table 8.1: Feature/Use Case Dependencies

Feature Name	Feature Category	Use Case Name	Use Case Category/Variation Point (VP)	Variation Point Name
EBS Kernel	Common	Detect	Kernel	
		Control	Kernel	
		Compute	Kernel	
		Actuate	Kernel	
Short-Range Radar Sensing	Default	All Use Cases	VP	Range Radar Sensing
Long-Range Radar Sensing	Alternative	All Use Cases	VP	Range Radar Sensing

2. Dynamic Modeling for the EBS PL System-Using Statechart Modeling

The dynamic analysis and design processes result in dynamic models such as the communication and statechart models. Using the proposed pattern require developing a safety-based statechart to specify the entity's behavior without safety violation.

Figure 8.3 is a statechart model of the EBS SPL system which shows the general structure view of the system design without safety control (before using our proposed pattern).

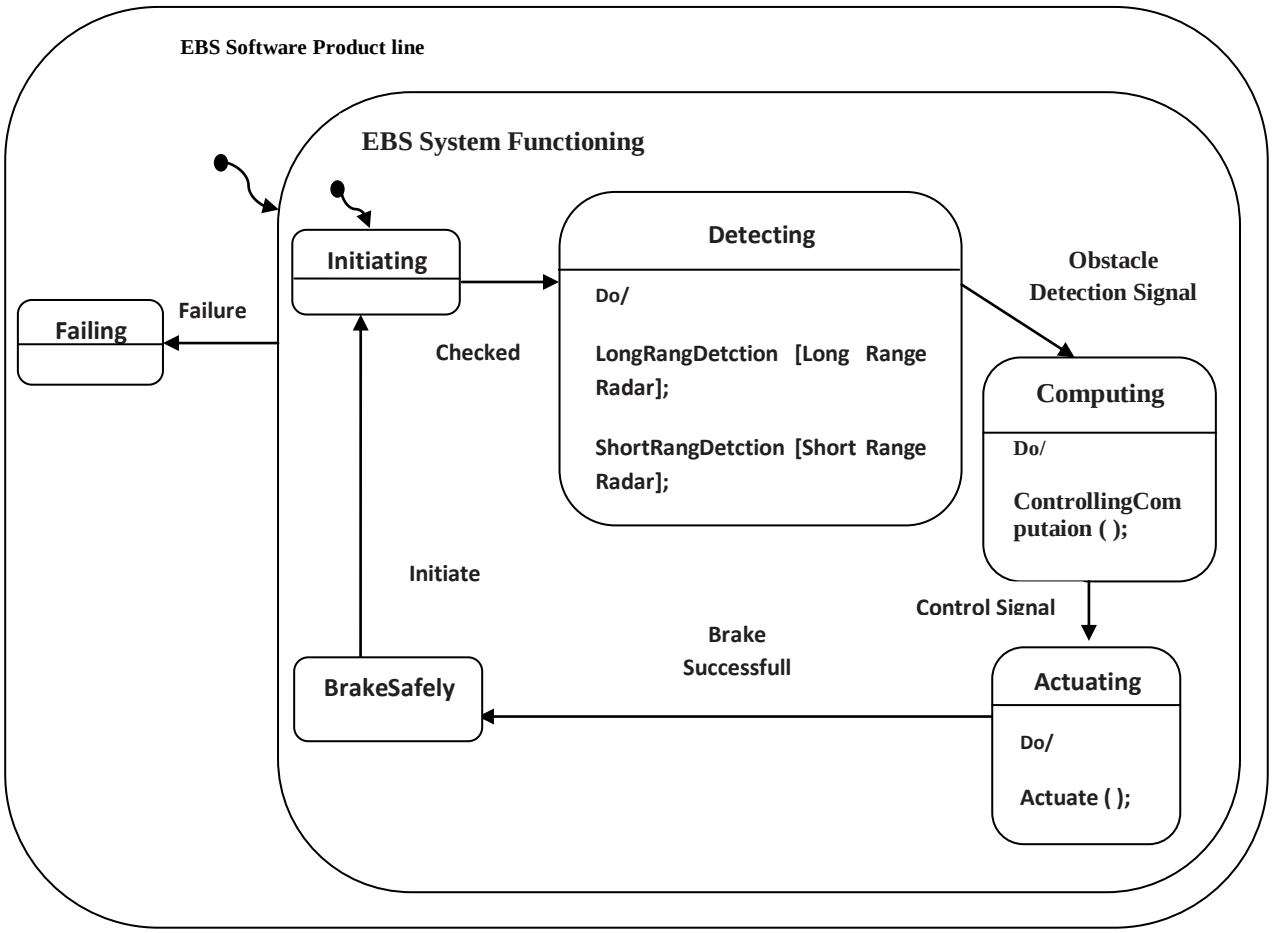


Figure 8.3: The Statechart Model of the EBS SPL before using safety control/without using our proposed safety and security design pattern.

All the data above are considered to be an input to our methodology. Below, we describe how each step of the architecture design method is applied in the study.

8.2.1.3 Applying the Process Steps

The application of each step of the proposed architectural design method is described in the following lines and sections.

Step 1: Safety requirements elicitation and analysis

In step 1: (Safety requirements elicitation and analysis), we can use any safety analysis method(s) for product line to do the step of safety analysis. In this example we used the "Bi-Directional Safety Analysis of Product Lines" proposed by Qian Feng and Robyn R. Lutz (Feng and Lutz, 2005). It is a sufficient methodology and a bi-directional in that it combines a forward analysis (from failure modes to effects) with a backward analysis (from hazards to contributing causes). The methodology for software safety analysis of a product line proposed here uses the Extended Commonality and Variability Analysis (XCA) and a hazards list to drive the bi-directional safety analysis (Feng and Lutz, 2005). Findings from application of the bi-directional safety-analysis method included new safety-related software requirements both for all the systems in the product line (commonalities) and for only some of the product-line systems (variabilities), as well as discovery of a new hazard (s).

8.2.1.3.A Safety and Security Issues

As mentioned above that since the nature of CPSs is the interaction with the physical world, so they must operate dependably, safely, securely, and efficiently and in real-time.

In the EBS SPL system case study, and as known in reality, there could be malfunctions with nonzero probabilities. For example, failure of the sensors to detect obstacles in an admissible interval, possible message loss, and CU failure are some examples of undesirable but possible characteristics of such systems. The attackers could remotely hack into the dashboard system of the vehicle, using software vulnerability. The dashboard system is also connected to the internal network of connected ECUs using a popular network standard known as Controller Area

Network (CAN) (Siddiqui *et al.*, 2017). This gives access to all the other actuation units such as the brakes, accelerator, steering control etc. (Siddiqui *et al.*, 2017). In the other words, because there is an interconnection between outside area and the vehicle, the modern vehicles are vulnerable to security threats which can adversely affect the safety (Schmittner *et al.*, 2015). Therefore, we can reason that if an attacker is able to access the in-vehicle network either directly or remotely, safety and security of the vehicle are endangered (Schmittner *et al.*, 2015). For that we need a safety control as well as including sufficient security features to resist security attacks.

Step 2, 3, and 4:

We can summarize the works that have done in step 2, 3, and 4 as follow:

After conducting the safety analysis in step1 we repeat the analysis process again in order to create revised scenarios. And as we mentioned above, that this process is a scenario-oriented process, so the architecture is created in iteration manner, by take the scenarios and then ranked and make them in sorted groups. Fig. 8.4 presents one of the final results for these steps (a general scenario). As we see, Fig. 8.4 is a statechart specification and a safety-oriented solution. That leads to the question "why we use the statechart modeling?" The answer is "because we want to select a safety-driven design pattern of statechart which enhances the safety in the software architecture".

8.2.1.3.B Safety and Security Risk Scenarios

This section describes how statecharts can be used to model state-dependent interaction scenarios.

The safety assessment model is interconnected with the system model and potential attack and failure (risk) scenarios are described through the models. We used the statechart models to describe the risk scenarios, see Sec. 8.2.1.3.C. And then we used these models to define the Markov chain of each scenario (e.g. Fig. 8.15). These Markov models are then used in the mathematical calculations in the assessment process, see Sec. 8.4.

We can classify the risk scenarios into two categories: security-less safety risk scenarios and security safety risk scenarios. Here we can consider all the defined scenarios are safety and security risk scenarios. The following are some important and general safety risk scenarios.

The scenarios for the car suggested in this work are however different from that for a typical car. This meant that scenarios for this work had to be defined. We assume that in each risk scenario there may be a hazard (s) which can lead to dangerous situation like an accident. Examples of such hazards are: unintended take-off, unintended standstill, unintended braking, loss of brake function, loss of brake trigger function. As all the aforementioned hazards can cause by an attacker we can consider all the defined scenarios are safety and security risk scenarios. These scenarios are later used in the safety assessment process, see Sec. 8.4.

The following are some of the most important risk scenarios:

Scenario 1*: The EBS system is in Initiating state and an unintended take-off occurred when there is an obstacle in front of the vehicle that can cause an accident.*

Example of the security effect: Denial of service attack on in-vehicle CAN blocks the information transmission on-board. And this situation implies that the vehicle would not be able apply correct commands which may lead to hazards.

Scenario 2*: The vehicle is not able to gain real time information from its sensors about its surroundings.*

Example of the security effect: With the successful DoS attack, communication channels are blocked with sufficient amount of irrelevant data packets. This would cause command inputs to either be lost in the transmission or be delayed long enough for a hazard to occur.

Scenario 3*: Failure to detect or late detection of an object. (Similar to that happens with scenario 2)*

Example of the security effect: Hazard which can be also caused by DoS attack on in-vehicle CAN is a failure to detect or late detection of an object.

Scenario 4*: Failure in the computing process*

Example of the security effect: As mentioned in Section 8.2 that when an obstacle is detected, a signal will be sent to the CU via communication media. After receiving

the signal, the CU computes some required parameters aiming to control the speed of the vehicle and braking safely. But in situation if the in-vehicle CAN is flooded with DoS information packets, the EBS sub-system will not be able to perform safety critical functions such as computing. This is the highest degree severity attack since it blocks one of the core safety functions of the EBS sub-system. Other example of the failure is: failing to trigger the actuator.

Scenario 5*: Failure in the actuating process*

Example of the security effect: Latency of system functions can be introduced by loading other applications to the processor and hence reducing performance of the overall system. If this is introduced to the processor which is being used for the actuation, severity of the attack increases.

Scenario 6*: Manipulating the detection message by the attacker which leads to incorrect command issued by the control unit.*

Example of the security effect: The attacks where attacker(s) manipulates the message being sent to other vehicles in the network (OR to other ECUs in the in-vehicle network) for the purpose of creating an illusion of an accident on the road and/or for initiating emergency braking (Surkovi, 2018). In this scenario the hazard like "unintended standstill" and "unintended braking" may be occur which may cause an accident.

Step 5: Define safety-related test cases:

The test cases are defined early that because we need creating a document as a plan to asses or evaluate the architecture. The output of this step is a definition of architecture evaluation plan. This plan is used to evaluate the architecture in the end of the each iteration and in the last evaluation of the architecture.

Step 6: Apply scenarios to select safety-driven architectural pattern:

In this system we have selected a safety-driven architectural design pattern. We used our proposed safety and security design pattern; see Chapter 6 (Sec. 6.4.4, Fig. 6.3 and 6.4).

8.2.1.3.C The Design-oriented Statechart Models

Chapter 6 shows how the proposed pattern development approach is used to develop a new safety and security pattern. As we mentioned in the previous chapter that using our proposed safety and security pattern require developing a statechart to specify the entity's behavior without safety and security violation. This pattern allows an object to alter its behavior and change its internal state when there is a safety or security violation, and to protect it from introducing in unsafe states. Therefore, it is efficient to use a state-based architectural design approach in the overall SPLAs design lifecycle. For that the state-driven architecture design for safety–critical software product lines presented in Chapter 4 is used.

There are different statechart models have been created through the architectural design process of the EBS SPLA. These statecharts are to model different aspects or artifact. Examples of these models are:

- the statechart model of the EBS software product line before using our proposed safety and security pattern, Fig. 8.3;
- the statechart model of the EBS product line after using the safety and security pattern, Fig. 8.4;
- the statechart models of the different risk scenarios (that before and after using of the proposed pattern), e.g. Fig. 8.5 and 8.6.

Note that, in this thesis the risk scenarios have been used in the safety assessment process, see Sec. 8.4.

For limitation and closed similarity between the different statechart models, we just present some of these statechart models, as follows:

- The statechart model of the EBS SPL before using the developed pattern is depicted in Fig. 8.3.
- The statechart model in Figure 8.4 describes the structure of the EBS software product line after using the safety and security pattern in term of statechart diagram (the general structure of the EBS SPL).
- Figures 8.5, 8.6, 8.7 and 8.8 model two of the individual scenarios of usage (Scenario 1 and Scenario 2 respectively) that before and after using the proposed safety and security design pattern. The sequences determined by the red dashed arcs describe the main activities in the scenarios in the

case there is a safety and security control (or after using the safety and security pattern), and also where there is no a safety and security control.

- The statechart models of the other individual scenarios have been developed in a similar way to that of scenario 1 and scenario 2.

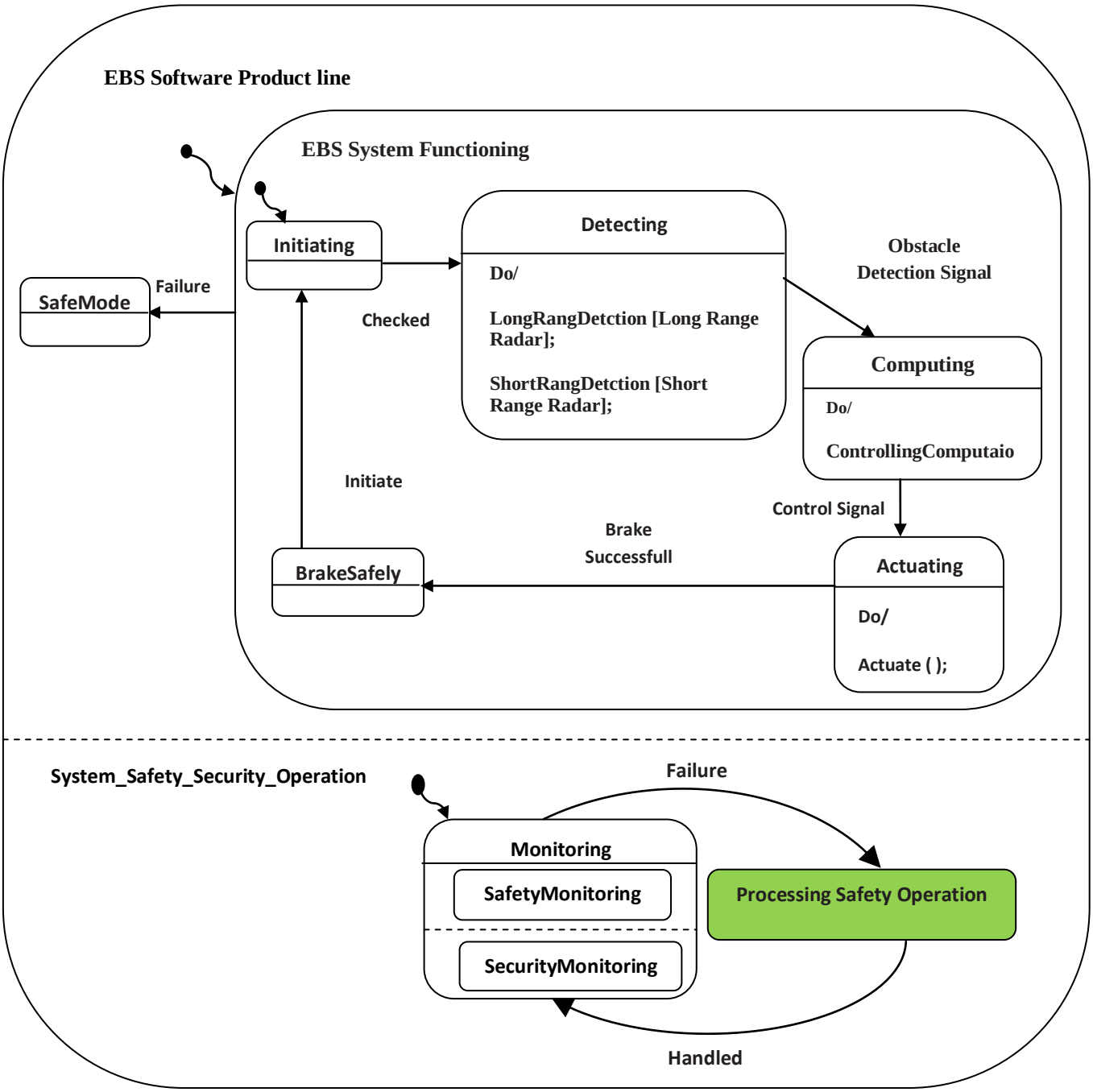


Figure 8.4: The Statechart Model of the EBS software product line after using our safety and security pattern.

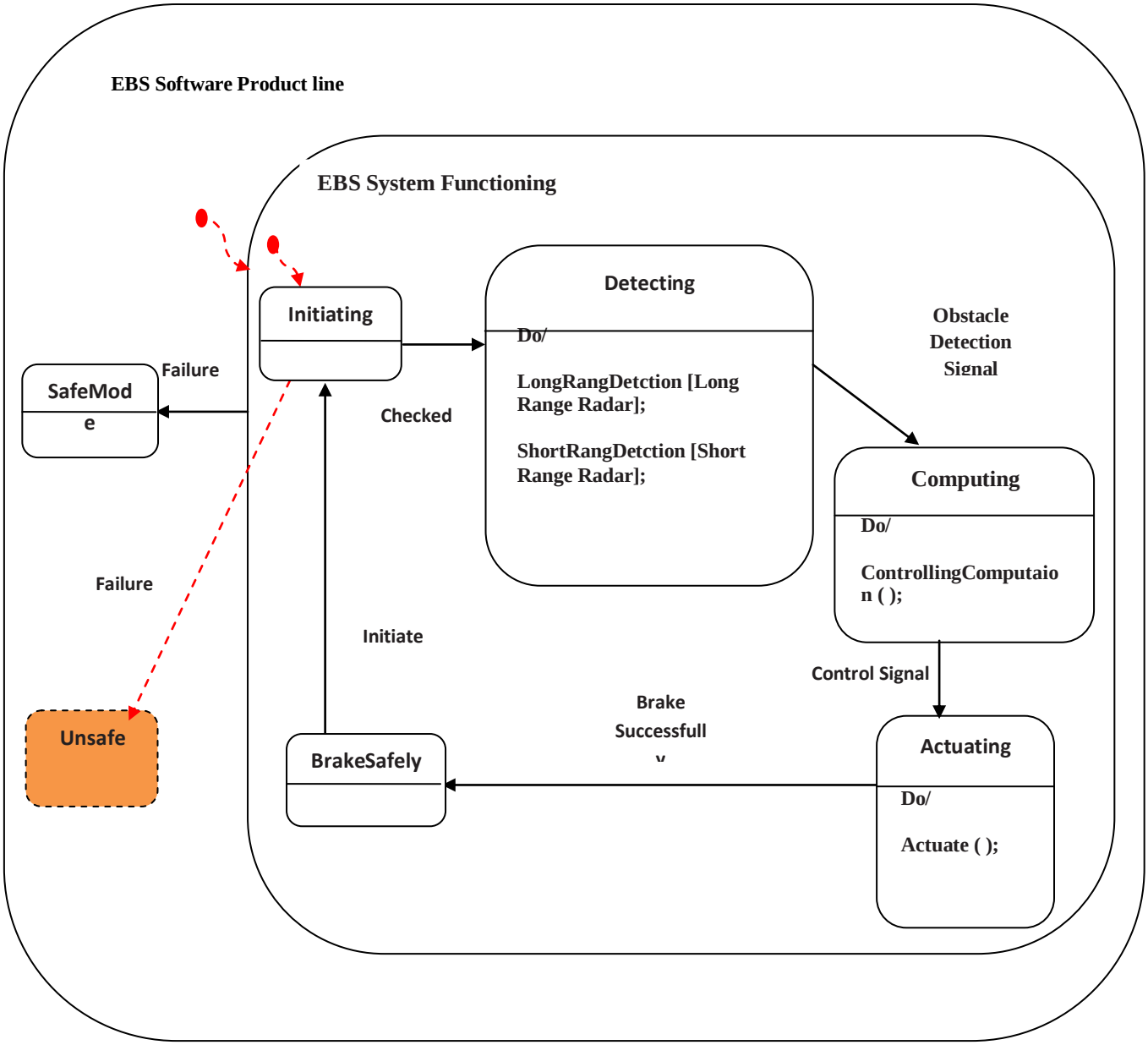


Figure 8.5: The Statechart Model of the Scenario 1 (Scenario of usage)- before using our safety and security pattern- Example 1.

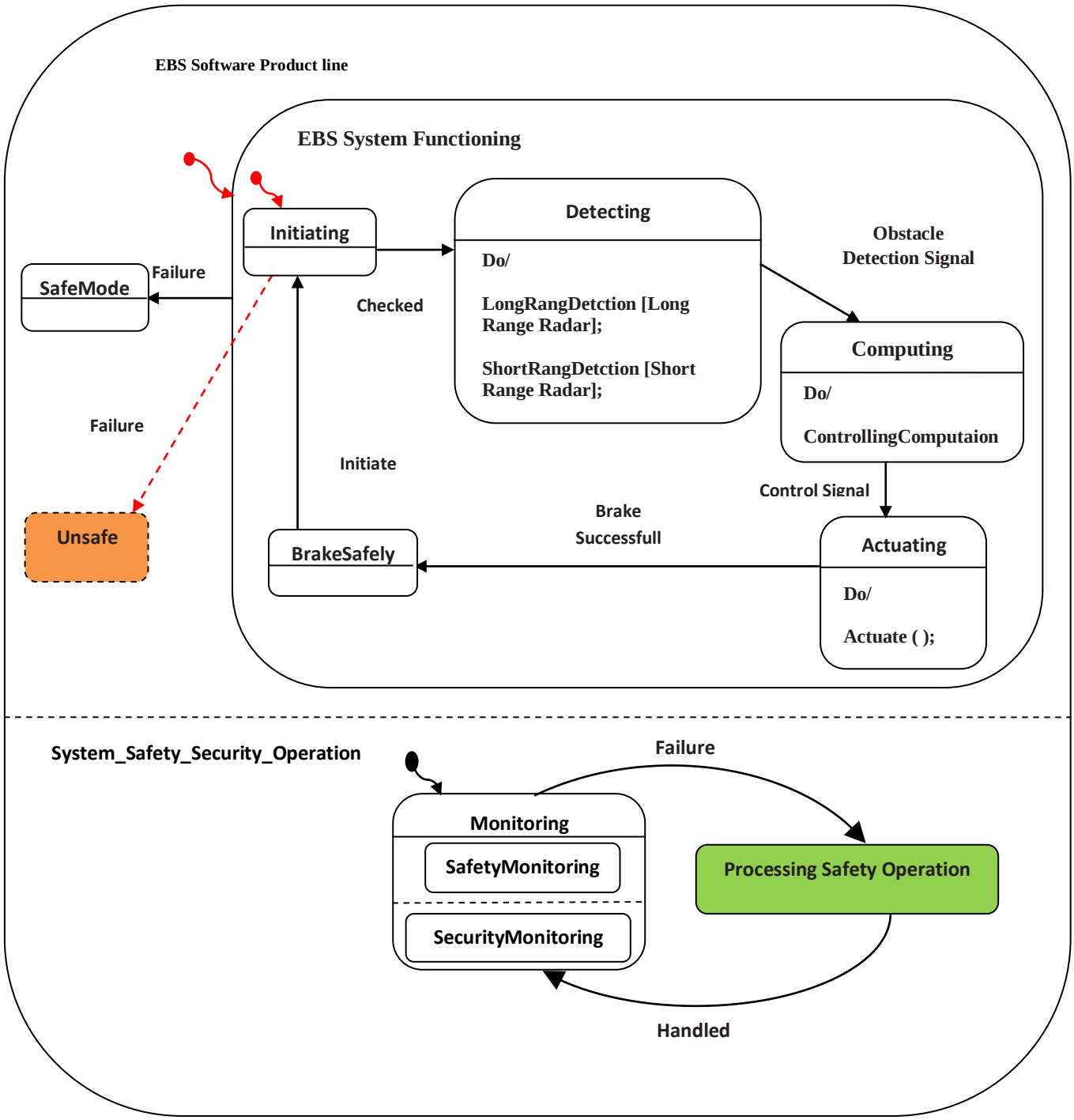


Figure 8.6: The Statechart Model of the Scenario 1 (Scenario of usage) - after using our safety and security pattern- Example 1.

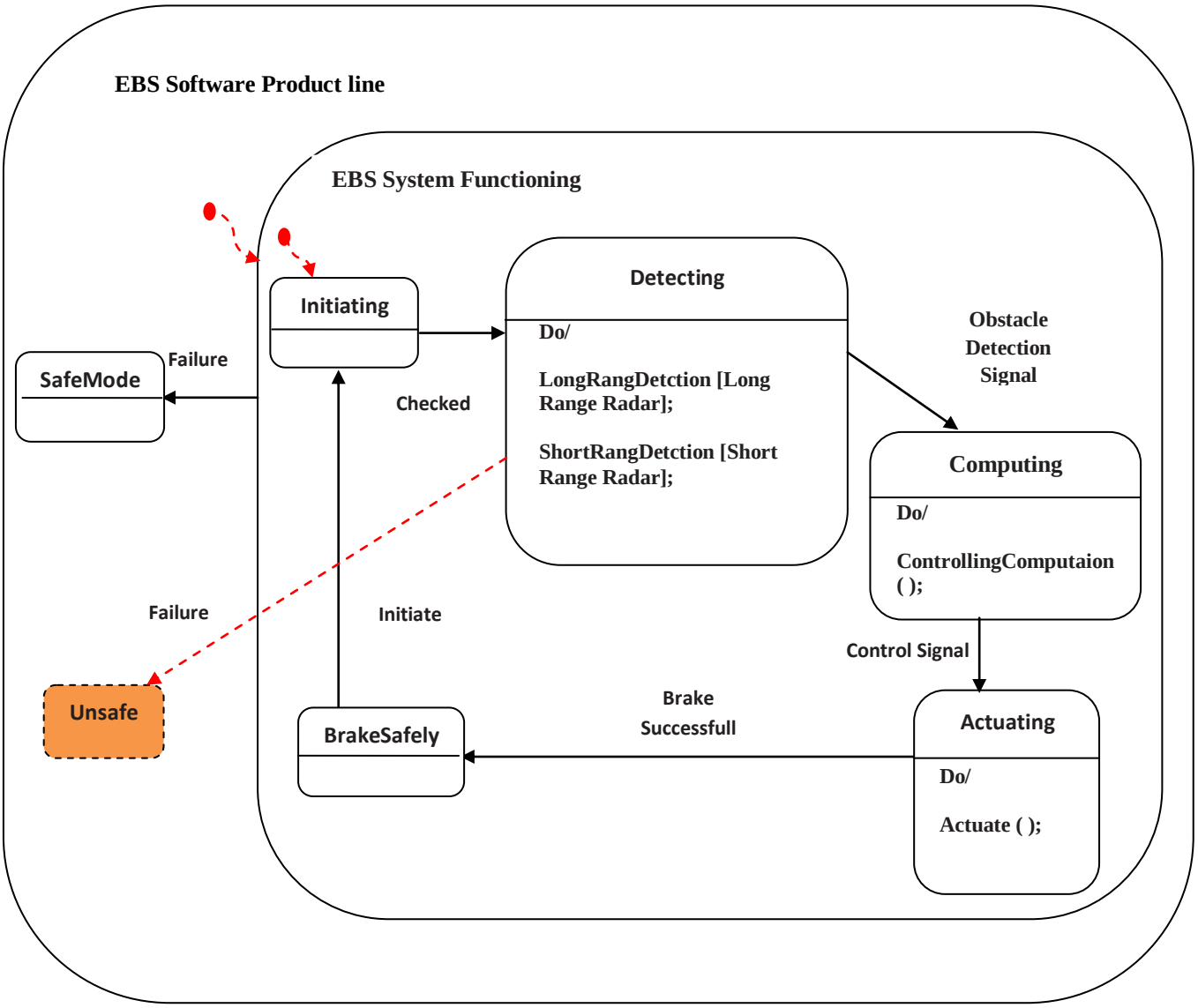


Figure 8.7: The Statechart Model of the Scenario 2 (Scenario of usage) - before using our safety and security pattern- Example 1.

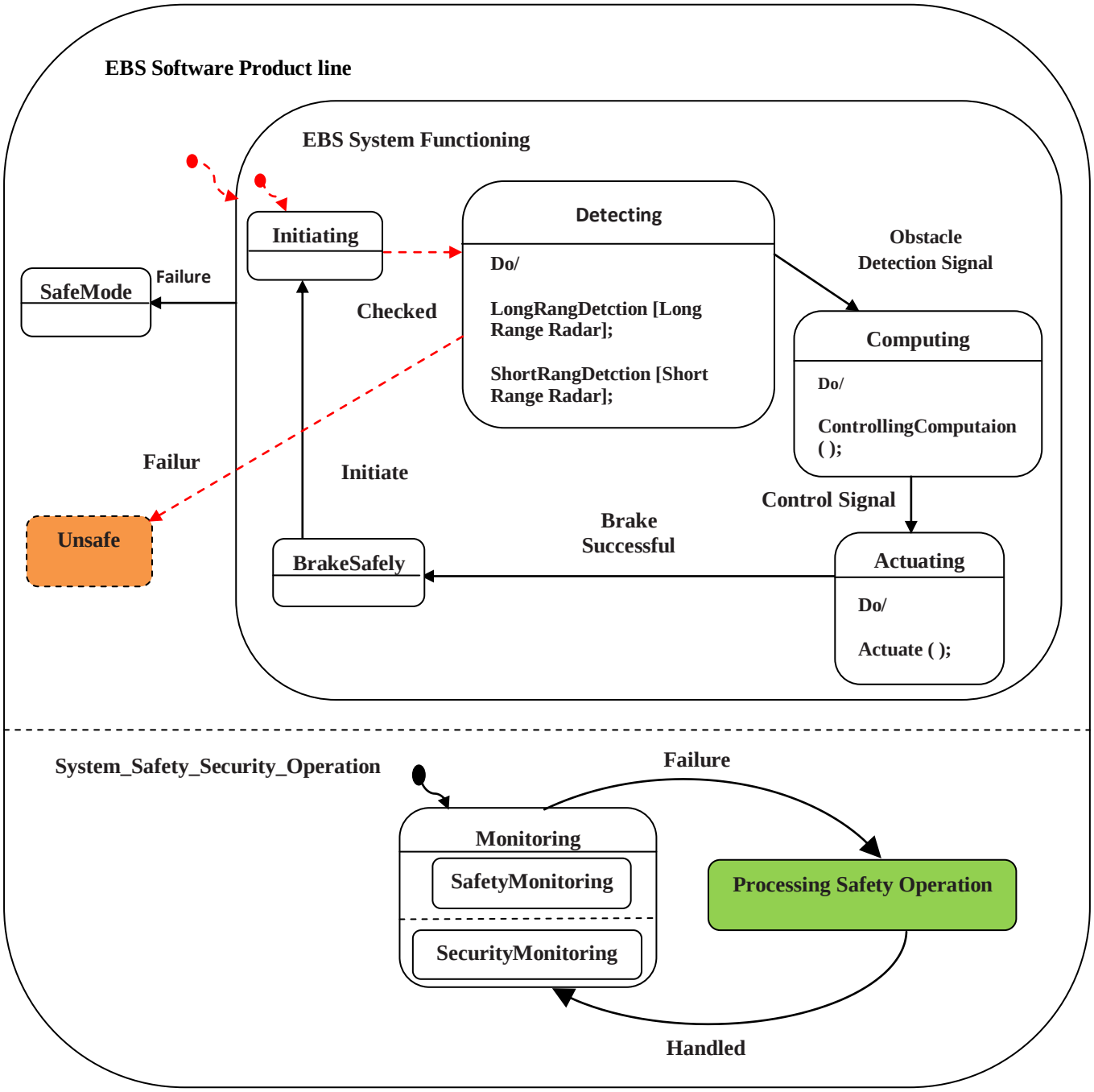


Figure 8.8: The Statechart Model of the Scenario 2 (Scenario of usage) - after using our safety and security pattern- Example 1

Step 7, 8 and 9:

After applying the steps 7, 8, and 9 the final architecture is produced. Figure 8.9 shows an abstract view of the final architecture of the EBS SPL which depicts it in an object-oriented design.

The statechart design model in UML class diagram is described in Section 8.2.2 bellow, and depicted in Fig. 8.9. This UML class diagram describes the architecture of the EBS software product line after using the safety and security pattern.

Note that, due to similarities among the produced models the other models and details are not presented.

8.2.2 The description of the developed EBS SPL Architecture

As we mentioned in Chapter 4 that the output of our architectural method is a software product line architecture. This section presents the final output of the design process.

The main goal of the Object-oriented methodologies using statecharts is to describe in sufficient detail the steps to be followed for describing the behavior of objects (Niaz and Tanaka, 2003). We can implement the statecharts, which specify the dynamic behavior of the classes to implement the behavior of an object-oriented system (Niaz and Tanaka, 2003). In this context we have to implement the UML statecharts in an object oriented design structure. Number of approaches are defined for implementing the statecharts in the object-oriented design. For more detail about this context see references (Yacoub, 1998) (Niaz and Tanaka, 2003)(Ammar, 2013).

This section presents an abstract view of the developed EBS SPL Architecture after using our safety design pattern- A design solution structure in UML notation. Figure 8.9 bellow shows the statechart design model of the EBS SPLA in UML class diagram. This UML class diagram describes the architecture (the reference architecture) after using the safety and security design pattern.

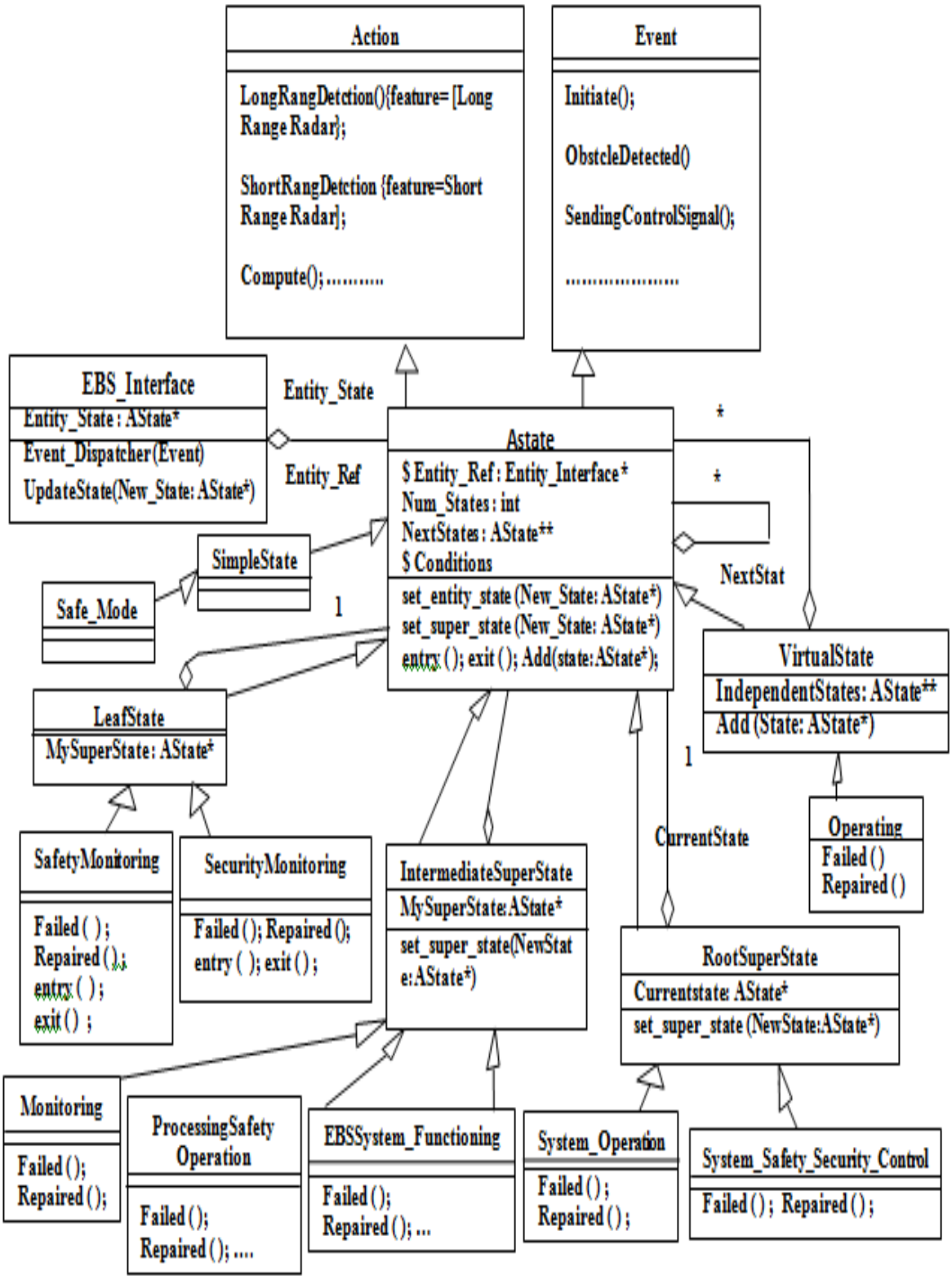


Figure 8.9: An abstract view of the EBS product line architecture after using our safety design pattern- A design solution structure in UML notation.

8.3 The Case Study 2

In this thesis and as a second case study, we use the Smart Microwave Oven Control Systems Software Product Line. The Smart Microwave Oven is a special home appliance that has several operations, such as setting command, setting timer, starting and so on, and that can be operated remotely. The microwave oven will form the basis of this product line, which will offer options from basic to top-of-the-line (Gomaa, 2004).

The process used with this second case study is similar to that used with the first case study. So, and due to lack of space, in the following sections we will just present a short descriptions about the important aspects of the case study, including the safety and security issues, presenting of some individual risk scenarios, as well as presenting of some models produced through the design process such as the statechart models and the statechart design model in UML class diagram. Note that, due to the similarity among the models the other models and details have not been presented.

8.3.1 Safety and Security Issues

Sometimes the device may malfunction. Example, the microwave may keep cooking for an hour, which is not required by the users. Another risk example, the Microwave oven may blow up or become too hot to touch. Additionally, some of unauthorized influences (e.g. DoS- (Denial of Service), the use of IoT devices for malicious purposes) lead to failures and failures of the critical systems that are part of the IoT. For example, disconnect the line between the remote system and the Microwave oven which can lead to dangerous situations. Consequently, we need a safety and security control. The control means it can detect such malfunctioning or even attacks and deal with that by updating the state of the devices, stop it (using operate use case), and inform the user what happened. Also it can address the security issues that influence the safety of the system.

8.3.2 Safety and Security Risk Scenarios

The scenarios for this work had to be defined. We assume that in each risk scenario there may be a hazard (s) which can lead to dangerous situation, for example the microwave oven might blow up or become too hot to touch. As we mentioned above that we can consider all the defined scenarios are safety and security risk scenarios. These scenarios are later used in the safety assessment process, see Sec. 8.4.

The following are some important and general safety risk scenarios.

1. The cooking with door opened scenario.

In normal situation the cooking is possible only when the door is closed. But sometimes and due to some hardware malfunctions the heating element can execute while the door is opened. Also the attackers can send fake signals (for example signal appears that the door is closed)

2. The Cooking is permitted when there is no an item in the oven.

Every microwave oven has a weight sensor. Cooking is permitted only when there is an item in the oven. The risk here is a permission of cooking when there is no an item in the oven. In this case the microwave oven may blow up or become too hot to touch. And that maybe happen due to hardware failures or security problems (like, sending a fake signal inform that there is an item in the oven).

Other safety risk scenarios include:

Note that and due to lack of space we omitted the full description of these scenarios.

3. The continuing of cooking without stop – due to system failure (e.g. failure in the Heating Element).

4. Changing in the cooking recipe by attacker scenario – Security and optional scenario.

5. Failure in the timer that makes system does not stop cooking the food.

6. The attacker send two signals to the microwave oven control object the first one indicates that the door is closed and second one is a start cooking command.

7. Attacker frequently sends a Minute Plus signals when the system is in the Cooking state, which increases the time of cooking.

8. Changing in power level.

9. Unexpected shutdown to the microwave oven due to software failure and that when the system is in one of its critical states (specify just the critical states).

10. The system changes from Ready to Cook state to unsafe state, e.g. due to a big time value to cook, or changing to Cook state when the door is opened.

8.3.3 The Statechart Models

There are different statechart models created through the architectural design process of the Smart Microwave Oven SPLA. These statecharts are to model different aspects or artefacts. Examples of these models are: the statechart model of the Smart Microwave Oven software product line before using our proposed safety and security pattern, the statechart model of the Smart Microwave Oven product line after using the safety and security pattern, Fig. 8.10; the statechart model of the different risk scenarios (and that before and after using of the proposed pattern). As we mentioned in the above section (Sec. 8.2.1.3.B) that these risk scenarios are later used in the safety assessment process.

For limitation and closed similarity between the different statechart models, we just present one statechart model. This model (Fig. 8.10) describes the structure of the Smart Microwave Oven software product line after using the safety and security pattern in term of statechart diagram (the general structure of the SPL). The other statechart models have not been presented. In the following lines we explain some of the models using Figure 8.10:

- The statechart model of the Smart Microwave Oven SPL before using the developed pattern is similar to that model in Fig. 8.10 except the existent of the System_Safety_Security Control super state in this Figure.
- For Scenario 1: in Fig. 8.10, the sequences determined by the red dashed arcs describe the main activities in Scenario 1 in the case there is a safety and security control.
- The statechart models of the other individual scenarios are developed in similar way to that of scenario 1.

The statechart design model in UML class diagram (Fig. 8.11) is described in Section 8.3.4. This UML class diagram describes the architecture of the Smart Microwave Oven software product line after using the safety and security pattern. Finally, and as we mentioned in the above sections that due to similarity among the models and details the other models and details are not presented.

8.3.4 The description of the developed Microwave Oven SPLA

As we mentioned in Chapter 4 that the output of our architectural method is a software product line architecture. This section presents the final output of the design process. The section presents an abstract view of the Smart Microwave Oven SPL Architecture using our safety design pattern- A design solution structure in UML notation. Figure 8.11 bellow shows the statechart design models of the Smart Microwave Oven SPLA in UML class diagram. This UML class diagram describes the produced reference architecture after using the proposed safety and security pattern.

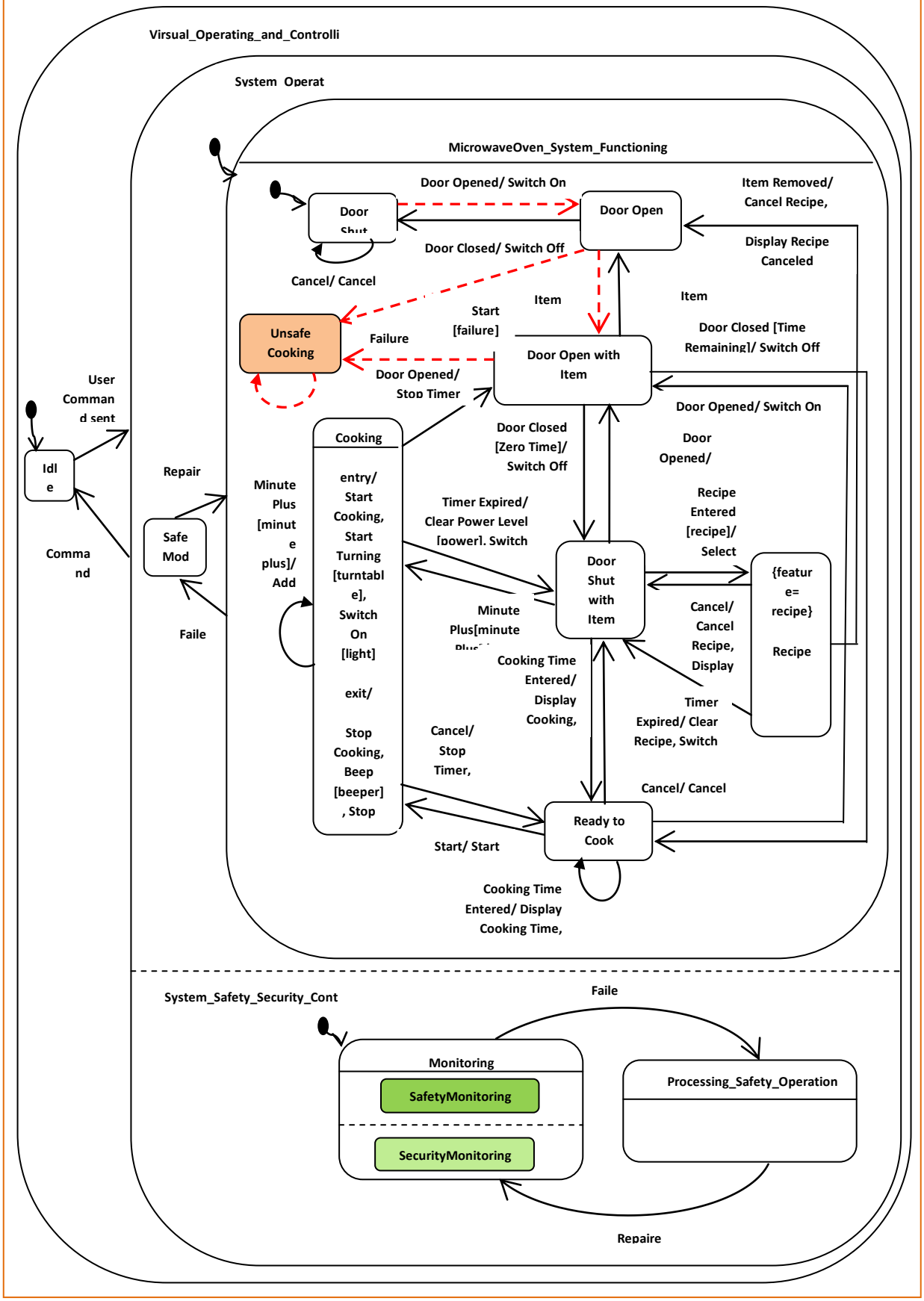


Figure 8.10: The Statechart Model of the Smart Microwave Oven product line after Using our Safety Pattern

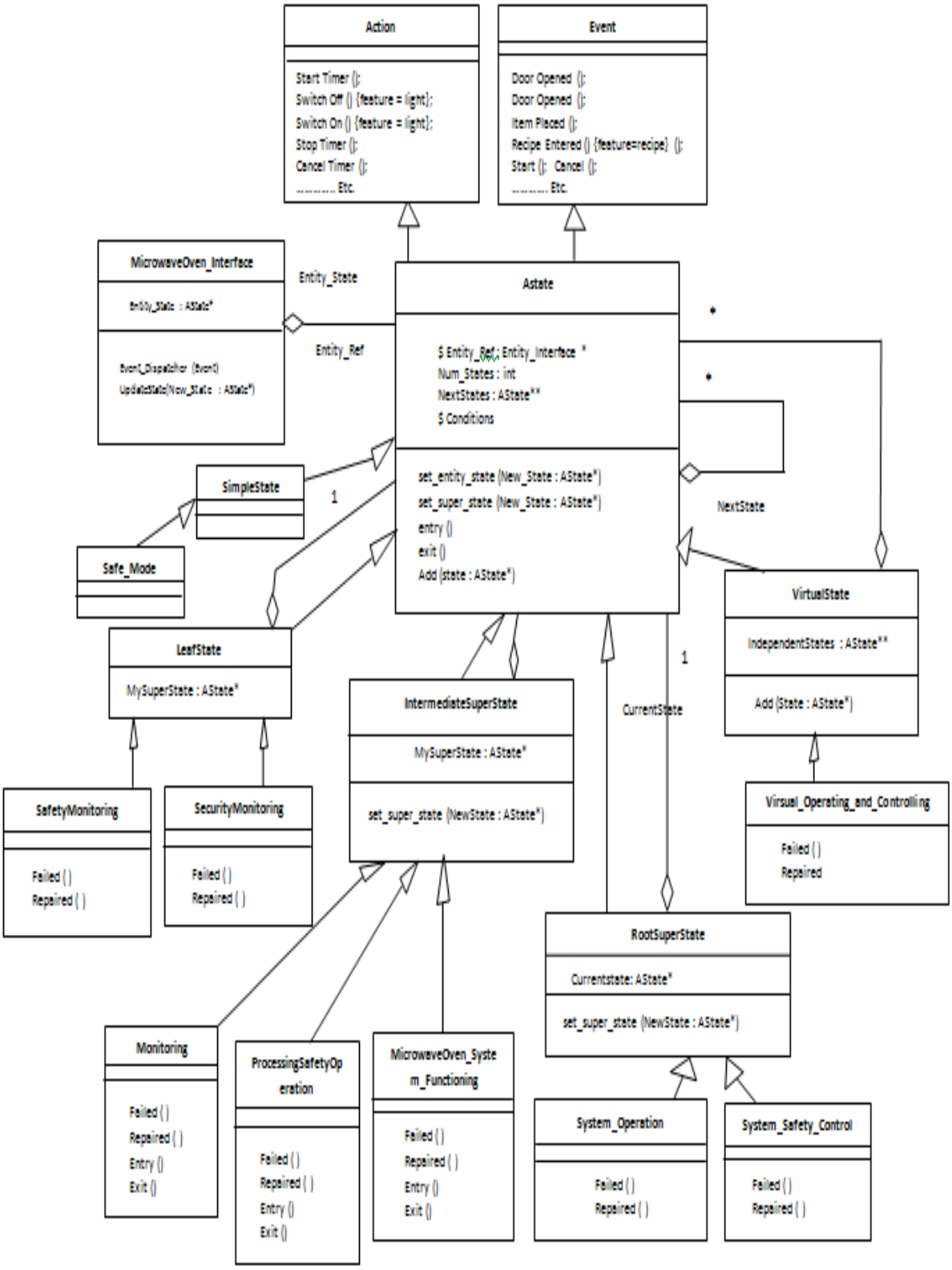


Figure 8.11: An abstract view of the Microwave Oven product line architecture after using our safety design pattern- A design solution structure in UML notation.

8.4 The Evaluation

The above sections (Sec. 8.2 and Sec. 8.3) illustrate the applicability of our work. This section shows its evaluation process with the using of the developed safety and security pattern (presented in Chapter 6) and by using the two application examples of the tow case studies. The final results of the evaluation process and the discussion of these results are shown in Section 8.4.3.

To accomplish this evaluation process we used the proposed safety assessment method presented in Chapter 7 with some adaptations (e.g. adapting the method to be a scenario-based assessment method, adding the metric or the concept of Relative Safety Improvement RSI). The safety and security risk scenarios of the two software product lines, the simplified Automated Electromechanical Braking PL and the Smart Microwave Oven PL defined respectively in Sec. 8.2 and 8.3 above have been used.

The safety assessment process model used here is a simplified mathematical model for safety assessment of the product lines architectures. The effective of this model is that it is a dynamic model which means it addresses the system safety at the runtime. The major steps of the assessment model after its adaptation to be a scenario-based model are mentioned and described briefly in the following lines:

- Define the statechart design models before and after using the proposed pattern for each defined scenario in the given software product line system.

- Developing the Markov chain models before and after using the proposed pattern for each scenario- These models can elicit by using the statechart model defined in the above step.

- Conduct the calculation processes of the safety assessment process using mathematical methods upon the resulted Markov model. This step includes Creation of the Transition Probability Matrix for each Markov model.

Markov chains are effective tools that used for evaluating the safety and reliability of architectures (Varshosaz and Khosravi, 2013). After defining the Markov chains then they will be evaluated regarding the probabilities that the system is in a certain state at time *t*. In this thesis we used the steady state evaluation technique which calculates the probabilities for $t \rightarrow \infty$.

Notes:

Note 1*: Here we have used Equation 4 (presented in Chapter 7) to calculate the transition probabilities on the Markov chain. And then we created the Transition Probability Matrix p. Other Markov processes have been used like steady state operation. Finally we calculate the probability of each state for the system states before and after using the proposed design pattern for each scenario. And finally we will observe the calculation results and then write the assessment results. For more explanation refer to the Illustrative Example in Sec. 7.5*

Note 2*: In our work and for both examples we have applied this assessment process by using the kernel system, an advanced product of the Software Product Lines as well as the individual risk scenarios. As mentioned above, for chapter lines limitation, we have just presented the results of using the individual risk scenarios.*

8.4.1 The Evaluation-Using Individual Risk Scenarios-With Example 1

The final result of this evaluation is to show the improvement in the system safety design with the influence of the security issues and that after using the proposed safety and security pattern. This means that the calculations process must be conducted in both two cases, before and after using the pattern and that for each scenario.

To explain how the calculations in the assessment process are conducted we describe this by using scenario 1. And these calculations are then repeated for all the other scenarios. For the other scenarios the process is similar to that with scenario 1. The selected scenario is scenario 1 (The EBS system is in Initiating state and an unintended take-off occurred when there is an obstacle in front of the vehicle that can cause an accident (see the Sub Sec. 8.2.1.3.B)

As we mentioned in the above lines that for lines limitation the section just presents the calculations details of the scenario 1, that in the two cases, before and after using the proposed safety and security pattern.

The details of the evaluation process are as follow:

Firstly: The calculation results of scenario 1- before using the proposed pattern

The process can be conducted in a steps as follows:

Step 1: Define the statechart model scenario 1 in the SPL system before using the proposed pattern.

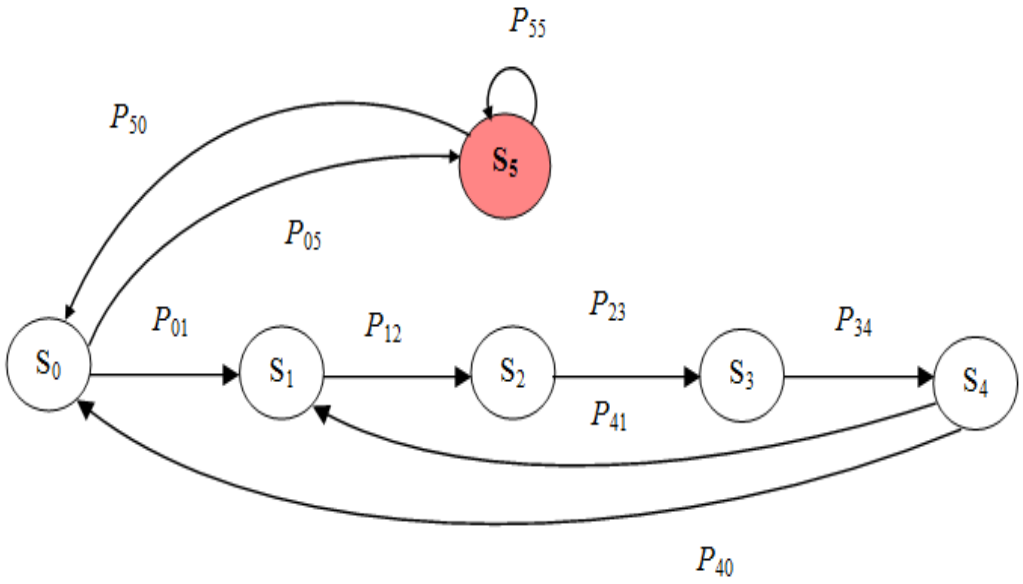
The statechart in Fig. 8.5 shows the description model of the scenario (The red dashed arcs in the statechart).

Step 2: Create the Markov Chain from the statechart model presented in Fig. 8.5.

Fig. 8.12 shows the Markov chain for scenario 1 before using the proposed safety design pattern. We can use the states abbreviations in Table 8.2.

Table 8.2: The abbreviations of the Markov states

The State	Initial State	Detection state	Computing state	Actuating state	Safe Brake	Unsafe State	Safe State
Abbreviations	S_0	S_1	S_2	S_3	S_4	S_5	S_6



Legend: - S_n = The System States; P_{ij} = The Transition Probability

Figure 8.12: The Markov Chain model of scenario 1 before using the safety security pattern-Example 1

Step 3: Calculate and define the transitions probabilities for the given Markov chain.

We use the maximum likelihood estimation method (MLE) mentioned in Chapter 7 (Sec. 7.2.2.2) to do that. Fig. 8.13 shows the Markov chain with the transition probabilities.

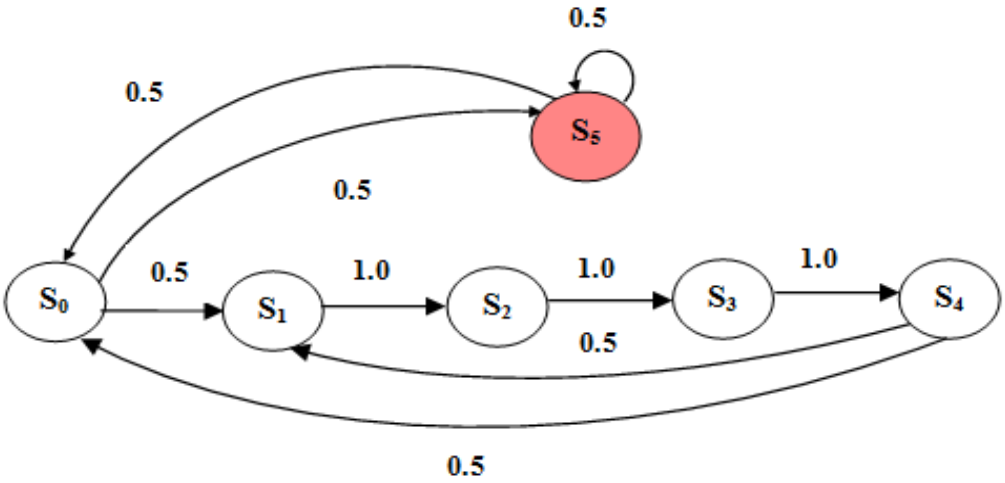


Figure 8.13: the Markov chain with the transition probabilities - scenario 1 before using the safety and security pattern-Example 1

Step 4: Create the Transition Probabilities Matrix TPM

Then we create the Transition Probabilities Matrix (P) of the Markov model. The matrix in Fig. 8.14 is the TPM (*p*) .

$$[S_0\ S_1\ S_2\ S_3\ S_4\ S_5\ S_6] * \begin{array}{c} \\ S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \end{array} \begin{pmatrix} S_0 & S_1 & S_2 & S_3 & S_4 & S_5 \\ 0.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.5 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.5 \end{pmatrix} = [S_0\ S_1\ S_2\ S_3\ S_4\ S_5]$$

Figure 8.14: An equation containing the produced Transition Probabilities Matrix of Scenario 1 before using the safety and security pattern-Example 1

Step 5: Define the Markov process.

By using the concept steady state we define the probability of each state of the Markov model which present the abstract state of the system.

Step 6: Solving the given Markov chain to obtain the steady state probability vector.

After solving the resulted equations system produced from the equation in Fig. 8.14, the steady state probabilities are obtained. Table 8.3 shows the probability of the system to be in each state after the execution of the system for long time (in case of scenario 1).

Secondly: The calculation results of scenario 1- after using the proposed pattern

These calculations have been also conducted for the same scenario (scenario 1) of the system in the case there is a safety and security control in the design of the system (or after using the safety and security pattern). The details are as follow: Fig. 8.6 shows the description model of the scenario (The red dashed arcs in the statechart), Fig. 8.15 shows the Markov chain with the transition probabilities. Then we create the Transition Probabilities Matrix of this Markov model, Fig. 8.16. The Table 8.3 also shows the probability of the system to be in each state after the execution of this system for long time that after using the pattern. The reason is to make a comparison between the two results. This step is to show the Relative improvement in the safety design. As we mentioned above that the overall calculations are then repeated for all the other scenarios. Figure 8.17 presents a comparison between the probability of the system to be in unsafe state for each scenario before and after using the developed pattern. It is obvious that there is a considerable improvement in the system safety design after using the proposed safety and security pattern. In Sec. 8.4.3 discussions of the final results as well as summarizing of the overall evaluation process of our work have been presented.

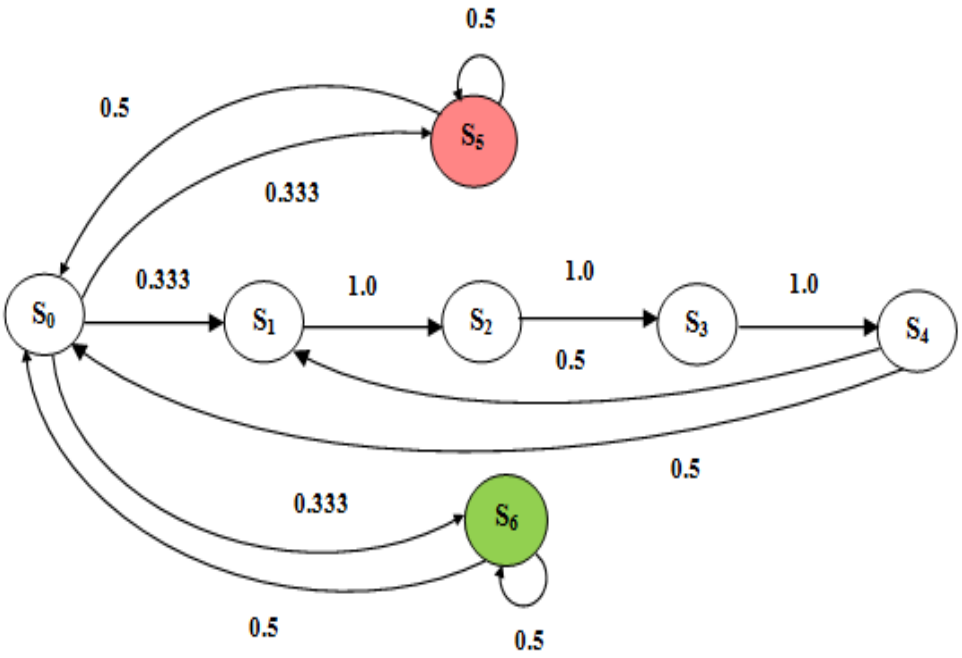


Figure 8.15: The Markov model with transition probabilities of scenario 1 after using the safety security pattern-Example 1

$$[S_0\ S_1\ S_2\ S_3\ S_4\ S_5\ S_6] * \begin{array}{c} \\ S_0 \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \end{array} \begin{array}{c} \begin{array}{ccccccc} S_0 & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 \end{array} \\ \begin{pmatrix} 0.0 & 0.333 & 0.0 & 0.0 & 0.0 & 0.333 & 0.333 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.5 & 0.0 \\ 0.5 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.5 \end{pmatrix} \end{array} = [S_0\ S_1\ S_2\ S_3\ S_4\ S_5\ S_6]$$

Figure 8.16: An equation containing the produced Transition Probabilities Matrix of Scenario 1 after using the safety and security pattern-Example 1

Table 8.3: The probability of the system to be in each state for scenario 1 that before and after using the safety and security pattern-Example 1

The probability of the system to be in each state for scenario 1 that before and after using the safety and security pattern	**Initiating**	**Detecting**	**Computing**	**Actuating**	**Brake Safely**	**Unsafe State**	**Safe State**
	S_0	**S_1**	**S_2**	**S_3**	**S_4**	**S_5**	**S_6**
Before using the pattern	0.1667	0.1667	0.1667	0.1667	0.1667	0.1667	----
After using the pattern	0.2	0.1334	0.1334	0.1334	0.1334	0.1333	0.1333

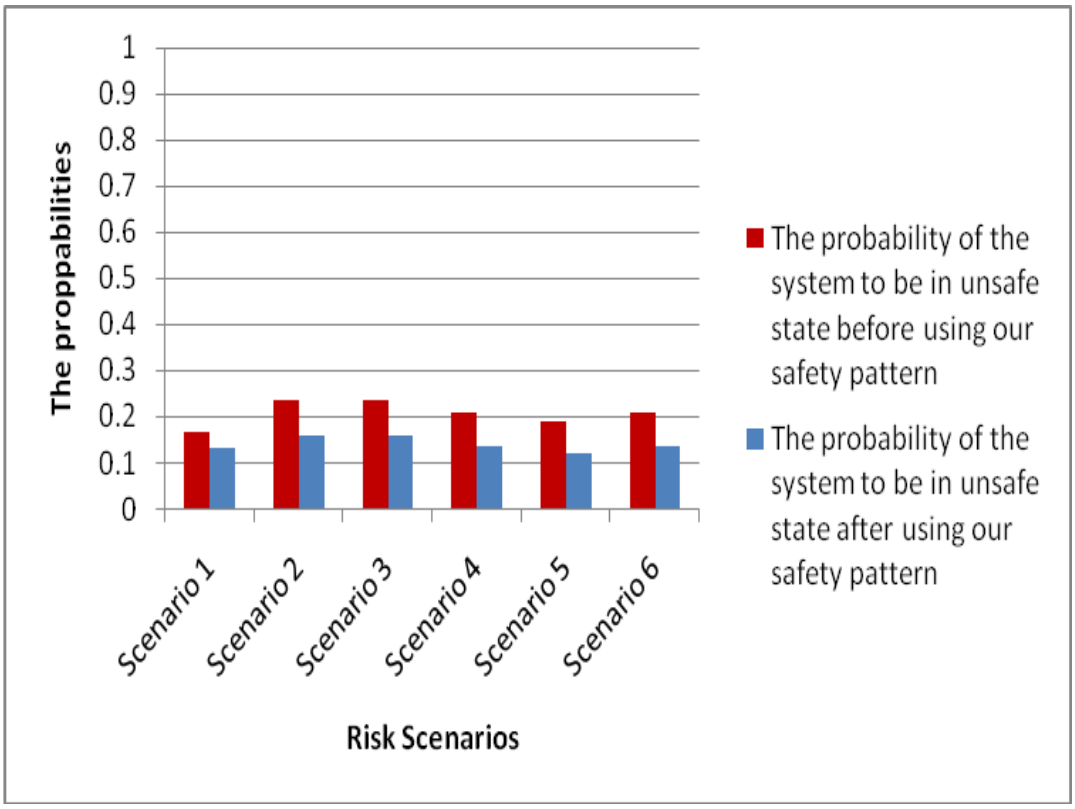


Figure 8.17: Comparison between the probability of the system to be in unsafe state in each scenario before and after using the pattern-Example 1

8.4.2 The Evaluation Using Individual Risk Scenarios-With Example 2

The evaluation process with example 2 is executed in a similar way to that used with example 1. This section presents summarized details of the process. It just presents the calculations details of the scenario 1 (The cooking with door opened scenario, see Sec. 8.3.1) and that after using the proposed safety and security pattern.

The details are as follow:

Fig. 8.8 shows the description model of the scenario (The red dashed arcs in the statechart); Fig. 8.18 shows the Markov chain with the transition probabilities; then defining of Transition Matrix, Fig. 8.19.

After defining the Markov chain then it can be evaluated regarding to the probabilities that the system is in a certain state at time *t*. In this thesis we used the steady state evaluation technique which calculates the probabilities for $t \rightarrow \infty$.

In the last, the Table 8.4 shows the probability of the system to be in each state after the execution of this system for long time (in case of scenario 1). Table 8.4 also shows the probability of the system to be in each state after the execution of this system for long time that in case there is no safety and security control (or before using the pattern). Fig. 8.20 presents a comparison between the probability of the system to be in unsafe state in each scenario and that before and after using the developed pattern. It is obvious that there is a considerable improvement in the system safety design after using the proposed safety and security pattern.

In Section 8.4.3 discussions of the final results as well as summarizing of the overall evaluation process of our work have been presented.

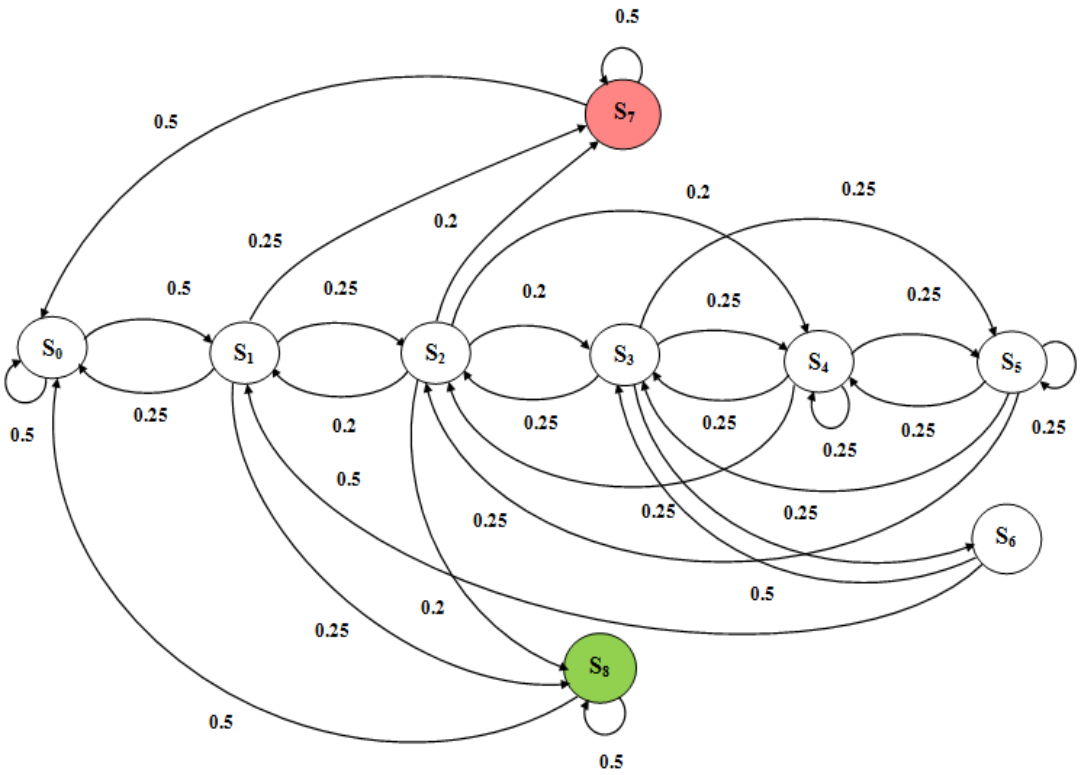


Figure 8.18: the Markov model with transition probabilities of scenario 1 after using the safety security pattern-Example 2

$$[S_0\ S_1\ S_2\ S_3\ S_4\ S_5\ S_6\ S_7\ S_8] * \begin{array}{c} \\ S_0 \\ S_1 \\ S2 \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \\ S_8 \end{array} \begin{array}{c} \begin{array}{ccccccccc} S_0 & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 & S_7 & S_8 \end{array} \\ \begin{pmatrix} 0.5 & 0.5 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.25 & 0.00 & 0.25 & 0.00 & 0.00 & 0.00 & 0.00 & 0.25 & 0.25 \\ 0.00 & 0.2 & 0.00 & 0.2 & 0.2 & 0.00 & 0.00 & 0.2 & 0.2 \\ 0.00 & 0.00 & 0.25 & 0.00 & 0.25 & 0.25 & 0.25 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.25 & 0.25 & 0.25 & 0.25 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.25 & 0.25 & 0.25 & 0.25 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.5 & 0.00 & 0.5 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.5 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.5 & 0.00 \\ 0.5 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.5 \end{pmatrix} \end{array} = [S_0\ S_1\ S_2\ S_3\ S_4\ S_5\ S_6\ S_7\ S_8]$$

Figure 8.19: An equation containing the produced Transition Probabilities Matrixes of Scenario 1 after using the safety and security pattern-Example 2

Table 8.4: The probability of the system to be in each state for scenario 1 that before and after using the safety and security pattern-Example 2

The probability of the system to be in each state for scenario 1 that before and after using the safety and security pattern	**Door Shut**	**Door Opened**	**Door Opened with Item**	**Door Shut with Item**	**Ready to Cook**	**Cooking**	**Recipe**	**Unsafe State**	**Safe State**
	S_0	**S_1**	**S_2**	**S_3**	**S_4**	**S_5**	**S_6**	**S_7**	**S_8**
Before using the pattern	0.3097	0.1909	0.1108	0.0667	0.075	0.0471	0.0167	0.1826	------
After using the pattern	0.357	0.198	0.075	0.036	0.0405	0.0255	0.009	0.129	0.129

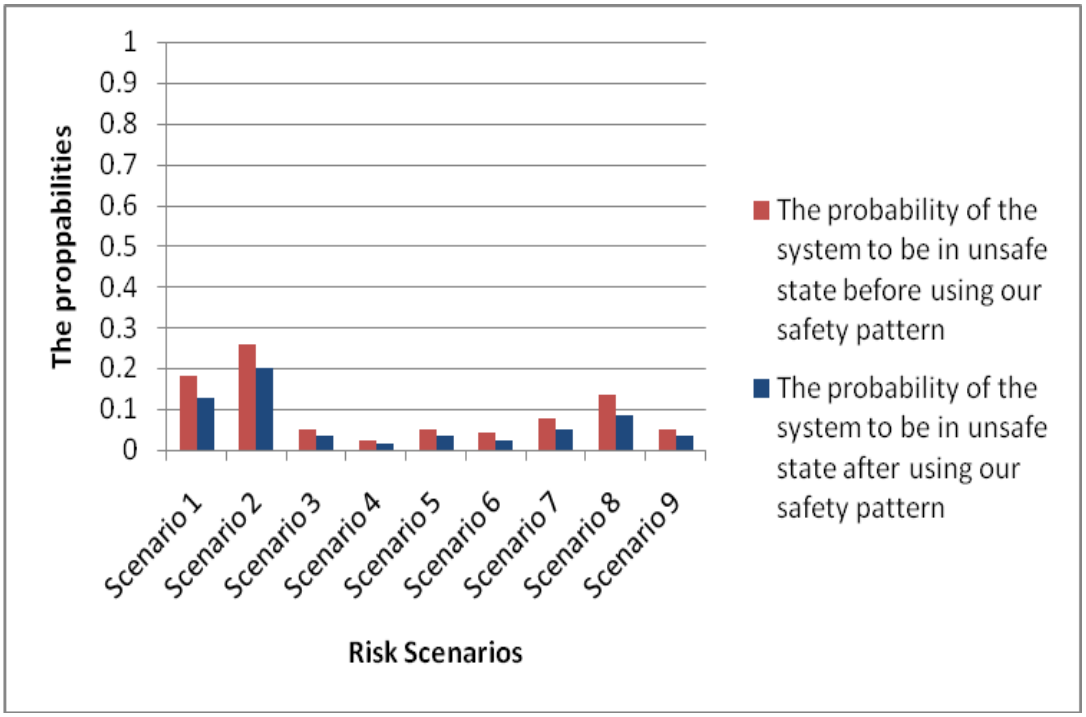


Figure 8.20: Comparison between the probability of the system to be in unsafe state in each scenario before and after using the pattern-Example 2

8.4.3 The Final Results and Discussion

This section presents the final results of the evaluation process as well as a short discussion on the overall results. The section also shows the effectiveness of our work. The final step of our evaluation process is to compute the ***Relative Safety Improvement*** RSI. The Relative Safety Improvement RSI is a safety assessment metric proposed by Ashraf in (Armoush, 2010) which gives an indication about the safety improvement in the design which can be achieved by the pattern. This metric is defined as "the percentage improvement in safety (reduction in probability of unsafe failure) relative to the maximum possible improvement which can be achieved when the probability of unsafe failure is reduced to the minimum possible value (0)". Based on this definition, the relative safety improvement for a design pattern can be expressed as shown in the following equation, (*Equation 1*, mentioned in Chapter 3):

$$\left[RSI = \left(1 - \frac{P_{UF(new)}}{P_{UF(old)}}\right) \times 100\% \right] \qquad 1$$

- *RSI*: Relative Safety Improvement.
- $P_{UF(old)}$: Probability of unsafe failure in the basic system (or in the system before using a safety control).
- $P_{UF(new)}$: Probability of unsafe failure in the design pattern (or in the system after using a safety control).

In this thesis we assume that the probability of the system to be in unsafe state is equivalent to the probability of the unsafe failure. This assumption is achieved by considering that all failures are unsafe failures. By this assumption we do the calculation of the RSI for all the individual scenarios in the two examples of the two case studies.

After applying that we will show the final results presented in the figures bellow (Fig. 8.21 and Fig. 8.22). The final results depicted in the figures (Fig. 8.21 and Fig. 8.22) show that there is a considerable improvement in the safety design of the systems after using our proposed safety and security pattern of statechart in the design process.

As we mentioned in the beginning of this chapter, that this part of our work concerns with the implementation and evaluation of our research contributions. So, In order to show the applicability of this work, the proposed state-driven design method for SSPLAs with the using of the proposed safety and security design pattern is applied on the design process of two software product line architectures. To evaluate our work, a simplified safety assessment model is developed and applied on these case studies.

The research argues that how efficient to use a safety and security design patterns of statechart in the software architectural design process of the safety-critical systems.

The scenario-based assessment method is used for assessing the quality attributes which is the safety attribute. In this context the produced quality scenarios are based on the safety and security requirements. The evaluation process includes analysis of how well software architecture satisfies safety and security requirements. The above sections (Sec. 8.4.1 and 8.4.2) present the analysis results of the individual risk scenarios for the two case studies. These results are shown in Fig. 8.17 and Fig. 8.20 (for example 1 and example 2 respectively). For all the specified scenarios of the two examples, the Fig. 8.17 and Fig. 8.20 show a comparison between the probabilities of the systems to be in the *Unsafe* states, and that before and after using the proposed pattern in the case of executing the systems for a long time.

To make an investigation on our work in the stage of the architectural evaluation we will take both scenario 1 and 2 of the first case study as an example.

In the following lines we will present some discussions on the results as follow:

The results show that the probabilities of the system to be in *Unsafe* state before using our design pattern for *Scenario 1* and *Scenario 2* are: (0.1667, 0.2353 respectively); and after using our design pattern are: (0.1333, 0.16 respectively). It is clear from these results that the probabilities of the system to be in *Unsafe* states for *Scenario 2* are higher than them for *Scenario 1*. The last point means that *Scenario 2* is a high risk scenario if it is compared with the risk *Scenario 1* and that may give an indicator to the developer (or the architect) to deal with this situation. And this situation may require coming back to do more analysis, in order to reduce the probability of transition to the *Unsafe* state or may lead to update the design of the

resulted architecture. In this case the updating or reconfiguration process is easily executed, as it is just achieved by adding, deleting or reconfiguring of states, evens or action. The developer (architect) repeats all the activities in this context with each risk scenario.

In the other hand, by observing the results of the Relative Safety Improvement (RSI) we find that the results depicted in Fig. 8.21 shows that the relative safety improvement (RSI) achieved with *Scenario 2* is higher than that achieved with *Scenario 1*. One can observe that the risk *Scenario 2* has higher Relative Safety Improvement rate than the one to the risk *Scenario 1* despite it is a higher risk. This point proves that by adding more safety and security control in each system state one can achieve a considerable improvement in the system safety and security design. Also by adding safety and security control we can reduce the risk and increase the safety and security level in the entire system design.

According to above, we can conclude that this state-based approach is highly supports the development of the safety-critical SPLAs or the development of the safety and security critical systems in general. By using the state-driven development it is efficient to address the dynamic behavior of the reactive systems and we can use this facility to describe the system behavior in term of safety and security as well as finding the risks, non-risks and sensitivity points. This point is considered as one of the strengths of our work. The results also have proved that it is effective to handle the safety and security together in the design of the safety pattern and that provides more benefits as it is a high level reuse.

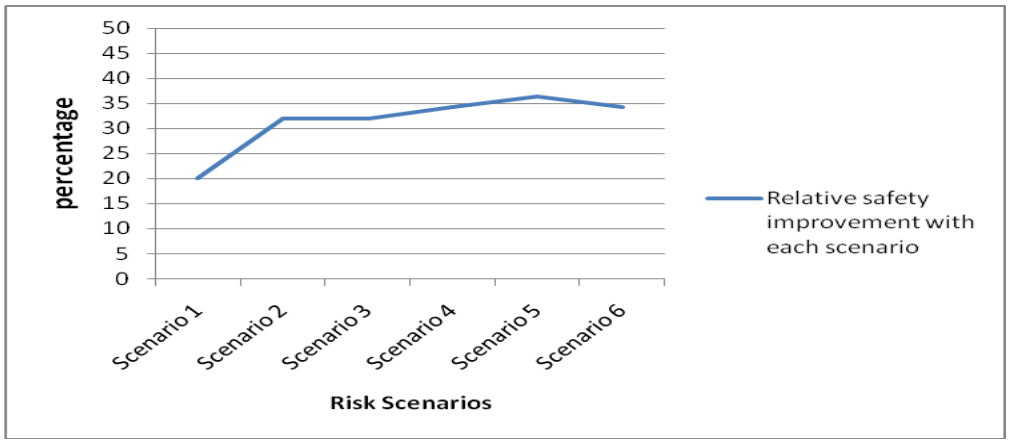


Figure 8.21: The Relative Safety Improvement after using the developed pattern-Example 1

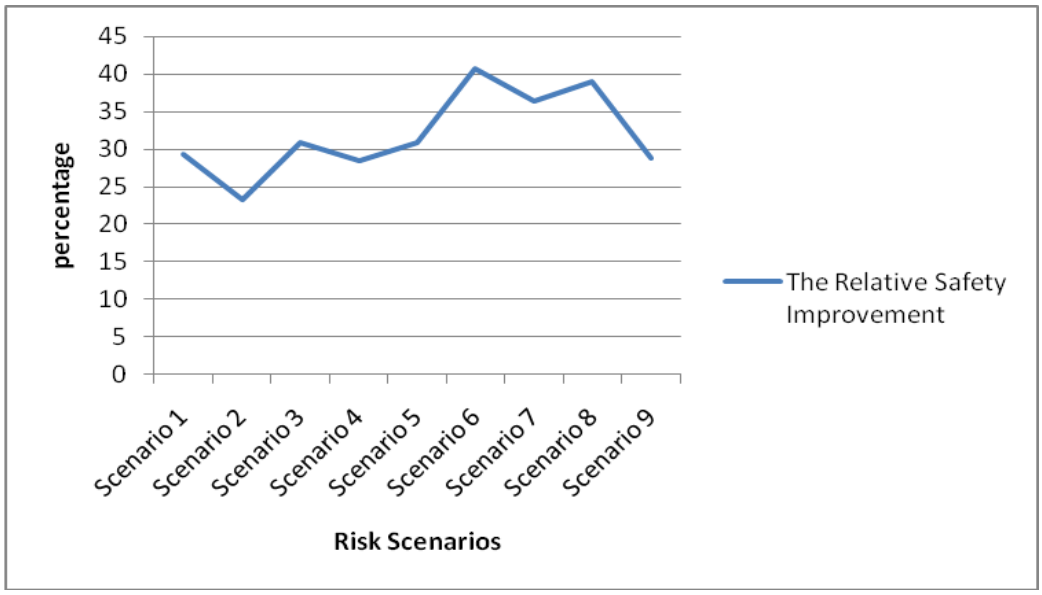


Figure 8.22: The Relative Safety Improvement after using the developed pattern-Example 2

8.5 Chapter Summary

In order to show the applicability of our work as well as evaluate it, two case studies are presented in this chapter. The chapter presented an evaluation of the five main contributions of the research presented in this thesis. The final conclusions of this research are presented in the next chapter.

CHAPTER IX

9

CONCLUSION

This chapter summarizes work presented in this thesis, followed by a description of the contributions and future directions.

9.1 Thesis Summary

The high quality, short delivery time, and high productivity have become more and more important in developing embedded software for modern systems (Nagamine, Nakajima and Kuno, 2016).

The Software Product-Line (SPL) and reusable software components are suitable approaches for such systems, which are often re-engineered from existing systems. A successful SPL supports systematic software reuse and reduces the development effort, meanwhile, improves the quality of the member products.

Software architecture design is one of the critical steps in software development process. One of the effective approaches which are needed to ensure the product quality is Reference Architectures (called also Product Line Architectures). Developing a reference architecture which represents the base structure of the member products is the main task of the software product line architecture design.

The safety is considered one of the most critical issues in the design of the modern systems, specifically the cyber-physical systems (CPS). And as product-line engineering becomes more widespread, more safety-critical software product lines are being built.

With the increasing attention of software safety, how to improve software safety has already become a more important concerned issue, especially for the safety-critical systems (Huang, 2013). Safety-based design at architecture level can effectively improve software or system safety.

From the literature (Chapter 2) it is clear that the existing methods for the safety-based architectural design are not adequate to enhance the architectures design of the modern software product line systems. In the other words, the safety-based architectural design methods are limited in SPLs because of the complexity and variabilities existing in SPL architectures.

For that, in this thesis we have searched to define an efficient and effective method that can be used into the design process of the safety-critical software product line architectures. The thesis proposed a method for ***safety-driven software product line architecture design*** (SSPLA). The method is to enhance and manage the safety of software product lines. The key aspect of this method is the use of the concept design patterns which improves the design process.

As the Unified Modeling Language (UML) statechart diagram is a powerful tool for specifying the dynamic behavior of reactive objects, this facility can be used to describe the system behavior in term of safety. Based on this facility the proposed SPL architecture design method mentioned above is configured and adapted to be ***state-based architecture design method***.

The thesis has illustrated how the use of statechart semantics and patterns can effectively address and improve the safety of the SPLA design. According to the evaluation results, we can conclude that the state-based approach is highly supports the development of the safety and security critical systems. And by using the state-based development it is efficient to address the dynamic behavior of the reactive systems and we can use this facility to describe the system behavior in term of safety and security as well as finding the risks, non-risks and sensitivity points.

The significance of this research is that it presents several significant advances to the fields of safety engineering and design. It presents a process of concurrently developing a system concept from the safety and functional perspective. We believe this work presents an important step in making the design and safety processes more efficient and effective for the software product line.

In general, the work is highly supports the object-oriented software architectures development for the product lines. The research will benefit both architects and safety engineers who can design product line architectures or develop software product in domain of embedded systems and cyber-physical systems.

We review the concrete contributions of our research in more detail in the following section (Sec. 9.2):

9.2 Thesis Contributions

The contribution of this thesis involves method for safety-driven software product line architecture design (SSPLA). The method is to enhance and manage the safety of software product lines. More specifically, the main contributions of this thesis are as follows:

- We have surveyed the literature on software product line architectures design. The survey study provided in Chapter 2 is considered as a systematic literature review of the existing research on software product line architecture (SPLA) design based on quality attributes.
- We have defined a new s***afety-driven SPLA design method*** (Chapter 4). The key aspect of this method is the use of the design patterns concept which improves the design process.
- For efficiency, a number of efforts have been made. In this context the proposed design method mentioned above is configured and adapted to be ***state-based architecture design method*** (Chapter 4). As the Unified Modeling Language (UML) statechart diagram is a powerful tool for specifying the dynamic behavior of reactive objects, this facility can be used to describe the system behavior in term of safety. The adaptation results in a new state-driven architectural design method. This adaptation means that most of the process steps should be based on or around the statechat semantic.
- It is evidente that a pattern based development of the reference architecture can support the development and application process of the product lines. In this context ***a new statechart-based safety design pattern*** has been developed (Chapter 5). The proposed design pattern is called ***safety design***

pattern of statechat. This pattern extends capabilities of both the statecharts design patterns and safety patterns. The pattern allows an object to alter its behavior and change its internal state when there is a safety violation, and to protect it from introducing in unsafe states. The result is an object-oriented design pattern which handles the safety attribute. To extend the statechart pattern to capture the variability existing in the SPLs, and because the complexities exist in the PL the thesis proposed using of parameterization approach (proposed by Gomaa, (Gomaa, 2011)).

- As there is a tight interplay between safety and security, and in order to address the influence of the security issues in the safety design using patterns, ***a pattern development approach*** that interlinks safety and security patterns is proposed (Chapter 6). This pattern approach is then used to enhance our proposed safety design pattern of statechart (presented in Chapter 5) to address the security in the pattern (see Sec. 6.4). This developed version is considered as ***a new safety and security pattern***.
- To evaluate our work we have defined ***a simplified safety assessment model*** (Chapter 7) which is used to evaluate the safety improvement in the design of the SPLA after using the proposed safety design pattern.
- Finally, In order to show the applicability of our work as well as evaluate it, ***two case studies*** are presented (Chapter 8).

9.3 Future Work

This section discusses future work to improve or complement the research in this thesis. There are several potential courses for furthering the research presented in this thesis, some of which are discussed in this section. The following research and development topics are good candidates for future work:

- The work proposed a method for safety-driven software product line architecture design (SSPLA). The thesis has introduced some efforts to make the make this method more efficient (e.g. making the process activities compatible and consistent). For future work, more efforts are

needed, for example addressing of the other product line aspects (like reusability and maintainability).

- We have developed a new safety-driven design pattern of statechart. The main idea of this new pattern is combining the concepts of the traditional safety patterns with the concepts of statechart patterns. Further research could be conducted to extend the idea of the safety design pattern of statechart to develop a complete product line quality pattern language.
- In this thesis and in order to enhance the safety patterns to address the influence of the security issues on the safety, a pattern development approach that interlinks safety and security patterns is proposed (Chapter 6). The introduced approach for pattern development still needs more work to address other issues such as the reliability, complexity and the impact on execution time.
- In Chapter 7 a new Safety Assessment Model is presented. This model is considered as a state-based safety assessment model. It is a simplified mathematical model for safety assessment of the software product lines architectures. It has been developed to show the safety improvement in the design of the software product line architectures after using our design method, and the safety design pattern. A further work is needed to assess other product line issues such as the complexity and reusability. Also a further research is needed to calculate the safety risk in the software product line architectures.

LIST OF PUBLICATIONS

Refereed Journal Articles:

1. Mozamil Ebnauf, Hany H. Ammar ,"Security and Safety Patterns in Software Product Lines Architectures", Now in the publishing progress.

Mozamil Ebnauf, Hany H. Ammar , "Safety-driven Software Product Line Architectures Design (SSPLA) Methodology for Embedded Systems", the Red Sea University Journal of Basic and Applied Science, Vol.2 Special Issue (1), 18 June 2017.

2. Mozamil Ebnauf, Hany H. Ammar , "Safety-driven software product line architecture design, A survey paper", International Journal of Computer Applications Technology and Research (IJCATR), Volume 5, Issue 10, October 2016.

3. مزمل ابنعوف, هاني عمار, "منهجية التصميم المقاد بالسلامة لمعماريات خط الإنتاج "، المجلة العلمية لاتصالات الجمعية العربية للحاسبات، الجزء الخاص للمؤتمر الدولي لعلوم وهندسة الحاسوب (ICCA 2016) العدد الأول من المجلد التاسع 2016.

Refereed Conference Proceedings:

1. Mozamil Ebnauf, W. Abdelmoez , Aisha Hassan, Hany H. Ammar, M. Abdelhamid, "State-driven Architecture Design for Safety–critical Software Product Lines," in 2019 7th IEEE International Conference on Mechatronics Engineering (ICOM), October 2019, Kuala Lampur, Malaysia.

2. Mozamil Ebnauf, Hany H. Ammar," Safety-driven Software Product Line Architectures Design (SSPLA) Methodology for Embedded Systems" , CSIT 705, First International Conference on Engineering and Applied Sciences, February 2017, Port Sudan, Sudan.

3. Mozamil Ebnauf, Hany H. Ammar, " منهجية التصميم المقاد بالسلامة لمعماريات خط الإنتاج", Paper No 4, International Conference of Computer in Arabic (ICCA 2016), March 2016, Khartoum, Sudan.

REFERENCES

Adamczyk, P. (2003) 'The Anthology of the Finite State Machine Design Patterns', in *Pattern Languages of Programming conference*.

Aleti, A. *et al.* (2012) 'Software Architecture Optimization Methods: A Systematic Literature Review', *IEEE Transactions on Software Engineering*, 99, p. 1. doi:10.1109/TSE.2012.64.

Ammar, H.H. (2013) 'Lectures :The Software Product Line Architectures'.

Amorim, T. *et al.* (2017) 'Systematic pattern approach for safety and security co-engineering in the automotive domain', in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 329–342. doi:10.1007/978-3-319-66266-4_22.

Andrade, H.S. De (2013) 'Software Product Line Architectures : Reviewing the Literature and Identifying Bad Smells', (September), p. 99.

Armoush, A. (2010) *Design Patterns for Safety-Critical Embedded Systems, Thesis*. RWTH Aachen University.

Atkinson, C. and Muthig, D. (2002) 'Component-Based Product-Line Engineering with the UML', in *Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools*. Berlin, Heidelberg: Springer-Verlag (ICSR-7), pp. 343–344.

Bachmann, F. *et al.* (2000) 'The Architecture Based Design Method', p. 61.

Bashroush, R. *et al.* (2017) 'CASE Tool support for variability management in software product lines', *ACM Computing Surveys*, 50(1). doi:10.1145/3034827.

Bass, L.J., Klein, M.H. and Bachmann, F. (2001) 'Quality Attribute Design Primitives and the Attribute Driven Design Method', in *PFE*.

Bayer, J., Flege, O. and Gacek, C. (2000) *Creating Product Line Architectures*.

Behjati, R. *et al.* (2013) 'SimPL: A product-line modeling methodology for families of integrated control systems', *Information and Software Technology*, 55(3), pp. 607–629. doi:https://doi.org/10.1016/j.infsof.2012.09.006.

Bogdanov, K. and Holcombe, M. (2001) *Statechart testing method for aircraft control systems, Test. Verif. Reliab*.

Bosch, J. (2000) *Design & Use of Software Architectures—Adopting and Evolving a Product Line Approach*.

Bures T. et al (2017) 'Software Engineering for Smart Cyber-Physical Systems: Challenges and Promising Solutions', *ACM SIGSOFT S.E Notes. 42. 19-24. 10.1145/3089649.3089656* [Preprint].

Burns, M. (2019) 'Cyber-Physical Systems and Internet of Things NIST Special Publication 1900-202 Cyber-Physical Systems and Internet of Things'.

Buschmann, F. and Maunier, R. (2001) *Pattern-Oriented Software Architecture, Volume 1, Architecture*.

Canterbury Christ Church University (2019) 'Mendeley'. Available at: learning.research.support@canterbury.ac.uk.

Capilla, R. *et al.* (2014) 'An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry', *Journal of Systems and Software*, 91(1), pp. 3–23. doi:10.1016/J.JSS.2013.12.038.

Colanzi, T. and Vergilio, S. (2013) 'Representation of Software Product Line

Architectures for Search-Based Design', in *2013 1st International Workshop on Combining Modelling and Search-Based Software Engineering, CMSBSE 2013 - Proceedings*, pp. 28–33. doi:10.1109/CMSBSE.2013.6604433.

Crnkovic, I. and Stafford, J. (2013) 'Embedded Systems Software Architecture', *Journal of Systems Architecture: the EUROMICRO Journal*, 59, pp. 1013–1014. doi:10.1016/j.sysarc.2013.11.005.

Dabrowski, C. and Hunt, F. (2011) 'Using Markov chain and graph theory concepts to analyze behavior in complex distributed systems', in *The 23rd European Modeling and Simulation Symposium*. Citeseer.

Documentation.help (2022) *No Title, documentation.help*. Available at: https://documentation.help/StarUML/documentation.pdf.

Dong, J. *et al.* (2008) 'The Research of Software Product Line Engineering Process and Its Integrated Development Environment Model', in *2008 International Symposium on Computer Science and Computational Technology*, pp. 66–71. doi:10.1109/ISCSCT.2008.100.

Dr. Ali Assi (2011) *Engineering Education and Research Using MATLAB*. Edited by D.A. Assi. InTech.

Ebnauf, M. and Al., E. (2019) 'State-driven Architecture Design for Safety–critical Software Product Lines', in *2019 7th IEEE International Conference on Mechatronics Engineering (ICOM)*. Putrajaya-Malaysia.

Engström, E. and Runeson, P. (2011) 'Software product line testing–a systematic mapping study', *Information and Software Technology*, 53(1), pp. 2–13.

Fenelon, P. *et al.* (1994) 'Towards integrated safety analysis and design', *ACM SIGAPP Applied Computing Review*, 2(1), pp. 21–32.

Feng, Q. and Lutz, R.R. (2005) 'Bi-directional safety analysis of product lines', *Journal of Systems and Software*, 78(2), pp. 111–127. doi:10.1016/J.JSS.2005.02.028.

Firesmith, D. (2004) 'Engineering Safety Requirements, Safety Constraints, and Safety-Critical Requirements', *J. Object Technol.*, 3, pp. 27–42.

Gama, Helm, Johnson, V. (1995) *Design Patterns Elements of Reusable Object-Oriented Software*. Edited by V. Gama, Helm, Johnson.

Gomaa, H. (2004) *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. 350 Bridge Pkwy suite 208 Redwood City, United State: Addison Wesley.

Gomaa, H. (2011) *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*. Cambridge University Press.

Goseva-Popstojanova, K. *et al.* (2003) 'Architectural-level risk analysis using UML. IEEE Trans Software Eng', *Software Engineering, IEEE Transactions on*, 29, pp. 946–960. doi:10.1109/TSE.2003.1237174.

Hallstrom, J.O., Soundarajan, N. and Tyler, B. (1995) 'Monitoring Design Pattern Contracts'.

Han, K., Weimerskirch, A. and Shin, K. (2014) 'Automotive Cybersecurity for In-Vehicle Communication', *IQT Quarterly*, 6(1), pp. 22–25. Available at: https://kabru.eecs.umich.edu/papers/publications/2014/IQT Quarterly_Summer 2014_Han et al.pdf.

von Hanxleden, R. *et al.* (2014) 'SCCharts: Sequentially Constructive Statecharts for Safety-Critical Applications', *ACM SIGPLAN Notices*, 49, pp. 372–383. doi:10.1145/2666356.2594310.

Harel, D. (1987) 'Statecharts: a visual formalism for complex systems', *Science of*

Computer Programming, 8(3), pp. 231–274. doi:https://doi.org/10.1016/0167-6423(87)90035-9.

Hassan, A., Goseva-Popstojanova, K. and Ammar, H. (2005) 'UML based severity analysis methodology', in, pp. 158–164. doi:10.1109/RAMS.2005.1408355.

Hautamäki, J. (2005) 'Pattern-based tool support for frameworks towards architecture-oriented software development environment'.

Hevner, A.R. *et al.* (2004) 'Design science in information systems research', *MIS quarterly*, pp. 75–105.

Heymans, P. and Trigaux, J.-C. (2003) *Software Product Lines: State of the art*.

Huang, Y. (2013) 'Safety-Oriented Software Architecture Design Approach', *Atlantis Press* [Preprint].

International Organization for Standardization and International Electrotechnical Commission (2005) 'Information technology - Open Distributed Processing - Unified Modeling Language (UML) Version 1.4.2', *Iso/Iec 19501:2005(E)*, 4(1), p. 454. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Unified+Modeling+Language+Specification#2.

Jamal, S. and Eric, S. (2003) *Pattern-Based Approach for Object Oriented Software Design*.

Jan Bosch (2001) 'Software product lines and software architecture design', in *Proceedings of the 23rd International Conference on Software Engineering (ICSE'01)* . LOS ALAMITOS: IEEE (The Institute of Electrical and Electronics Engineers).

javatpoint (2022) *No Title*. Available at: https://www.javatpoint.com/uml-tools.

Jensen, D.C. and Tumer, I.Y. (2013) 'Modeling and Analysis of Safety in Early Design', *Procedia Computer Science*, 16, pp. 824–833. doi:10.1016/J.PROCS.2013.01.086.

John Ryan O'Farrell (2009) *Development of A Software Architecture Method for Software Product Families and its Application to the AubieSat Satellite Program*. Graduate Faculty of Auburn University.

Kang, K.C. *et al.* (1998) 'FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures', *Ann. Softw. Eng.*, 5(1), pp. 143–168.

Karthika, R.A., Rahamtula, S. and Anusha, Y. (2018) 'Internet of things for industrial monitoring and control applications', *International Journal of Engineering and Technology(UAE)*, 7(2.21), pp. 280–282. doi:10.14419/ijet.v7i2.21.12381.

Kasanen, E., Lukka, K. and Siitonen, A. (1993) 'The constructive approach in management accounting research', *Journal of management accounting research*, 5(1), pp. 243–264.

Kassir, A. (2018) *UC Irvine UC Irvine Electronic Theses and Dissertations Title Absorbing Markov Chains with Random Transition Matrices and Applications*. IRVINE. Available at: https://escholarship.org/uc/item/52h7q78h.

Kitchenham, B.A. (2004) 'Procedures for Performing Systematic Reviews', *Keele, UK, Keele University*, 33, pp. 1--26.

Kostakos, V. *et al.* (2016) 'Modelling smartphone usage: A Markov state transition model', *UbiComp 2016 - Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 486–497. doi:10.1145/2971648.2971669.

L. Bass, M. Klein, and F.B. (2002) 'Quality Attribute Primitives and the Attribute Driven Design Method', in *4th International Workshop on Software Product-*

Family Engineering,. Berlin Heidelberg: Ed. Springer.
Len Bass and Paul Clements and Rick Kazman (2003) *Software Architecture in Practice (2nd Edition)*. Addison-Wesley Professional.
Leveson, N. (2011) *Engineering A Safer World, Medicine, conflict, and survival*. London, England: Massachusetts Institute of Technology. doi:10.1080/13623699.2017.1382166.
Leveson, N.G. *et al.* (1991) 'Experiences using statecharts for a system requirements specification', in *Proceedings of the Sixth International Workshop on Software Specification and Design*, pp. 31–41. doi:10.1109/IWSSD.1991.213079.
Leveson, N.G. (1991) 'Software safety in Embedded Computer Systems', *Communications of the ACM*, 34(2), pp. 34–46. doi:10.1145/102792.102799.
Li, L., Safe, L.L. and Université, S. (2018) 'Safe and secure model-driven design for embedded systems To cite this version : HAL Id : tel-01894734 ´ Mod eles Approche Orient ee pour la ´ et la S ecurit S uret e ˆ e Embarqu es'.
Liliana Dobrica, E.N. (2000) 'Attribute-Based Product-Line Architecture Development for Embedded Systems', in *the 3rd Australasian Workshop on Software and Systems Architectures. IEEE*. Sydney, pp. 76–88.
Liu, J., Dehlinger, J. and Lutz, R. (2007) 'Safety analysis of software product lines using state-based modeling', *Journal of Systems and Software*, 80(11), pp. 1879–1892. doi:10.1016/J.JSS.2007.01.047.
M.Sharafi, S.D. (2013) 'A Scenario-Based Approach for Architecture Reconstruction of Product Line'.
Mannion, M., Camara, J. (2004) 'Theorem Proving for Product Line Model Verification', in *Proceedings of the 5th International Workshop on Product Family Engineering (PFE-5)*. Springer, Berlin-Heidelberg, pp. 211-224, pp. 211–224.
Mannion, M. (2002) 'Using First-Order Logic for Product Line Model Validation', in *Proceedings of the Second International Conference on Software Product Lines*. Berlin, Heidelberg: Springer-Verlag (SPLC 2), pp. 176–187.
Mari Matinlassi and Eila Nieme and Liliana Dobrica (2002) *Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture*. Finland: VTT Technical Research Centre of Finland.
McDermid, J. (2002) 'Software Hazard and Safety Analysis', in Damm Wernerand Olderog, E.-R. (ed.) *Formal Techniques in Real-Time and Fault-Tolerant Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 23–34.
Medvidovic, N., Malek, S. and Mikic-Rakic, M. (2003) 'Software Architectures and Embedded Systems', in.
Meister, J., Reussner, R.H. and Rohde, M. (2004) 'Applying patterns to develop a product line architecture for statistical analysis software', *Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, pp. 291–294.
Mozamil Ebnauf and Hany H. Ammar (2017) 'Safety-driven Software Product Line Architectures Design (SSPLA) Methodology for Embedded Systems', *The Red Sea University Journal of Basic and Applied Science*, Vol.2(1).
Mozamil Elgodbe and Ammar, H.H. (2016) *Safety-driven Software Product Line architectures Design, A Survey Paper, International Journal of Computer Applications Technology and Research*. Available at: www.ijcat.com.
Murwantara, I.M. (2012) 'Hybrid ANP: Quality attributes decision modeling of a

product line architecture design', in *2012 2nd International Conference on Uncertainty Reasoning and Knowledge Engineering*. IEEE, pp. 30–34.
Mustapi'c, G. (2004) *Architecting software for complex embedded systems : quality attribute based approach to openness*. Mälardalen University . Available at: http://www.es.mdh.se/publications/667- (Accessed: 20 December 2021).
Nagamine, M., Nakajima, T. and Kuno, N. (2016) 'A case study of applying software product line engineering to the air conditioner domain', in *ACM International Conference Proceeding Series*, pp. 220–226. doi:10.1145/2934466.2934489.
Niaz, I. and Tanaka, J. (2003) 'Code Generation From Uml Statecharts', *Proceedings of the IASTED International Conference on Software Engineering and Applications*, 7.
Nunes, V. *et al.* (2012) 'Variability management of reliability models in software product lines: An expressiveness and scalability analysis', in *2012 Sixth Brazilian Symposium on Software Components, Architectures and Reuse*. IEEE, pp. 51–60.
Obbink, H.T. *et al.* (2000) 'A component-oriented platform architecting method for families of software-intensive electronic prod', in.
Opengatesw.net (2021) *Microsoft Office Excel*. Available at: https://www.opengatesw.net/ms-excel-tutorials/What-is-Excel-Used-For.htm.
Pär J Ågerfalk, B.F.B.L.B.L.L.O. and S.T. (2006) 'Open Source in the Software Product Line: An Inevitable Trajectory', in *10th International Software Product Line Conference*. Baltimore, Maryland, USA.
Parnas, D.L. (1976) 'On the Design and Development of Program Families', *IEEE Transactions on Software Engineering*, SE-2, pp. 1–9.
Paulo Leita˜o, Luis Ribeiro and Thomas Strasser (2016) 'Smart Agents in Industrial Cyber–Physical Systems', in *Proceedings of the IEEE*. IEEE, pp. 1086–1101.
Peter, M. (2011) *Embedded Systems Foundations of Cyber-Physical Systems*. 2nd Edition. Springer Science+Business Media B.V.
Peter Wallin, S.L.J.F.J.A. (2012) 'Problems and their mitigation in system and software architecting', *Information and Software Technology* , 54.
Petersen, K. *et al.* (2008) 'Systematic mapping studies in software engineering', in *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*, pp. 1–10.
Philippow, I. (2003) 'Design pattern recovery in architectures for supporting product line development and application', *... Variability for OO Product ...*, pp. 42–57. Available at: http://www.theoinf.tu-ilmenau.de/~riebisch/home/publ/04-phil.pdf.
Pinzger, M. *et al.* (2004) *Architecture Recovery for Product Families*. Springer_verlag Berlin Heidelberg. doi:10.1007/978-3-540-24667-1_26.
Place, P.R.H. and Kang, K.C. (1993) *Safety-Critical Software: Status Report and Annotated Bibliography*. Pennsylvania.
Pleuss, A. *et al.* (2012) 'Model-Driven Support for Product Line Evolution on Feature Level', *J. Syst. Softw.*, 85(10), pp. 2261–2274. doi:10.1016/j.jss.2011.08.008.
Pohl, K., Böckle, G. and van der Linden, F. (2005) *Software Product Line Engineering, Software Product Line Engineering*. doi:10.1007/3-540-28901-1.
Polzer, A. *et al.* (2012) 'Managing complexity and variability of a model-based embedded software product line', *Innovations in Systems and Software Engineering*, 8(1), pp. 35–49. doi:10.1007/s11334-011-0174-z.
Ramakrishna and Satish et. al. (1996) 'Run time Assertion Schemes for Safety

Critical Systems', in *Ninth IEEE Symposium on Computer Based Medical Systems*. Ann Arbor, Michigan.
Ran, A. (1995) 'MOODS, Models for Object-Oriented Design of State', *PLoPD* [Preprint].
Raspotnig, C., Karpati, P. and Opdahl, A.L. (2018) 'Combined assessment of software safety and security requirements: An industrial evaluation of the CHASSIS method', *Journal of Cases on Information Technology*, 20(1), pp. 46–69. doi:10.4018/JCIT.2018010104.
Rauhamäki, J., Vepsäläinen, T. and Kuikka, S. (2012) 'Architectural patterns for functional safety', in.
Ray, P.P. (2018) 'A survey on Internet of Things architectures', *Journal of King Saud University - Computer and Information Sciences*, 30(3), pp. 291–319. doi:10.1016/j.jksuci.2016.10.003.
Rehn, C. (2009) 'Software Architectural Tactics and Patterns for Safety and Security', *TU Kaiserslautern*, 67663. Available at: http://www.christian-rehn.de/downloads/seminar_safe_sec.pdf.
Robson, C. (2002) *Real world research : a resource for social scientists and practitioner-researchers*. Oxford, UK; Madden, Mass.: Blackwell Publishers.
Sadiku, M.N.O. *et al.* (2017) 'Cyber-Physical Systems: A Literature Review', *European Scientific Journal, ESJ*, 13(36), p. 52. doi:10.19044/esj.2017.v13n36p52.
Schmittner, C. *et al.* (2015) 'A Case Study of FMVEA and CHASSIS as Safety and Security Co-Analysis Method for Automotive Cyber-Physical Systems', in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*. New York, NY, USA: Association for Computing Machinery (CPSS '15), pp. 69–80. doi:10.1145/2732198.2732204.
Shaw, M. (2001) 'The coming-of-age of software architecture research', in *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, pp. 656–664. doi:10.1109/ICSE.2001.919142.
Shaw, M. (2003) 'Writing Good Software Engineering Research Papers: Minitutorial', in *Proceedings of the 25th International Conference on Software Engineering*. USA: IEEE Computer Society (ICSE '03), pp. 726–736.
Shi, J. *et al.* (2011) 'A Survey of Cyber Physical Systems', *Proc. of the Int. Conf. on Wireless Communications and Signal Processing* [Preprint]. doi:10.1109/WCSP.2011.6096958.
Siddiqui, A.S. *et al.* (2017) 'A secure communication framework for ECUs', *Advances in Science, Technology and Engineering Systems*, 2(3), pp. 1307–1313. doi:10.25046/aj0203165.
Sommerville, I. (2009) *Software Engineering*. Nine editi. Addison-Wesley.
Sparx Systems (2016) *UML Models Enterprise Architect User Guide Series CREATED WITH*. Available at: https://sparxsystems.com (Accessed: 20 December 2021).
Surkovi, A. (2018) 'AN ATTACK MODEL OF AUTONOMOUS SYSTEMS OF Dˇ zana Hani ´ Amer Surkovi ´ Supervisor : Aida Cauˇ'.
Svahnberg, M. *et al.* (2003) 'A Quality-Driven Decision-Support Method for Identifying Software Architecture Candidates.', *International Journal of Software Engineering and Knowledge Engineering*, 13, pp. 547–573. doi:10.1142/S0218194003001421.
Swarup, M. Ben and Ramaiah, P.S. (2009) 'A Software Safety Model for Safety

Critical Applications', in.
Systems, I. recommended practice for architectural description of software-intensive (2000) 'IEEE-Std-1471-2000.'
Tan, L, Lin, Y. and Ye, H. (2012) 'Modeling Quality Attributes in Software Product Line Architecture', in *2012 Spring Congress on Engineering and Technology*, pp. 1–5. doi:10.1109/SCET.2012.6341971.
Tan, Lei, Lin, Y. and Ye, H. (2012) 'Quality-Oriented Software Product Line Architecture Design', *Journal of Software Engineering and Applications*, 05, pp. 472–476.
TechTarget (2021) *Microsoft Office Excel*, *Excel*. Available at: https://searchenterprisedesktop.techtarget.com/definition/Excel.
Tetzner, R. (2021) *How to Get Research Published in Journals*. Available at: https://www.linkedin.com/pulse/using-word-its-functions-academic-scientific-writing-rene-tetzner.
Thane, H. (1999) 'Safe and Reliable Computer Control Systems Concepts and Methods', in.
Training.org (2021) *The Importance of MS Excel*, *Training.org*. Available at: https://www.1training.org/blog/importance-ms-excel/.
Unphon, H. (2009) 'Making Use of Architecture throughout the Software Life Cycle - How the Build Hierarchy Can Facilitate Product Line Development', in *Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge*. USA: IEEE Computer Society (SHARK '09), pp. 41–48. doi:10.1109/SHARK.2009.5069114.
Urli, S., Blay-Fornarino, M. and Collet, P. (2014) 'Handling complex configurations in software product lines: A tooled approach', *ACM International Conference Proceeding Series*, 1, pp. 112–121. doi:10.1145/2648511.2648523.
Vaccare Braga, R.T. *et al.* (2012) 'The ProLiCES Approach to Develop Product Lines for Safety-Critical Embedded Systems and its Application to the Unmanned Aerial Vehicles Domain', *CLEI Electronic Journal*, 15(2), pp. 1–13. doi:10.19153/cleiej.15.2.8.
Varshosaz, M. and Khosravi, R. (2013) 'Discrete time Markov chain families: Modeling and verification of probabilistic software product lines', *ACM International Conference Proceeding Series*, (August 2013), pp. 34–41. doi:10.1145/2499777.2500725.
W.R. Dunn (2002) *Practical Design of Safety-Critical Computer Systems*. William Dunn (July 1, 2002).
Wan, J. *et al.* (2011) 'Advances in cyber-physical systems research', *KSII Transactions on Internet and Information Systems (TIIS)*, 5(11), pp. 1891–1908.
Weiss, D.M. and Lai, C.T.R. (1999) *Software Product-Line Engineering: A Family-Based Software Development Process*. USA: Addison-Wesley Longman Publishing Co., Inc.
White, J. *et al.* (2013) 'Evolving feature model configurations in software product lines', *Journal of Systems and Software*, pp. 119–136. doi:10.1016/j.jss.2013.10.010.
Wieringa, R. *et al.* (2006) 'Requirements engineering paper classification and evaluation criteria: a proposal and a discussion', *Requirements engineering*, 11(1), pp. 102–107.
Wu, W. and Kelly, T. (2004) 'Safety tactics for software architecture design', in *Proceedings - International Computer Software and Applications Conference*,

pp. 368–375. doi:10.1109/cmpsac.2004.1342860.
Yacoub, S. and H.A. (1998) ‘A Pattern Language of Statecharts’, in *Third Conference on Pattern Languages of Programming*, p. [1] S. and H. A. Yacoub, “A Pattern Language of St.
Zhu, J. *et al.* (2011) ‘Improving Product Line Architecture Design and Customization by Raising the Level of Variability Modeling’, in, pp. 151–166. doi:10.1007/978-3-642-21347-2_12.