Reliability and Safety of Critical Device Software Systems

Neeraj Kumar Singh

Nancy-Université
Université Henri Poincaré

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

➢ Contact SCD Nancy 1 : theses.sciences@scd.uhp-nancy.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

Nancy-Université
Université Henri Poincaré

UFR STMIA
École Doctorale IAEM Lorraine

Fiabilité et Sûreté des Systèmes Informatiques Critiques

Thèse

présentée et soutenue publiquement le 15 novembre 2011
pour l'obtention du

Doctorat de l'Université Henri Poincaré - Nancy 1
(Mention : informatique)

par

Neeraj Kumar SINGH

Composition du jury :

Rapporteurs :	Prof. Yamine AIT-AMEUR	-	IRIT, ENSEIHT,Université de Toulouse
	Prof. John FITZGERALD	-	Newcastle University
Examinateurs :	Prof. Catherine DUBOIS	-	ENSIEE
	Dr. Yann GUERMEUR	-	DR CNRS LORIA
	Prof. Dominique MÉRY	-	Université Henri Poincaré Nancy 1
	Dr. Isabelle PERSEIL	-	Ingénieur de Recherche, INSERM
	Prof. Olivier ROUX	-	Ecole Centrale de Nantes
Invité :	Prof. Dr. Etienne ALIOT	-	C.H.U de Nancy

Loria Laboratoire lorrain de recherche en informatique et ses applications
Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Dedicated to Lord Krishna, and
My Loving Parents

Résumé

Les systèmes informatiques envahissent notre vie quotidienne et sont devenus des éléments essentiels de chacun de nos instants de vie. La technologie de l'information est un secteur d'activités offrant des opportunités considérables pour l'innnovation et cet aspect paraît sans limite. Cependant, des systèmes à logiciel intégré ont donné des résultats décevants. Selon les constats, ils étaient non fiables, parfois dangereux et ne fournissaient pas les résultats attendus. La faiblesse des pratiques de développement constitue la principale raison des échecs de ces systèmes. Ceci est dû à la complexité des logiciels modernes et au manque de connaissances adéquates et propres. Le développement logiciel fournit un cadre contribuant à simplifier la conception de systèmes complexes, afin d'en obtenir une meilleure compréhension et d'assurer une très grande qualité à un coût moindre. Dans les domaines de l'automatique, de la surveillance médicale, de l'avionique..., les systèmes embarqués hautement critiques sont candidats aux erreurs pouvant conduire à des conséquences graves en cas d'échecs.

La thèse vise à résoudre ce problème, en fournissant un ensemble de techniques, d'outils et un cadre pour développer des systèmes hautement critiques, en utilisant des techniques formelles à partir de l'analyse des exigences jusqu'à la production automatique de code source, en considérant plusieurs niveaux intermédiaires. Elle est structurée en deux parties: d'une part des techniques et des outils et d'autre part des études de cas. La partie concernant des techniques et des outils présente une structure intégrant un animateur de modèles en temps-réel, un cadre de correction de modèles et le concept de charte de raffinement, un cadre de modélisation en vue de la certification, un modèle du coeur pour la modélisation en boucle fermée et des outils de générations automatiques de code. Ces cadres et outils sont utilisés pour développer les systèmes critiques à partir de l'analyse des exigences jusqu'à la production du code, en vérifiant et en validant les étapes intermédiaires en vue de fournir un modèle formel correct satisfaisant les propriétés souhaitées attendues au niveau le plus concret. L'introduction de nouveaux outils concourent à améliorer la vérification des propriétés souhaitées qui sont cachées aux étapes initiales du développement du système. Nous évaluons les propositions faites au travers de cas d'études du domaine médical et du domaine des transports.

De plus, le travail de cette thèse a étudié la représentation formelle des protocoles médicaux, afin d'améliorer les protocles existants. Nous avons complétement formalisé un protocole réel d'interprétation des ECG en vue d'analyser si la formalisation était conforme à certaines propriétés relevant du protocole. Le processus de vérification formelle a mis en évidence des anaomalies dans les protocoles existants. Nous avons aussi découvert une structure hiérarchique pour une interpétation efficace permettant de découvrir un ensemble de conditions qui peuvent être utiles pour diagnostiquer des maladies particulières à un stade précoce. L'objectif principal du formalisme développé est de tester la correction et la consistance du protocole médical.

Abstract

Software systems are pervasive in all walks of our life and have become an essential part of our daily life. Information technology is one major area, which provides powerful and adaptable opportunities for innovation, and it seems boundless. However, systems developed using computer-based logic have produced disappointing results. According to stakeholders, they are unreliable, at times dangerous, and fail to provide the desired outcomes. Most significant reasons of system failures are the poor development practices for system development. This is due to the complex nature of modern software and lack of adequate and proper understanding. Software development provides a framework for simplifying a complex system to get a better understanding and to develop the higher fidelity quality systems at lower cost. Highly embedded critical systems, in areas such as automation, medical surveillance, avionics, etc., are susceptible to errors, which can lead to grave consequences in case of failures.

This thesis intends as a contribute to further the use of formal techniques for the development computing systems with high integrity. Specifically, it addresses the issue that formal methods are not well integrated into established critical systems development processes by defining a new development life-cycle, and a set of associated techniques and tools to develop highly critical systems using formal techniques from requirements analysis to automatic source code generation using several intermediate layers with a rigorous safety assessment approach. The approach has been realised using the Event-B formalism. This thesis has mainly two parts: techniques & tools and case studies. The techniques and tools section consists of a development life-cycle methodology, a framework for real-time animator, refinement chart, a set of automatic code generation tools and formal logic based heart model for close loop modeling. A new development methodology, and a set of associated techniques and tools are used for developing the critical systems from requirements analysis to code implementation, where verification and validation tasks are used as intermediate layers for providing a correct formal model with desired system behavior at a concrete level. We introduce new tools to help to verify desired properties, which are hidden at the early stage of the system development. We also critically evaluate the proposed development methodology and developed techniques and tools through case studies in the medical and automotive domains.

In addition, the thesis addresses the formal representation of medical protocols, which is useful for improving the existing medical protocols. We have fully formalised a real-world medical protocol (ECG interpretation) to analyse whether the formalisation complies with certain medically relevant protocol properties. The formal verification process has discovered a number of anomalies in the existing protocols. We have also discovered a hierarchical structure for efficient ECG interpretation that helps to find a set of conditions that can help to diagnose particular diseases at an early stage. The main objective of the developed formalism is to test correctness and consistency of the medical protocol.

Acknowledgments

This thesis would not have been written without a number of people. I would like to express my deep and sincere gratitude to my supervisor Prof. Dominique Méry for his day-to-day guidance, inspiring discussions, enduring supervision and encouragement. His wide knowledge and his logical way of thinking have been of great value for me. His extensive discussions, critical suggestions and interesting explorations around my work have been very helpful for this study. He has also been a kind and effective advisor, allowing me a great amount of freedom while being actively involved in my research and nudging me in the right directions.

Furthermore, I would like to thank my thesis committee: Dr. John Fitzgerald, Prof. Yamine Ait-Ameur, Prof. Catherine Dubois, Dr. Yann Guermeur, Dr. Isabelle Perseil, Prof. Olivier Roux and Prof. Dr. Etienne Aliot, for their participation and constructive comments.

My sincere thanks also goes to Dr. S. Ramesh, and Dr. Manorajan Satpathi, for offering me the summer internship opportunities in their groups (India Science Lab, General Motor, Bangalore, India) and leading me working on diverse exciting projects. I would like to thank Amel Amblard (Clinical Research Manager, Sorin Group-ELA Medicals) and research team of Sorin Group, Paris to demonstrate current challenging problems and to show the development process of a cardiac pacemaker. Furthermore, I thank Prof. Dr. Etienne Alliot, Head of the Cardiology Department of CHU Nancy, and all his colleagues MD doctors, who share their experiences for implanting a cardiac pacemaker and give me an opportunity to learn about the heart system through their fruitful discussions. Furthermore, I am grateful to cardiologist experts Prof. Yves Juillière (MD, Cardiology) and Dr. Frédérique Claudot (PhD) and biomedical experts Dr. Didier Fass (PhD) of the Université Henri Poincaré Nancy, who shared their experiences for helping to design the methodology and verifying the correctness of proposed approach.

Specially, I would like to extend my gratitude to Dr. John Fitzgerald, Dr. Alan Wassyng and Dr. Didier Fass for their advices in the area of certification and environment modeling. I would also like to extend my gratitude to Prof. Dominique Cansell and Dr. Stephan Merz for their advices and time.

Further, I would like to thank the members of the team Dr. Pascal Fontaine and Dr. Denis Roegel for the valuable suggestions, for answers to theoretical and practical questions and for many interesting discussions. Furthermore, I would like to thank to Dr. J. Paul Gibson who helps to improve the quality of this thesis through careful reading.

I gratefully acknowledge the support of everyone at MOSEL group for extending their cooperation, providing a very stimulating research environment and making my whole stay at LORIA a memorable experience. I wish to thank all my colleagues and staff from the LORIA for their help in technical and nontechnical areas.

A section like this would be incomplete, unless I mention my office-mates and friends Joris, Nazim, Manamiary, Sabina, Mumtaz, Santosh, Hernan, Henri, Thixiyan, Guillaume, Abdu and Crista. I am grateful to all my indian friends from Nancy, especially Ritesh, Ashish, Tushar, Shashi, Shaik, Srikrishna and Utpala for their continuous moral support during my good and bad days and who made my life at Nancy so much diverse and exciting.

Last but not the least, I would like to thank my family and friends in India for helping me to get where I am today. Most of all, I would like to thank my family: my parents and brothers, for loving, inspiring and supporting me spiritually throughout my life.

Contents

0 Présentation de la thèse en français **1**
- 0.1 Motivation . . . 1
- 0.2 Approche . . . 3
- 0.3 Contribution et vue générale de la thèse . . . 6
  - 0.3.1 Contribution . . . 6
  - 0.3.2 Vue générale de la thèse . . . 7
- 0.4 Publications . . . 8
- 0.5 Aperçu des chapitres . . . 10
  - 0.5.1 Etat de l'art . . . 10
  - 0.5.2 Modélisation avec Event-B . . . 10
  - 0.5.3 Méthodologie formellement fondée pour le développement de de systèmes critiques . . . 10
  - 0.5.4 Animateur Temps Réel et Traçabilité des Exigences . . . 10
  - 0.5.5 Chartes de raffinement . . . 10
  - 0.5.6 L'atelier EB2ALL: génération automatique des codes . . . 11
  - 0.5.7 Modélisation du fonctionnement du cœur . . . 11
  - 0.5.8 Etudes de cas . . . 11
- 0.6 Conclusion et Perspectives . . . 12
  - 0.6.1 Contributions . . . 13
  - 0.6.2 Perspectives . . . 18

1 Introduction **21**
- 1.1 Motivation . . . 21
- 1.2 Approach . . . 23
- 1.3 Contribution and Outline . . . 25
  - 1.3.1 Contribution . . . 25
  - 1.3.2 Thesis outline . . . 27
- 1.4 Publications . . . 28

2 State of the Art **31**
- 2.1 Introduction . . . 31
- 2.2 Software in Safety-critical Systems . . . 32
  - 2.2.1 Safety Standards . . . 33
- 2.3 Traditional System Engineering Approach . . . 34
  - 2.3.1 The Software Safety Life-cycle . . . 35
  - 2.3.2 Hazards Analysis . . . 36
  - 2.3.3 Risk Assessment and Safety Integrity . . . 37
  - 2.3.4 Safety Integrity and Assurance . . . 39
- 2.4 Standard Design Methodologies in Development . . . 39
- 2.5 Industrial Application of Formal Methods . . . 40

2.5.1 IBM's Customer Information Control System . . . 40
2.5.2 The Central Control Function Display Information System (CDIS) 40
2.5.3 The Paris Métro Signaling System (SACEM) . . . 41
2.5.4 The Traffic Collision Avoidance System (TCAS) . . . 41
2.5.5 The Rockwell AAMP5 Microprocessor . . . 41
2.5.6 The VIPER Microprocessor . . . 41
2.5.7 The Mondex Electronic Purse . . . 42
2.5.8 The BOS Control System . . . 42
2.5.9 The Intel® Core^{TM} i7 Processor Execution Cluster . . . 42
2.6 Formal Methods for Safety-critical systems . . . 43
2.6.1 Why Formal Methods? . . . 43
2.6.2 Motivation for their use . . . 44

3 The Modeling Framework : Event-B 49
3.1 Introduction . . . 49
3.1.1 Overview of B . . . 49
3.1.2 Proof-based Development . . . 50
3.1.3 Scope of the B Modelling . . . 51
3.1.4 Related Techniques . . . 51
3.2 The Event-B Modelling Notation . . . 52
3.2.1 Contexts . . . 52
3.2.2 Machines . . . 53
3.2.3 Modeling actions over states . . . 53
3.2.4 Proof Obligations . . . 55
3.2.5 Model refinement . . . 56
3.2.6 Tools Environments for Event-B . . . 57

I Techniques and Tools 59

4 Critical System Development Methodology 61
4.1 Introduction . . . 61
4.2 Related Work . . . 63
4.3 Overview of the Methodology . . . 65
4.3.1 Informal Requirements . . . 65
4.3.2 Formal Specification . . . 66
4.3.3 Formal Verification . . . 67
4.3.4 Formal Validation . . . 68
4.3.5 Real-time Animation Phase . . . 68
4.3.6 Code Generation . . . 69
4.3.7 Acceptance Testing . . . 70
4.4 Benefits of Using our Proposed Approach . . . 70
4.4.1 Improving Requirements . . . 70
4.4.2 Reducing Error Introduction . . . 71

4.4.3 Improving Error Detection . . . 71
4.4.4 Reducing Development Cost . . . 71
4.5 Conclusion . . . 71

5 Real-Time Animator and Requirements Traceability 73
5.1 Introduction . . . 73
5.2 Motivation . . . 75
5.2.1 Traceability . . . 77
5.3 Related Work . . . 78
5.4 Animation . . . 79
5.4.1 Benefits of Animation . . . 79
5.4.2 Limitations of Animation . . . 80
5.5 Proposed Architecture . . . 80
5.5.1 Data acquisition & Preprocessing . . . 81
5.5.2 Feature extraction . . . 82
5.5.3 Database . . . 82
5.5.4 Graphical animations tool: Macromedia Flash . . . 82
5.5.5 Animator: Brama plug-in . . . 83
5.5.6 Formal modeling Language: Event-B . . . 83
5.6 Applications and Case Studies . . . 84
5.7 Limitations . . . 84
5.8 Conclusion . . . 85

6 Refinement Chart 87
6.1 Introduction . . . 87
6.2 Related Work . . . 88
6.3 Refinement Chart . . . 89
6.4 Applications and Case Studies . . . 92
6.5 Conclusion . . . 92

7 EB2ALL : An Automatic Code Generator Tool 95
7.1 Introduction . . . 95
7.2 Related Work . . . 97
7.3 A Basic Framework of Translator . . . 98
7.3.1 Selection of a Rodin Project . . . 99
7.3.2 Introduction of a Context File . . . 100
7.3.2.1 Motivation . . . 100
7.3.2.2 Selection of a Context File . . . 100
7.3.2.3 Refinement using a new Context File . . . 101
7.3.3 Generated Proof Obligations . . . 102
7.3.4 Filter Context and Concrete machine Modules . . . 102
7.3.5 Basic Principles of Code Generation . . . 103
7.3.5.1 Process Context and Machine Files using Lexical and Syntax Analysis . . . 104

7.3.5.2 Process Context Files . . . 104
7.3.5.3 Mapping Event-B Constant Types to Programming Language . . . 108
7.3.5.4 Process Machine Files . . . 111
7.3.5.5 Mapping Event-B Variable Types to Programming Language . . . 112
7.3.5.6 Mapping Event-B Events to Programming Language . . . 112
7.3.6 Events Scheduling . . . 119
7.3.7 External Code Injection and Code Verification . . . 122
7.3.8 Compiling and Running the Code . . . 124
7.4 How to use Code Generator plugins . . . 124
7.4.1 Assessment of the Translation tool . . . 124
7.5 Limitations . . . 125
7.6 Conclusions . . . 126

8 Formal Logic Based Heart-Model 127
8.1 Introduction . . . 127
8.1.1 Motivation . . . 128
8.2 Related Work . . . 129
8.3 Background . . . 130
8.3.1 The Heart System . . . 130
8.3.2 Basic overview of Electrocardiogram (ECG) . . . 131
8.3.3 ECG Morphology . . . 132
8.4 Proposed Idea . . . 132
8.4.1 Heart Block . . . 137
8.4.1.1 SA block: . . . 137
8.4.1.2 AV block: . . . 137
8.4.1.3 Infra-Hisian block: . . . 137
8.4.1.4 Left bundle branch block: . . . 137
8.4.1.5 Right bundle branch block: . . . 138
8.4.2 Cellular Automata Model . . . 138
8.5 Functional Formal Modeling of the Heart . . . 141
8.5.1 The Context and Initial Model . . . 141
8.5.2 Abstract Model . . . 142
8.5.3 Refinement 1: Introducing Steps in the Propagation . . . 144
8.5.4 Refinement 2: Impulse Propagation . . . 145
8.5.5 Refinement 3: Perturbation the Conduction . . . 147
8.5.6 Refinement 4: Getting a Cellular Model . . . 150
8.5.7 Model Validation and Analysis . . . 152
8.6 Discussion . . . 154
8.7 Conclusion . . . 154

II Case Studies **157**

9 The Cardiac Pacemaker **161**
9.1 Introduction . . . 161
9.1.1 Why Model-Checker? . . . 163
9.1.2 Related Work for the Cardiac Pacemaker . . . 163
9.2 Basic Overview of Pacemaker system . . . 164
9.2.1 The Heart System . . . 164
9.2.2 The Pacemaker System . . . 165
9.2.3 Bradycardia Operating Modes . . . 166
9.3 Event-B Patterns for Modeling Cardiac Pacemaker . . . 167
9.3.1 Action-Reaction Pattern . . . 167
9.3.2 Time-based Pattern . . . 168
9.4 Refinement Structure of a Cardiac Pacemaker . . . 168
9.5 Development of the Cardiac Pacemaker using Refinement Chart . . . 169
9.6 Formal development of the one-electrode cardiac pacemaker . . . 172
9.6.1 Context and Initial Model . . . 172
9.6.1.1 Abstraction of AOO and VOO modes: . . . 172
9.6.1.2 Abstraction of AAI and VVI modes: . . . 174
9.6.1.3 Abstraction of AAT and VVT modes: . . . 176
9.6.2 First refinement: Threshold . . . 177
9.6.3 Second refinement: Hysteresis . . . 178
9.6.4 Third refinement: Rate Modulation . . . 179
9.7 Formal Development of the Two-Electrode Cardiac Pacemaker . . . 181
9.7.1 Context and Initial Model . . . 181
9.7.1.1 Abstraction of DDD mode: . . . 182
9.7.1.2 Abstraction of DVI mode: . . . 189
9.7.1.3 Abstraction of DDI mode: . . . 189
9.7.1.4 Abstraction of VDD mode: . . . 190
9.7.1.5 Abstraction of DOO mode: . . . 191
9.7.2 First refinement:Threshold . . . 192
9.7.2.1 First refinement of DDD mode: . . . 192
9.7.2.2 First refinement of DVI mode: . . . 195
9.7.2.3 First refinement of DDI mode: . . . 196
9.7.2.4 First refinement of VDD mode: . . . 197
9.7.3 Second refinement of DDD mode: Hysteresis . . . 197
9.7.4 Third refinement: Rate Modulation . . . 198
9.8 Model Validation and Analysis . . . 200
9.9 Closed-Loop Model (Heart & Cardiac Pacemaker) . . . 202
9.9.1 Formal Development of The Cardiac Pacemaker . . . 203
9.9.1.1 Abstract Model: Introducing, Pacing and Sensing Activities with Normal and Abnormal Heart Behavior. . . . 203
9.9.1.2 Refinement 1: Introducing *threshold* in Cardiac Pacemaker and Impulse Propagation in the Heart System. . . . 203

9.9.1.3 Refinement 2: Introduction of Hysteresis for cardiac pacemaker model and Perturbation the Conduction for the heart model. . . . 204
9.9.1.4 Refinement 3: Introduction of Rate Modulation for the Cardiac Pacemaker Model and a Cellular Model for the Heart system. . . . 204
9.10 Real-Time Animation Using Pacemaker Case Study . . . 206
9.11 Code Generation for A Cardiac Pacemaker using EB2ALL Tool . . . 207
9.12 Discussion and Conclusion . . . 210
9.12.1 Discussion . . . 210
9.12.2 Conclusion . . . 211

10 Adaptive Cruise Control (ACC) 215
10.1 Introduction . . . 215
10.2 PID Controller . . . 217
10.3 Overview of Methodology . . . 218
10.4 Informal Description of ACC . . . 220
10.4.1 Basic Components of ACC . . . 221
10.4.2 Basic I/Os of ACC . . . 222
10.5 Development of the ACC System using Refinement Chart . . . 223
10.6 Formal development of the ACC . . . 225
10.6.1 Abstraction of ACC system . . . 225
10.6.2 First Refinement: State-flow Model . . . 228
10.6.3 Second Refinement: Detailed modeling of State-flow . . . 231
10.6.4 Third Refinement: Controller Definition . . . 235
10.7 Model Validation and Analysis . . . 238
10.8 Formal Specification into Simulink . . . 239
10.9 Code Generation for ACC System using EB2ALL . . . 241
10.10Discussion and Conclusion . . . 244
10.10.1 Discussion . . . 244
10.10.2 Conclusion . . . 245

11 Electrocardiogram (ECG) Interpretation Protocol using Formal Methods 249
11.1 Introduction . . . 250
11.2 Related Work . . . 251
11.3 Selection of Medical Protocol . . . 254
11.4 Basic overview of Electrocardiogram (ECG) . . . 255
11.4.1 Differentiating the P-, QRS- and T-waves . . . 255
11.5 Formal Development of the ECG Interpretation . . . 256
11.5.1 Abstract Model : Assessing Rhythm and Rate . . . 256
11.5.2 Overview of the Full Refinement Chain . . . 262
11.5.2.1 First Refinement : Assess Intervals and Blocks . . . 262

11.5.2.2 Second Refinement : Assess for Nonspecific Intraventricular Conduction Delay and Wolff-parkinson-white Syndrome . . . 264
11.5.2.3 Third Refinement : Assess for ST-segment Elevation or Depression . . . 265
11.5.2.4 Fourth Refinement : Assess for Pathologic Q-wave . . . 266
11.5.2.5 Fifth Refinement : P-wave . . . 268
11.5.2.6 Sixth Refinement : Assess for left and right ventricular hypertrophy . . . 268
11.5.2.7 Seventh Refinement : Assess T-wave . . . 269
11.5.2.8 Eighth Refinement : Assess Electrical Axis . . . 270
11.5.2.9 Ninth Refinement: Assess for Miscellaneous Conditions . 271
11.5.2.10 Tenth Refinement: Assess Arrhythmias . . . 272
11.6 Statistical Analysis and Lesson Learned . . . 274
11.6.1 Statistical Analysis . . . 274
11.6.2 Lesson learned . . . 275
11.6.2.1 Ambiguous . . . 276
11.6.2.2 Inconsistencies . . . 277
11.6.2.3 Incompleteness . . . 277
11.7 Conclusion . . . 277

12 Conclusion and Outlook 281
12.1 Contributions . . . 281
12.2 Consequences and Future Challenges . . . 286

A Certification Standards 289
A.1 What is Standards? . . . 289
A.2 ISO/IEC Standards . . . 290
A.2.1 IEC 61508 - Software Safety in E/E/EP Systems . . . 291
A.2.2 IEC 62304 - Process Requirements for Medical Device Software . . 292
A.3 IEEE Standards Association . . . 293
A.3.1 IEEE 1012-yyyy . . . 294
A.3.2 IEEE 730-yyyy . . . 294
A.3.3 IEEE 1074-yyyy . . . 294
A.4 FDA . . . 294
A.5 Common Criteria . . . 296
A.5.1 CC Evaluation Assurance Level (EAL) . . . 296

Bibliography 299

CHAPTER 0

Présentation de la thèse en français

"A perfection of means, and confusion of aims,
seems to be our main problem."
(Albert Einstein)

L'intégration au développement d'un système critique d'une méthode formelle offre un potentiel important, pour améliorer la fiabilité et faciliter l'obtention d'une certification. Cette thèse propose un cadre de développement et un ensemble d'outils et de techniques en vue du développement de systèmes hautement critiques, en utilisant des approches fondées sur la vérification et la validation formelles, en partant de l'analyse des exigences vers la génération de code source; cette méthodologie repose sur la production de plusieurs niveaux de modélisation. L'objectif principal de cette thèse est de faire progresser l'utilisation des techniques formelles pour le développement de systèmes logiciels critiques. De manière plus spécifique, cette thèse part du constat que les méthodes formelles n'ont pas été assez bien intégrées dans le processus de développement des systèmes critiques. Elle suggère une démarche pour un nouveau cycle de développement et propose un ensemble de techniques et d'outils associés pour concevoir des systèmes hautement critiques, en utilisant des techniques formelles depuis les exigences jusqu'à la production automatique du code; ce processus est fondé sur l'utilisation de plusieurs niveaux intermédiaires avec une approche rigoureuse établissant la sûreté. Cette approche a été validée en utilisant le formalisme Event B et des outils, et, en y ajoutant quelques nouveaux aspects comme l'animateur temps-réel, les chartes de raffinement, un ensemble d'outils de traduction de modèles Event B en un langage exécutable et une méthodologie de développement de systèmes critiques. La pertinence de cette démarche a été évaluée sur des études de cas du domaine médical et du domaine des transports. Dans ce chapitre, nous présentons les motivations de ce travail et les concepts principaux de notre démarche proposée pour développer une nouvelle méthodologie de développement de systèmes, ainsi que les techniques et outils associés.

0.1 Motivation

De nos jours, les systèmes logiciels ont envahi notre vie quotidienne de différentes façons. La technologie de l'information est un domaine majeur avec des opportunités favorisant

grandement l'innovation. Cependant, les systèmes logiciels développés produisent des résultats décevants et font défaut dans la réalisation du système en fonction de son cahier des charges; cette situation les rend non fiables et *de facto* dangereux. Une des causes de cet échec est la pauvreté des pratiques [Leveson 1993; Zhang 2010; Gibbs 1994; Price 1995; Yeo 2002] de développement; ceci est dû à la nature complexe des logiciels actuels et au manque de compréhension du processus de développement. Le développement logiciel propose un cadre pour simplifier la compréhension du système complexe et de développer des systèmes avec une plus grande fidélité à un coût moindre. Les systèmes hautement critiques embarqués tels que les systèmes des transports, ou de la médecine ou de l'avionique, sont sujets à des erreurs qui ne sont pas supportables en cas de problèmes. Tout problème dans ces systèmes peut conduire à deux types de conséquences directes ou indirectes. Les conséquences directes peuvent conduire à des pertes financières, de propriétés ou des atteintes aux personnes, etc. En plus de ces éléments, les entreprises en question peuvent perdre la confiance des clients dans le cas d'un problème sur un produit. Dans ce contexte, un haut degré de sûreté et de sécurité est exigé, pour considérer les systèmes critiques. Un système est supposé accomplir une tâche de manière sûre en présence de problèmes. Un raisonnement formel et rigoureux sur les algorithmes et sur les mécanismes sous-jacents de tels systèmes est exigé pour comprendre précisément le comportement du système à un niveau de la conception. Cependant, le développement de systèmes fiables est une tâche significativement complexe, qui impacte la fiabilité d'un système.

Le développement [Woodcock 2007; Gaudel 1996; Jetley 2006] fondé sur les méthodes formelles est une approche très commune et populaire qui permet de gérer la complexité croissante d'un système avec l'assurance de la correction dans le cadre du génie logiciel moderne. Les techniques formelles maîtrisent, de plus en plus, les fonctionnalités associées à la criticité de la sûreté des systèmes hautement critiques. Ces techniques sont aussi considérées comme un moyen de remplir les exigences de certificats [ISO ; IEEE-SA ; CC ; FDA] de standards en vue d'évaluer un système critique avant son utilisation pratique. De plus, les systèmes critiques peuvent être effectivement analysés à des étapes précoces du développement, ce qui permet d'explorer les erreurs conceptuelles, les ambiguïtés, la correction des exigences et du flôt de conception avant une mise en œuvre du système effectif. Cette approche permet de corriger des erreurs plus facilement et à moindre coût.

Nous formulons les objectifs suivants en lien avec le développement d'un système critique et nous proposons un méthodologie de développement formel et un nouvel ensemble de techniques et d'outils pour remplir ces objectifs:

- L'établissement d'une théorie unifiée pour le développement de systèmes critiques.
- Construire un ensemble complet et intégré d'outils pour les systèmes critiques qui soutiennent les activités de vérification, y compris la spécification formelle, la validation des modèles en temps réel, d'animation et de génération automatique de code.
- Modélisation de l'environnement pour le développement du système en boucle fermée à des fins de vérification.
- Développement fondé sur le raffinement afin de produire des modèles moins erronés,

des spécifications plus faciles des systèmes critiques et de réutiliser de telles spécifications pour des conceptions futures.

- Décomposition du système complexe en différents sous-systèmes indépendants et recomposition du système après validation.
- Développement basé modèle et cadres de conception basée sur des composants.
- L'intégration systémique des éléments critiques et la possibilité d'annoter les modèles à des fins diverses (par exemple, orienter la synthèse des outils de vérification).
- Certification fondée sur l'animation
- Exigences et métriques pour la certification et la sûreté.

Les objectifs énumérés sont couverts dans cette thèse par le biais développement d'une nouvelle méthodologie dy cycle de développement et un ensemble de techniques et d'outils associés pour le développement des systèmes critiques. La méthodologie du cycle de développement est un processus de développement pour les systèmes, afin de capter les caractéristiques essentielles précisément d'une manière intuitive. Une nouvelle série de techniques et d'outils et une méthodologie de développement sont développées pour gérer les besoins courants, les spécifications du système fondées sur le raffinement, la vérification, l'annotation des modèles utilisant un ensemble de données temps réel grâce à un animateur temps réel et finalement la génération automatic de codes à partir de la spécification formelle vérifiée. Toutes ces approches peuvent aussi aider à obtenir des certificats des normes internationales [ISO ; IEEE-SA ; CC ; FDA].

0.2 Approche

Dans cette thèse, nous présentons une méthodologie du cycle de vie de développement et un ensemble de techniques et d'outils associés, en vue de la conception de systèmes hautement critiques; cette approche est fondée sur des techniques formelles depuis les exigences jusqu'à la mise en œuvre du code. Ce document est structuré en deux parties: d'une part des techniques et des outils et d'autre part des étdues de cas. La partie sur les techniques et les outils comprend la description de la méthodologie du cycle de développement, un cadre pour l'animation temps-réel [Méry 2010b], le concept de charte de raffinement [Méry 2011e], un ensemble d'outils de génération automatique de codes exécutables [Méry 2010a; Méry 2011b; EB2ALL 2011] et un modèle logico-mathématique du cœur pour une modélisation en boucle fermée [Méry 2011f; Méry 2011i]. La méthodologie de développement et les outils associés sont utilisés pour développer un système critique depuis les exigences jusqu'à la génération de code exécutable; les tâches de vérification et de validation sont réalisées à des niveaux intermédiaires, afin de fournir un modèle formel correct satisfaisant les comportements attendus exigés au niveau concret. L'introduction de nouveaux outils fournit une aide à la vérification des propriétés attendues qui sont cachées pour les étapes initiales du dévelopement du système. Par exemple, un animateur temps-réel donne un moyen pour découvrir des exigences cachées; c'est une technique efficace

pour utiliser l'ensemble des données temps-réel dans un modèle formel, sans devoir générer le code source dans un langage de programmation cible[Méry 2010b]; c'est aussi un moyen pour communiquer avec des experts du domaine (par exemple des experts médicaux) participant au processus de dévelopement du système (développement d'un système médical). Dans le paragraphe suivant, nous donnons la description de la méthoddologie de développement et toutes les techniques et tous les outils associés.

La nouvelle méthodologie est une extension du modèle waterfall [Acuña 2005; Bell 1993; Schumann 2001; Wichmann 1992] et s'appuie sur des techniques formelles, afin de la lier à des approches rigoureuses en vue de la production de systèmes critiques fiables. Elle combine le raffinement avec la vérification, l'animation temps-réel et finalement la production automatique de code source. Le processus de développement de système est apprécié parallèlement par une approche d'évaluation de la sûreté [Leveson 1991] pour être conforme aux standards de certification. La méthodologie du cycle de vie comprend sept étapes principales: en premier lieu, les exigences informelles résultant d'une version structurée des exigences où chaque fragment est classé en fonction d'une taxinomie fixée. Dans la seconde phase, les exigences informelles sont représentées dans un langage de modélisation formelle disposant d'une sémantique précise et ces exigences sont enrichies de contraintes temporelles et d'invariants. La troisième étape comprend une vérification formelle s'appuyant sur le raffinement en vue de vérifier la consistance interne et la correction des spécifications. L'étape quatre est une étape qui détermine en quoi un modèle formel est une représentation précise du monde réel dans la perspective offerte par l'emploi d'un model checker. La cinquième étape est une animation du modèle sur des données réelles plutôt que des données jouets et offre une façon simple aux spécifieurs de construire une visualisation fondée sur le domaine des experts; ces mêmes experts peuvent vérifier si le modèle formel construit correspond à leurs attentes. La sixième étape produit le code source à partir des modèles formels vérifiés et validés et finalement une dernière étape est dédiée à l'acceptation du système en produisant les tests nécessaires. Ce type d'approche est très utile pour vérifier des propriétés complexes d'un système et pour découvrir des problèmes potentiels comme le blocage ou la vivacité au cours de phases amont du développement du système.

Selon le cycle de vie de développement d'un système critique, tout d'abord nous mettons l'accent sur la traçabilité des exigences en utilisant un animateur en temps réel [Méry 2010b]. La modélisation formelle des exigences est une tâche difficile, qui est utilisée pour le raisonnement dans les premières phases du développement du système et permet de s'assurer de l'exhaustivité, de la cohérence, et de la vérification des exigences. L'animation temps réel d'un modèle formel a été reconnue pour être une approche prometteuse pour soutenir le processus de validation du cahier des charges. Le principe est de simuler les comportements souhaités d'un système donné en utilisant des modèles formels dans l'environnement en temps réel et de visualiser la simulation dans une certaine forme attrayante pour les intervenants. L'environnement en temps réel contribue à la construction, la clarification, la validation et la visualisation d'une spécification formelle. Un tel type d'approche est également utile pour la certification fondée sur des preuves.

Les techniques de raffinement [Abrial 2010; Abrial 1996a; Back 1979] ont un rôle-clé dans la modélisation d'un système complexe avec une approche incrémentale. Une charte

de raffinement est une représentation graphique d'un système complexe utilisant une approche en couches, où des blos fonctionnels sont décomposés en plusieurs sous-blocs à un nouveau niveau de raffinement, sans changer le comportement du système initial raffiné. Le but final est d'utiliser ces chartes de raffinement pour obtenir une spécification assez détaillée pour être implantée mais aussi de décrire correctement les comportements du système. Le propos des chartes de raffinement est de fournir une représentation facilement gérable pour les différents raffinements des systèmes. La charte de raffinement offre une vue claire dans l'assistance à l'intégration des systèmes; elle permet aussi de faciliter l'assemblage d'un système fondé sur des modes et différentes sortes d'éléments. Ceci est un aspect important non seulement pour développer des garanties de correction et de performances mais aussi pour assembler des composants avec un coût connu.

Une autre étape dans le cycle de développement du logiciel est l'implantation du code. Dans ce contexte, nous avons développé un outil de génération automatique de codes [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b] produisant un code efficace dans un langage de programmation cible (C, C++, Java and C#) à partir de modèles formels Event-B en lien avec l'analyse de problèmes complexes. Cet outil est une collection d'appliquettes, qui sont utilisées pour la traduction des modèles formels Event-B dans différents langages de programmation. L'outil de traduction est développé rigoureusement avec préservation des propriétés de sûreté. Nous présentons une architecture du processus de traduction pour produire un code du langage cible à partir des modèles Event-B; la tradution utilise une grammaire au travers d'une transformation dirigée par la syntaxe, pour produire une architecture ordonnançant le code et assurant la vérification du code produit.

Un modèle en boucle fermée d'un système est considéré comme un *de facto* standard pour les systèmes critiques dans les domaines médical, l'avionique et l'automobile pour valider le modèle du système à un stade précoce du développement du système, qui est un problème ouvert dans le domaine de la modélisation. Les stimulateurs cardiaques et défibrillateurs cardioverteurs implantables (DCI) sont les principaux dispositifs médicaux critiques, ce qui nécessitera une modélisation boucle fermée (intégration du système et de son environnement) à des fins de vérification, afin d'obtenir un certificat auprès d'organismes de certification. Dans ce contexte, nous proposons une méthodologie pour modéliser un système biologique, comme le cœur, pour la modélisation d'un environnement biologique. Le modèle de cœur est principalement basée sur l'analyse électrocardiographique, qui modélise le système cardiaque au niveau cellulaire. L'objectif principal de cette méthode est de modéliser le système cardiaque et de l'intégrer avec le modèle de dispositifs médicaux comme les stimulateurs cardiaques pour spécifier un modèle en boucle fermée. Les industriels s'affrontent pour un tel type d'approche depuis lontemps et visent à valider un modèle de système avec un environnement virtuel biologique.

L'évaluation du cadre proposé et les techniques et outils sont donnés sous la forme du développement de deux études de cas dans différents domaines: d'une part liés à la médecine et d'autre part liés à l'automobile. Le challenge du stimulateur cardiaque est liée au domaine médical et le régulateur de vitesse adaptatif (ACC) est lié au domaine automobile. Ces deux études de cas évaluent les principales caractéristiques clés des cadres proposés et des outils développés [Méry 2011g; Méry 2009; Méry 2010c] allant de l'analyse des besoins à la mise en œuvre du code.

Les techniques formelles ne sont pas seulement utiles pour les systèmes critiques, mais aussi elles sont aussi très utiles pour vérifier les propriétés de sécurité requises dans d'autres domaines, par exemple dans le domaine clinique pour vérifier l'exactitude des protocoles et des lignes directrices [Méry 2011h; Méry 2011j] . Des directives cliniques apportent une aide systématique aux praticiens qui soignent dans des circonstances cliniques. Aujourd'hui, un nombre important de directives et protocoles comportent des défauts ou des imprécisions. En effet, l'ambiguïté et l'incomplétude sont plus susceptibles d'anomalies dans les pratiques médicales. Notre objectif principal est de trouver les anomalies et d'améliorer la qualité des protocoles médicaux utilisant des techniques formelles, telles que B événementiel. Dans cette étude, nous utilisons le langage de modélisation Event-B pour capturer des directives à valider, pour améliorer les protocoles. Une évaluation de cette expérience est donnée par une étude de cas, par rapport à un protocole de la vie réelle de référence (Interprétation ECG) qui couvre une grande variété de caractéristiques du protocole concernant les maladies cardiaques.

0.3 Contribution et vue générale de la thèse

0.3.1 Contribution

Les contributions suivantes ont été faites dans cette thèse. Nous proposons une méthodologie de développement et un ensemble de techniques et d'outils associés pour développer des systèmes hautement critiques; cette méthodologie est fondée sur des approches de vérification et de validation à partir de l'analyse des exigences jusqu'à la génération automatique de code source, en utilisant des pas intermédiaires [Méry 2011e; Méry 2010d; Méry 2011g; Méry 2011f; EB2ALL 2011; Méry 2011a; Méry 2011b]. La partie des techniques et d es outils comprend une méthodologie de développement [Méry 2010d], un animateur temps-réel [Méry 2010b], le concept de charte de raffinement [Méry 2011e], des outils de génération automatique de codes [Méry 2010a; Méry 2011c; Méry 2011b] et un modèle du cœur fondé sur une logique formelle [Méry 2011f; Méry 2011i]. Notre approche de la spécification et de la vérification met en œuvre les techniques d'abstraction et de raffinement.

Cette technique utilisant Event-B, consiste à décrire rigoureusement le problème dans un modèle abstrait et de faire apparaître la solution ou les détails de conception au cours des pas de raffinement. Ici, nous présentons un développement incrémental et prouvé, pour modéliser et vérifier de telles contraintes de nature interdisciplinaire en Event-B [Cansell 2007; Abrial 2010]. Par le raffinement, nous vérifions que la conception détaillée d'un système dans le raffinement, est conforme aux modèles initiaux abstraits. Notre approche est fondée sur le langage de modélisation formelle Event-B supporté par la plate-forme Rodin qui intègre des outils, pour prouver les modèles et le raffinement des modèles; de plus, nous utilisons l'outil ProB [ProB ; Leuschel 2003] pour l'analyse des modèles et leur validation. L'outil Rodin génère des obligations de preuve liant les variables abstraites et les variables concrètes pour vérifier le raffinement. Nous avons utilisé les outils de preuve de Rodin [RODIN 2004] à la fois pour les générer mais aussi pour les vérifier soit automatiquement soit interactivement. La preuve de ces obligations de preuve nécessite l'addition

d'invariants de collage à ces modèles. Ces invariants de collage attestent de la relation entre les variables abstraites et les variables concrètes. La découverte de ces nouveaux invariants de collage apporte un éclairage au système et contribue au raisonnement visant à comprendre pourquoi une solution spécifique proposée au cours d'un raffinement, est une solution correcte du problème abstrait. Les méthodes formelles sont habituellement employées dans l'analyse des hypothèses, des relations et des exigences du système.

La méthodologie de cycle de développement proposée et des techniques et des outils associés [Méry 2010b; Méry 2010d; Méry 2011e; EB2ALL 2011; Méry 2011f; Méry 2011b] ont été évalués sur deux types de systèmes critiques: un stimulateur cardiaque et une système d'assistance au contrôle de la vitesse. Le stimulateur cardiaque a été proposé comme un Grand Challenge [Hoare 2009; Woodcock 2007] dans le cadre de l'action *Verified Software Initiative.* ACC est une application qui a été proposée par les laboratoires India Science Lab, General Motor. Les deux systèmes sont complexes et constituent un véritable challenge dans deux domaines applicatifs différents: médical et transport. Ces deux études de cas évaluent les éléments clés des cadres proposés et des outils développés [Méry 2011g]. Les techniques de modélisation fondées sur le raffinement réduisent l'effort de vérification de manière significative en concevant le système global par l'emploi du raffinement incrémental et chaque pas de raffinement est vérifié en utilisant un assistant de preuve, un model checker et des outils d'animation. Nous avons aussi utilisé un modèle de l'environnement pour le système cardiaque en vue d'obtenir un système à boucle fermée intégrant le pacemaker. Un animateur temps-réel [EB2ALL 2011; Méry 2011b] est utilisé pour vérifier les modèles formels et les comportements véritables du système selon les experts du domaine. De plus, un outil de génération de code permet de produire le code source à partir du modèle formel prouvé du système pour une implantation finale. La vérification formelle et la validation du modèle conduisent à la conformité selon les directives de qualité de FDA's QSR, ISO, IEEE, CC [ISO ; CC ; IEEE-SA ; FDA]. Selon FDA QSR, la validation est définie comme suit *confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use can be consistently fulfilled*. La vérification est définie comme suit: *confirmation by examination and provision of objective evidence that specified requirements have been fulfilled* [NITRD 2009; Méry 2010d].

De plus, nous avons modélisé le protocole médical d'analyse des ECG [Méry 2011h]. Le modèle formel du protocole est vérifié; ce modèle est non seulement effectif mais aussi utile pour améliorer le protocole médical existant. La vérification a découvert une structurev hiérarchique pour l'analyse des ECG; cette structure peut être efficacement utilisée pour découvrir un ensemble de conditions très utiles pour diagnostiquer une maladie particulière à un stade précoce, sans avoir à recourir à des diagnostics multiples.

0.3.2 Vue générale de la thèse

La thèse est organisée en 12 chapitres. Le chapitre 2 dresse un état de l'art et dresse une liste non-exhaustive d'applications industrielles des méthodes formelles. Le chapitre 3 décrit les techniques de modélisation utilisant le langage de modélisation Event-B et ensuite ce document est divisé en deux parties.

La partie 2 débute par le chapitre 4 décrivant la méthodologie de développement des systèmes critiques et décrit les différentes étapes de cette méthode. Dans le chapitre 5, l'architecture de validation sur des données réelles est décrite en détail; cette section aborde la question de l'animation temps réel et de la traçabilité des exigences. Puis, le concept de charte de raffinement est introduit et illustré. Le chapitre 7 apporte une description détaillée des outils de transformations des modèles Event-B en codes exécutables C, C++, Java ou C#; cet ensemble de transfirmations est présenté dans un outil appelé EB2ALL [EB2ALL 2011; Méry 2011d; Méry 2010a; Méry 2011c; Méry 2011b]. Enfin, cette première partie se termine par la description d'un modèle original du cœu en interprétant les signaux électriques et la polarité du cœur; dans ce chapitre, nous modélisons un système biologique, le cœur, en vue de décrire l'environnement biologique; ce modèle est fondé sur une approche à base d'automate cellulaires.

La seconde est consacrée à la présentation des deux études de cas complètes: l'une sur le pacemaker et l'autre sur le système de contrôle automatique de la vitesse; une étude de cas complète produit un modèle formel du protocole d'analyse des ECG et de diagnostics des maladies du cœur. Ainsi, dans le chapitre 9, nous présentons le développement formelle complet d'un pacemaker en utilisant les techniques et les outils présentées auparavant. Puis, le chapitre 10 présente un développement formel du système de contrôle de vitesse. De plus, dans cette derbière étude, nous produison un modèle *simulonk* à partir des modèles formels prouvés. Le chapitre 11 concerne le développement d'un modèle pour le protocole médical de diagnostic associé à l'interprétation des ECGs et cette étude montre comment les modèles formels peuvent être développés et exploités pour évaluer la qualité des protocoles médicaux. Dans le chapitre 12, une conclusion termine l'exposé et des annexes sur la certification sont placées pour que le lecteur puisse les consulter.

0.4 Publications

Les résultats présentés dans ce travail ont été publiés dans les documents suivants:

Journaux Internationaux

- D. Méry and **N. K. Singh** "*Formal Specification of Medical Systems by Proof-Based Refinement*", ACM Transaction on Embedded Computing Systems, May, 2011, (*in Press*).
- D. Méry and **N. K. Singh** "*Functional behavior of a cardiac pacing system*", International Journal of Discrete Event Control System, 2010.
- D. Méry and **N. K. Singh** "*A Generic Framework: from Modeling to Code*", Fourth IEEE International workshop UML and Formal Methods (UML&FM'2011), FM'2011, 20 June 2011, *(To appear in special issue of ISSE NASA journal, Innovations in Systems and Software Engineering)*.

Conférences Internationales

- D. Méry and **N. K. Singh** "*Automatic Code Generation from Event-B Models*", Proceedings of the 2011 Symposium on Information and Communication Technology,

SoICT 2011, Hanoi, Vietnam, ACM, ACM International Conference Proceeding Series, October 13-14, 2011. (*Accepted*).

- D. Méry and **N. K. Singh** "*Analysis of DSR protocol in Event-B*", 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2011), Springer LNCS,10-12 October 2011.

- D. Méry and **N. K. Singh** "*Formal Development and Automatic Code Generation : Cardiac Pacemaker*", International Conference on Computers and Advanced Technology in Education (ICCATE 2011), Communications in Computer and Information Science (CCIS), Springer, 3-4 November 2011 (*Accepted*).

- D. Méry and **N. K. Singh** "*EB2J : Code Generation from Event-B to Java*", 14th Brazilian Symposium on Formal Methods (SBMF 2011), 26-30 September 2011, (*Short Paper*) (*Accepted*).

- D. Méry and **N. K. Singh** "*Formalisation of the Heart based on Conduction of Electrical Impulses and Cellular-Automata*", International Symposium on Foundations of Health Information Engineering and Systems (FHIES 2011), Springer LNCS, 29-30 August 2011.

- D. Méry and **N. K. Singh** "*Medical Protocol Diagnosis using Formal Methods*", International Symposium on Foundations of Health Information Engineering and Systems (FHIES 2011),Springer LNCS, 29-30 August 2011.

- D. Méry and **N. K. Singh** "*Trustable Formal Specification for Software Certification*", in Proceeding ISoLA (2), Springer LNCS,Vol-6416, 312-326, 2010.

- D. Méry and **N. K. Singh** "*Real-time animation for formal specification*", in Proceeding Complex Systems Design & Management, Springer-Verlag, Paris, 27-29 October, 2010.

- D. Méry and **N. K. Singh** "*EB2C : A Tool for Event-B to C Conversion Support*", Poster and Tool Demo submission, and published in a CNR Technical Report in SEFM 2010.

Rapports Techniques

- D. Méry and **N. K. Singh** "*Technical Report on Formalisation of the Heart using Analysis of Conduction Time and Velocity of the Electrocardiography and Cellular-Automata* ", Technical Report (http://hal.inria.fr/inria-00600339/en/), 2011.

- D. Méry and **N. K. Singh** "*Technical Report on Interpretation of the Electrocardiogram (ECG) Signal using Formal Methods*", Technical Report (http://hal.inria.fr/inria-00584177/en/), 2011.

- D. Méry and **N. K. Singh** "*Pacemaker's Functional Behavior in Event-B*", Technical Report (http://hal.inria.fr/inria-00419973/en/), 2009.

- D. Méry and **N. K. Singh** "*Formal Development of Two-Electrode Cardiac Pacing System*" Technical Report, (https://hal.archives-ouvertes.fr/inria-00465061/en/), 2010).

0.5 Aperçu des chapitres

Nous donnons un aperçu court de chaque chapitre du document de cette thèse.

0.5.1 Etat de l'art

Dans ce chapitre, nous dressons un état de l'art sur les logiciels dans les systèmes critiques et nous donnons à la fois une description des approches traditionnelles utilisées dans les approches tradionelles de l'ingénierie des systèmes; ainsi que des exemples industriels d'utilisation de méthodes et de techniques formelles. Nous apportons une motivation pour l'emploi de telles méthodes et nous donnons aussi des éléments sur les standards liés à la certification.

0.5.2 Modélisation avec Event-B

Dans ce chapitre, nous introduisons les notations et les concepts de la méthode Event-B introduite et développée par Jean-Ramond Abrial. Nous insistons sur la notion de raffinement qui est un concept central de cette méthode et nous donnons aussi des éléments à développer.

0.5.3 Méthodologie formellement fondée pour le développement de de systèmes critiques

Nous décrivons la méthodologie proposée dans le diagramme de la figure 1 et nous schématisons globalement sa structre et son utilisation. Elle sera illustrée par desu cas d'études significatifs: le pacemaker et le systèmes d'assitance au contrôle de la vitesse.

0.5.4 Animateur Temps Réel et Traçabilité des Exigences

Ce chapitre propose une utilisation originale des modèles formels et des outils existant en les combinant pour obtenir une architecture 2 permettant de visualiser le comportement des systèmes décrits par des modèles formels mais sur des données réelles, afin de donner une vue lisible par les experts du domaine: en l'occurence les experts médicaux.

0.5.5 Chartes de raffinement

Nous avons proposé un concept nouveau, celui de charte de raffinement, qui permet de structurer les modèles développés et de simplifier leur développement 3, notamment par rapport aux systèmes modaux ou à modes de fonctionnement.

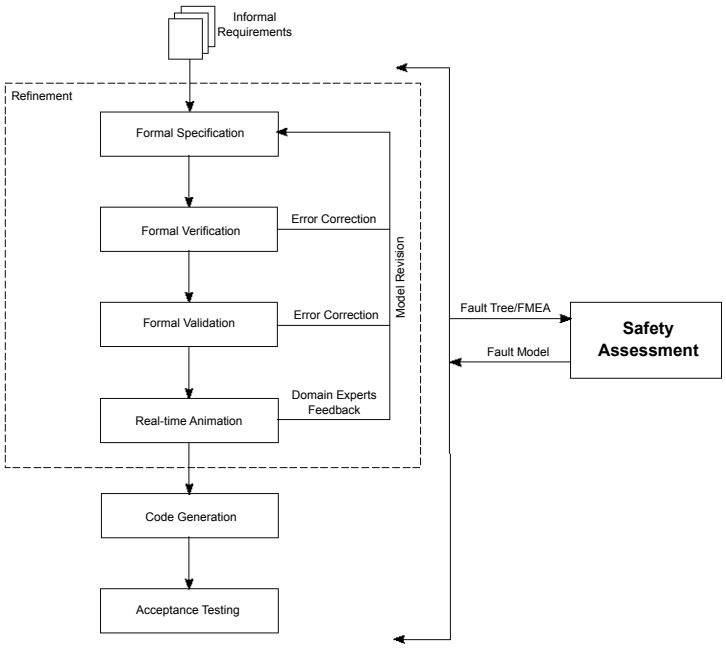


Figure 1: Méthodololologie formellement fondée pour le développement de systèmes critiques

0.5.6 L'atelier EB2ALL: génération automatique des codes

L'atelier EB2ALL 4 fournit des greffons permettant de traduire des mod§les Event-B en C, C++, C# et Java.

0.5.7 Modélisation du fonctionnement du cœur

Dans ce chapitre, nous développons une suite de raffinement sucessifs de modèles en vue de la modélisation du comportement du cøur du point de vue du flux électrique. L'importance de ce travail réside dans le fait de pouvoir combiner ce modèle et celui du pacemaker, afin d'obtenir une modélisation en boucle fermée.

0.5.8 Etudes de cas

Nous avons développé trois études de cas de taille significative. Une première étude de cas était celle du pacemaker proposée par la société Boston Scientific dans le cadre du Grand Challenge. Puis nous avons appliqué notre méthodologie sur un problème de modélisationdu système d'assitance au contrôle de la vitesse. Enfin, nous avons développé un modèle à partir de la description du protocole médical associé à l'analyse des ECG.

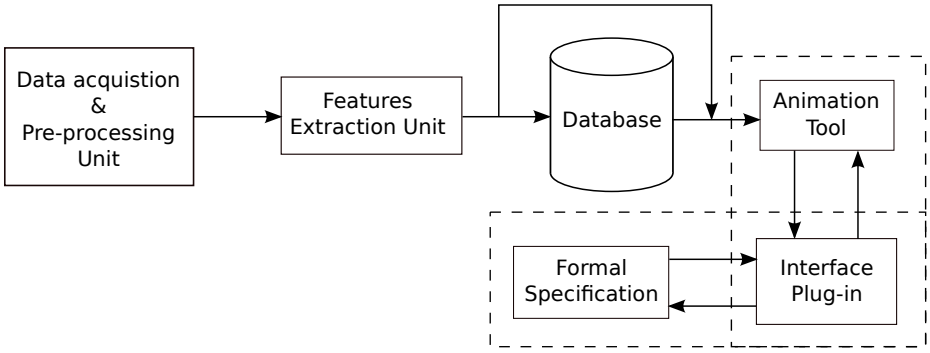


Figure 2: Architecture fonctionnelle pour animer un modèle formel sur des données réelles sans produire le code

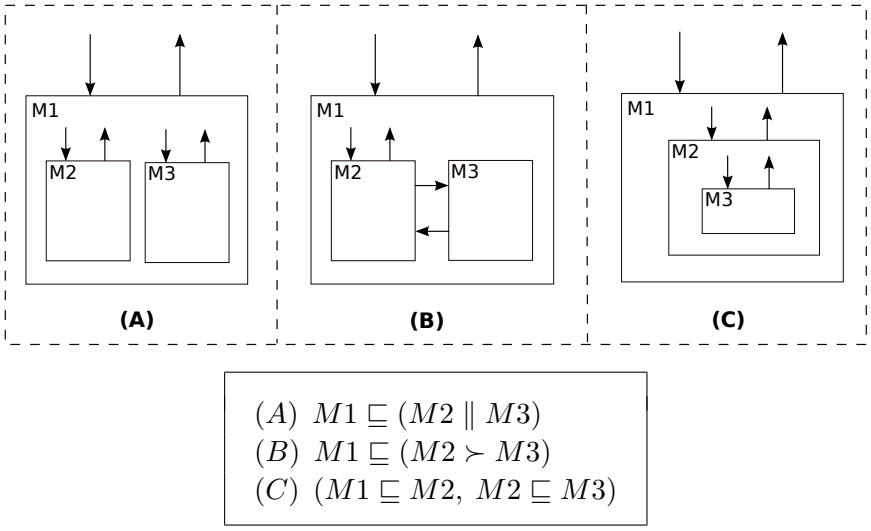


Figure 3: Chartes de raffinement

0.6 Conclusion et Perspectives

Les systèmes hautement critiques, telles que les systèmes médicaux, systèmes de l'avionique et systèmes de l'automobile, exigent une grande intégrité, la fiabilité des logiciels et du développement fondé sur la preuve pour l'obtention des certificats des organismes de certification [FDA ; IEEE-SA ; CC ; ISO], qui évaluent le système avant de l'utiliser . Dans ce contexte, l'adaptation de d'une méthode formelle est devenue un standard, pour répondre aux exigences élevées en matière de sécurité et de fiabilité vis-à-vis des organismes de certification [NITRD 2009]. Cependant, l'adaptation des méthodes formelles complique considérablement le processus de développement d'un système en raison de la complexité de la modélisation, ainsi que la complexité du système lui-même. Les techniques de modélisation basées sur le raffinement, réduisent l'effort de vérification de façon significative par la conception du système complet, en utilisant un processus de développement par étapes. Le système complet est vérifié à l'aide du démonstrateur, un vérificateur de modèle

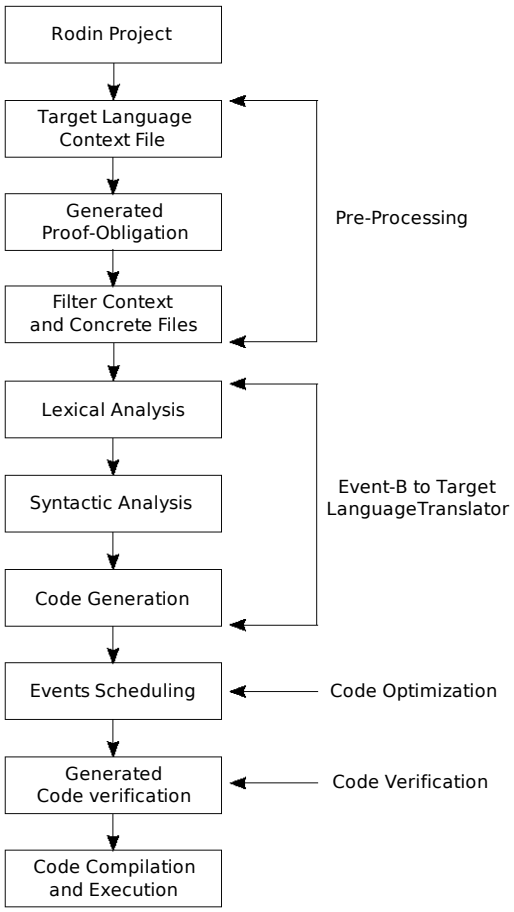


Figure 4: Architecture générale d'un outil de traduction

et d'outils d'animation. Par ailleurs, les systèmes critiques peuvent être analysés déjà à un stade précoce de leur développement; cela permet d'explorer des erreurs conceptuelles, des ambiguïtés, la correction des exigences et les problèmes de conception avant la mise en œuvre du système réel, et cette approche permet de corriger les erreurs plus facilement et de façon plus économique.

0.6.1 Contributions

Cette thèse présente un nouveau cadre de développement intégrant et un ensemble de techniques et d'outils, pour développer un système hautement critique fondée sur la vérification formelle et les approches de validation de l'analyse des besoins à la génération automatique de code source, en utilisant plusieurs étapes intermédiaires, qui peuvent être utiles pour la certification [FDA ; IEEE-SA ; ISO ; CC]. Nous introduisons de nouvelles techniques et des outils basés sur les méthodes formelles, qui contribuent au développement effectif, à la vérification et aux étapes essentielles de validation et de génération automatique de code à partir des modèles formellement prouvés du système. Les techniques et les outils proposés comprennent un cadre pour l'animateur [Méry 2010b] en

temps réel, le concept de charte de raffinement [Méry 2011e], un cadre de modélisation pour la certification [Méry 2010d], une modélisation formelle du cœur pour une modélisation à boucle fermée [Méry 2011f; Méry 2011i] et outils de génération automatique de code [Méry 2010a; EB2ALL 2011].

Dans le chapitre 1, nous avons donné un certain nombre d'objectifs et de besoins couverts dans ce travail de thèse. Ces résultats conduisent à une nouvelle méthodologie de développement, ainsi qu'à un ensemble de techniques et d'iutils associés en vue du développement de systèmes critiques. L'évaluation de la méthodologie de développement, tout comme les techniques et les outils, est donnée au travers de cas d'études bien connues liés au domaine médical et au domaine de l'automobile. Ce travail a établi une théorie unifiée pour le développement de systèmes critiques, poiur ce qui concerne notre premier objectif principal. Des techniques et des outils en lien avec un nouvelle méthodologie de cycle de développement valide la théorie unifiée proposée, ce qui est notre second objectif. Notre troisième objectif est lié à la modélisation de l'environnement en vue de développer des systèmes en boucle fermée. Notre modèle du cœur a été utilisé pour concevoir un modèle en boucle fermée du défibrilateur. L'approche de modélisation fondée sur le raffinement est une aide précieuse dans la conceptionde systèmes sans erreur et cela constitue notre quatrième objectif. Les trois objectifs suivants se concentrent sur la modélisation de différents composants, de sous-systèmes et finalement de l'intégration du système final. Dans ce contexte, nous avons proposé la charte de raffnement pour aider la modélisation à base de composants ainsi que pour l'intégration dans le système final. Un autre objectif se résume ainsi, la certification fondée sur l'évidence, au travers de technqiues d'animation temps réel. L'animateur temps réel peut alors être utilisé pour corriger des problèmes de niveau conceptuel, qui n'apparaissent pas avec les techniques traditionnelles. L'objectif final est lié à la certification de l'assurance de sûreté. Quand un système est développé selon notre approche, alors il peut être certifiable, parce que les standards de certification ont des exigences semblables pour valider les processus de système. Nous avons suivi une approche rigoureuse pour la conception de système plutôt qu'un décveloppement traditionnel de systèmes critiques. Dans un développement traditionnel, les méthodes formelles fournissent des assurances de sûreté et remplissent les exigences des standards. Notre nouvelle approche est complètement fondée sur des techniques formelles et développe, dans son intégralité et avec rigueur, un système en partant de l'analyse des exigences pour finalement produire du code, satisfaisant toustes les exigences des organismes de standardisation. Additionnellement, la méthodologie fournit une évaluation de la sûreté en vue de l'analyse du cycle complet de vie, ce qui est conforme aux exigences des organismes de certification. La complexité des systèmes critiques rend difficile leur compréhension et leur vérification. Plusieurs approches permettent de les vérifier, parmi lesquelles on citera le model checking, la preuve formelle ou la validation fondée sur la simulation. Il existe un grand nombre de problèmes liés aux systèmes critiques et plusieurs solutions pour chaque problème. Il est indispensable de raisonner rigoureusement sur le comportement des systèmes afin d'assurer que le comportement attendu est atteint.

Le langage de modélisation Event-B décrit un système de manière abstraite and permet d'ajouter progressivement des détails au cours de pas de raffinement en vue d'obtenir le modèle final du système. Les outils de la plate-forme Rodin [RODIN 2004; Abrial 2010]

fournisse une assistance à la preuve pour vérifier les conditions de vérification produites. Les conditions de vérification contribuent à la compréhension du problème et à la correction du système. Dans cette thèse, nous avons contribué à l'intégration du développement fondé sur la raffinement en utilisant Event-B, langage d'expression des spécifications formelles et de la vérification et de l'implantation du code pour des systèmes critiques. Une avancée significative est la nouvelle méthodologie du cycle de vie; il exploite les fondements mathématiques pour apporter un développement de système fondé sur une preuve complète rigoureuse en employant des techniques formelles à chaque étape allant de l'analsye des exigences à la production automatique de code. Cette méthodologie du cycle de vie est utilisée pour développer les systèmes critiques, en vue d'obtenir les standards de certificats, tels que IEC-62304 [IEC62304 2006] et les critères communs [CC ; Farn 2004; Mead 2003]. Cette méthodologie de développement combine l'approche du raffinement avec la vérification, le model checking, l'animation temps-réel et finalement produit le code source à l'aide des outils automatiques. Le processus de développement de système est évalué parallèlement par une approche d'évaluation [Leveson 1991] de la sûreté au regard des standards de certification. Appliquer ces nouvelles approches à des systèmes hautement critiques a des avantages: des erreurs qui pourraient ne pas être détectées sans des méthodes formelles. Le guide du NITRD [NITRD 2009] permet l'adoption des méthodes formelles dans un ensemble établi de processus pour le développement et la vérification d'un systèmes médical, en préconisant un raffinement évolutif plutôt qu'un changement radical de méthodologie.

Les travaux de thèse ont également fait progresser le développement de nouvelles techniques et outils pour soutenir la nouvelle méthodologie de cycle de vie, et sont t expliqués dans les phases ultérieures:

L'animateur temps-réel est utilisé pour valider le modèle formel avec des données réelles à un stade précoce de développement du système sans produire le code source [Méry 2010b], et pour combler le fossé entre les ingénieurs logiciel et les intervenants, pour construire le système de qualité et découvrir toutes les informations ambiguës des exigences. L'approche combinée de la vérification formelle et de l'animation temps réel permet le développement systématique d'une façon claire, concise, précise d'une spécification non ambiguë du système; il permet auxingénieurs d'animer la spécification formelle à un stade précoce de la de développement. Par ailleurs, il existe des applications scientifiques et juridiques, où l'animation du modèle formel peut être utilisé pour simuler (ou émuler) certains scénarios pour glaner plus d'informations ou mieux comprendre le système et pour améliorer le système final donné.

Une autre contribution significative à l'amélioration des techniques et des outils est la charte de raffinement, qui est utilisée pour présenter l'ensemble du système, en utilisant l'approche en couches dans les schémas graphiques, où les blocs fonctionnels sont divisés en plusieurs blocs plus simples à un niveau de raffinement nouveau, sans changer le comportement original du système. La charte de raffinement offre une vue claire de l'assistance dans l'intégration du système. Cette approche donne également une vue claire sur le système assemblant fondé sur les modes de fonctionnement et les différents types de fonctionnalités. C'est une question importante non seulement pour être en mesure de dériver au niveau du système de performance et de garantir l'exactitude, mais aussi pour

être capable d'assembler des composants d'une manière rentable. La complexité de la conception est réduite par les systèmes utilisant les modes et la structuration en détaillant cette conception à l'aide de raffinement.

La génération automatique de code à partir d'un modèle prouvé formel dans le langage cible de programmation est une étape essentielle pour la mise en œuvre du système et est une contribution tout aussi importante. Nous avons développé les principaux principes, règles et la mise en œuvre des solutions pour l'outil de traduction, et aussi des techniques de vérification de code pour générer du code (C, C++, Java et C #) relatif à des spécifications [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b] formelles Event-B. La syntaxe adoptée est restrictive, mais avec de nombreuses caractéristiques essentielles, pour les applications plus numériques, des supports puissants de méthodes d'analyse et de génération de code source de manière rapide et sûre dans le langages de programmation cible. Les avantages du développement et de l'amélioration de l'outil de traduction [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b] proviennent principalement de leur soutien actif à la traduction automatisée entre les deux composantes d'un modèle formel et le langage de programmation cible. Les adaptations des règles de traduction nécessitent des expériences plus complètes, surtout avec de grands modèles formels pour vérifier l'impact sur les temps d'exécution de certaines plates-formes spécifiques. Les gains dépendent ensuite des garanties fournies en utilisant une méthode formelle et du niveau de certification qui peut être obtenu par cette voie. A notre connaissance, seules quelques méthodes formelles supportent la génération de code, ce qui est aussi efficace en temps et en espace que le code écrit à la main.

Le développement d'un environnement pour la modélisation en boucle fermée est une contribution significative de ce travail de hèse. Nous avons ont présenté une méthodologie pour la modélisation d'un modèle mathématique du c ur basée sur une théorie logico-mathématique. L'objectif le plus important est que ce modèle formel permet d'obtenir une certification pour les appareils médicaux appareils concernant la cardiologie comme le stimulateur cardiaque. Il peut également être utilisé comme un outil de diagnostic pour identifier une situation critique du patient, en utilisant le modèle d'environnement du patient. Le modèle de cœur est basé sur l'analyse de l'électrocardiogramme, qui modélise le système circulatoire du cœur au niveau cellulaire. Cela a été l'un des problèmes lesplus difficiles à valider et à vérifier le comportement correct du modèle du système développé (un stimulateur cardiaque ou DAI) sous environnement biologique (à savoir le c ur). Cette approche de formalisation et le raisonnement à propos propagation de l'influx dans le système grâce au réseau de conduction du cœur. Le modèle du cœur suggère qu'une telle approche peut produire un modèle viable qui peut être soumis à une validation utilisée contre les logiciels de dispositifs médicaux à un stade précoce dans le processus de développement (à savoir stimulateur cardiaque). Le modèle de c ur est validé avec le aide de physiologistes et de cardiologues.

L'évaluation de la méthodeologie de développement du cycle de vie et un ensemble d'outils et de techniques associées sont réalisés à travers le développement d'études de cas à l'échelle industrielle, qui couvrent deux domaines différents liées aux soins médicaux, à l'instrumentation et à l'automobile. L'idée est de choisir deux différents types d'études de cas pour montrer la généralité de l'approche proposée, et donc de valider son applicabilité à

tous les autres domaines. Ces deux études de cas sont d'une part le stimulateur cardiaque et d'autre part un régulateur de vitesse [Méry 2009; Méry 2011g; Méry 2010b]. Nous avons appliqué notre méthodologie de développement et les techniques et outils associés pour la mise en uvre de système. Les approches combinées de vérification formelle, de validation, les chartes de raffinement, l'animation temps-réel et la génération automatique de code couvrent les éléments énumétés comme l'assurance certifiables et la sûreté, le développement de système sans erreur et l'intégration du systèmes. Les chartes de raffinement couvre tout spécialement les cadres de conception par composant et les décomposition, l'intégration des infrastructures critiques critiques. Notre étude de cas sur le stimulateur cardiaques a montré que les spécifications des exigences pouvaient être utilisées directement dans un environnement temps réel sans modification à l'évaluation des résultats des tests d'analyse. Nous pouvons constater à partir de nos études de cas que toutes ces éléments aident à concevoir des systèmes sans erreur et différentes phases du système ont été identifiées par des raffinements du développement formel, ainsi que des chartes de raffinement. Nous avons présenté de manière évidente qu'une telle analyse est fructueuse pour les deux groupes de personnes non spécialistes en méthode formelle. La deuxième observation à partir de notre expériences, est que le développement de plusieurs modèles nous a permis non seulement de trouver des erreurs dans les documents d'exigences, mais aussi nous a donné une occasion de mieux comprendre les exigences complexes telles que l'algorithme de contrôle d'un système critique. Par ailleurs, nous croyons que l'effort nécessaire est proportionnel aux avantages que nous tirons de développement des modèles multiples.

Afin d'évaluer l'utilité globale de notre approche, une sélection des résultats des étapes de formalisation et de vérification ont été présentés à un groupe de développeurs de stimulateur d'une compagnie franco-italienne de développement de tels systèmes. Les développeurs ont répondu favorablement à notre résultat du développement incrémental du stimulateur, ainsi que l'intégration du matériel et des logiciels. Ils ont vraiment acquiescé sur les chartes de raffinement, qui montrent les modes de fonctionnement et leur relation avec les transitions de mode. De même, pour une autre étude de cas liés au système de contrôle automatique de la vitesse (ACC), les résultats des étapes de formalisation et de vérification ont été élaborés sous le contrôle d'un groupe de chercheurs (General Motors, Bangalore, Inde). Tout au long de notre étude de cas, nous avons montré la spécification formelle et la vérification du système ACC. Par conséquent, la validation doit être effectuée par les deux types de personnes: d'une part un expert en modélisation et des experts du domaine. En nous fondant sur l'expérience décrite ci-dessus et sur nos conclusions, nous sommes convaincus de son utilité pour certains domaines, et par conséquent, nous envisageons d'utiliser cette méthodologie pour concevoir des systèmes hautement critiques. Le cadre proposé, les techniques et les outils offre un cadre de développement formel allant de la vérification formelle à la génération de code, conduisant à donner une réponse au défi de se conformer à QSR de la FDA, ISO/CEI et standards IEEE [Keatley 1999; IEEE-SA ; FDA ; ISO ; CC] et facilitat l'obtention de la certification pour les systèmes hautement complexes critiques.

Cette thèse contribue aussi domaine de la représentation formelle des protocoles médicaux. Le modèle formel du protocole médical est vérifié, et ce modèle vérifié est non seulement faisable, mais également utile pour améliorer le protocole médical existant lui-même. Nous avons entièrement formalisé un tel protocole du monde réel protocole médical (ECG

interprétation) dans une démarche incrémentale de formalisation en nous appuyant sur le raffinement et nous avons utilisé les outils de preuve pour analyser systématiquement, si la formalisation est conforme à certains protocoles pertinents sur le plan des propriétés médicales [Méry 2011h; Méry 2011j]. La vérification formelle du processus a découvert un certain nombre d'anomalies. Nous avons également découvert une structure hiérarchique pour l'interprétation d'ECG de manière efficace;: cela s permet de découvrir un ensemble de conditions qui peuvent être très utiles pour diagnostiquer une maladie particulière à un stade précoce du diagnostic, sans diagnostics multiples. Notre structure arborescente permet des solutions plus concrètes pour le protocole d'interprétation d'ECG et aide à améliorer le protocole d'interprétation ECG original. Les principaux objectifs de cette approche sont de tester l'exactitude et la cohérence des protocole médical. Cette approche est menée non seulement à des fins de diagnostic, mais elle peut être applicable pour couvrir un large groupe d'autres catégories (traitement, gestion, prévention, conseil, évaluation etc)[1] liés aux protocoles médicaux.

0.6.2 Perspectives

Cette thèse propose des travaux liés au développement d'une méthodologie de cycle de développement, avec des techniques et des outils associés, pour le développement de systèmes critiques et présente une évaluation à travers le traitement complet du stimulateur cardiaque et le système de contrôle de la vitess. Dans cette section, nous décrivons des s extensions possibles de notre travail présenté dans cette thèse.

Dans le chapitre 4, nous avons présenté la méthodologie du cycle de vie de développement de systèmes critiques, en utilisant des méthodes formelles. Dans cette thèse, nous avons présenté deux études de cas significatives liées au domaines médical et à celui de l'automobile, pour montrer l'efficacité de la l'approche proposée, sans appliquer les techniques d'évaluation de la sécurité. L'approche d'évaluation de la sécurité est également une partie importante de cette méthodologie de développement proposé. C'est notre travail à venir pour appliquer les approches d'évaluation de la sécurité à chaque niveau de la méthodologie du cycle de vie de développement en vue de concevoir un système critique.

Dans le chapitre 5, nous avons présenté une architecture temps réel animateur et de développer un prototype pour tester l'efficacité de l'étude de cas stimulateur cardiaque. L'architecture proposée n'est pas encore complet en raison de certaines limitations, les dispositifs d'acquisition, les algorithmes d'extraction des caractéristiques et ainsi de suite. À l'avenir cela devrait utiliser la même architecture dans le domaine multi, l'architecture a proposé d'utiliser les données en temps réel mis pour valider toute spécification du modèle formel dans le stade précoce de développement. Manuel d'application de cette architecture à appliquer en temps réel des données défini dans le modèle formel est fastidieuse, lourde et peut être source d'erreurs si elle n'est pas appliquée avec soin. Par conséquent, afin de développer une interface de programmation d'application (API) qui peut servir d'interface des composants automatiquement différents de l'architecture.

Dans le chapitre 6, nous avons donné l'idée de tableau de raffinement, ce qui est utile pour concevoir le système modal selon les modes de fonctionnement grâce à l'approche

[1] http://www.guideline.gov

de raffinement. En conséquence, nous avons utilisé développement manuel de la carte de raffinement dans notre stimulateur cardiaque et régulateur de vitesse adaptatif (ACC) des études de cas. Comme un travail futur, nous prévoyons de développer un environnement de développement intégré (IDE) pour la conception d'un système critique avec la charte de raffinement et la formalisation automatique du système critique.

Au chapitre 7, nous avons présenté un outil de EB2ALL [EB2ALL 2011], qui génère des codes source en C, C++, Java et des langages de programmation C# automatiquement. Cet outil est mis en œuvre avec succès, sauf le code de vérification étape généré. Nous travaillons sur notre outil de traduction et maintenant nous mettons en œuvre la dernière étape de notre chaîne d'outils, la vérification du code généré. C'est la tâche importante et difficile dans le développement de l'outil. La raison en est que la préservation au niveau du code des propriétés prouvé au niveau architectural est garanti seulement si - la plate-forme sous-jacente est correcte et - bien-fondé de finale du système lors du remplissage dans les talons pour des actions internes dans le code généré automatiquement. Il est de notre projet en cours, nous sommes donc poursuivre l'extension de cet outil pour soutenir toutes les autres restant B événementiel symboles formels, qui garantit la liberté pour un développeur d'un système formalisé critique et de génération automatique de code source d'un modèle développé dans n'importe quelle langue cible. À l'avenir, nous avons prévu d'étendre cet outil de traduction du langage de programmation d'autres types cibles tels que *langage de programmation API*, de sorte que cet outil de traduction [EB2ALL 2011; Méry 2010a] peut être utilisée par tous les secteurs industriels, alors que formelle vérification et la validation des techniques primaires de développer un système.

Au chapitre 8, nous présentons un modèle d'environnement pour le système cardiaque, que nous utilisons pour intégrer ma spécification formelle du pacemaker [Méry 2011g] et la spécification formelle du cœur pour modéliser en boucle fermée et pour vérifier le comportement souhaité du stimulateur cardiaque en vue de sa certification. Comme travaux futurs, nous avons besoin de tester un tel type de modèle de l'environnement pour tester d'autres systèmes médicaux liés aux systèmes cardiaque.

Au chapitre 9, l'objectif le plus important est que le modèle formel développé permet d'obtenir une certification pour le stimulateur cardiaque. Notre objectif est de créer le simulateur du stimulateur cardiaque qui repose sur ces modèles formels et qui peuvent être utilisés par le médecin pour analyser en temps réel les signaux du cœur et pour prédire les modes de fonctionnement. Il peut être également utilisé comme un outil de diagnostic pour diagnostiquer le patient et l'aider à prendre la meilleure la décision d'implantation d'un stimulateur cardiaque [Writing Committee 2008].

Au chapitre 10, nous avons présenté une nouvelle approche de modélisation pour la modélisation de systèmes hybrides. Nous avons montré un développement réussi formel de système ACC en utilisant des méthodes formelles. Nous avons terminé l'ensemble du processus de modélisation et de formalisation du système ACC et les propriétés, et nous nous sommes acquittés de toutes les obligations de preuve générées. À l'avenir on devrait utiliser la même approche du développement formel de système de contrôle très complexes hybrides dans plusieurs domaines où les contrôleurs hybrides sont utilisées. Nous avons trouvé des futurs possibles travaux connexes à notre approche sont:

- Utiliser différents types de régulation PID (P, PI, PD, PID)
- Vérification de propriétés complexes comme la composition de chaque contrôleur
- Composition parallèle, série et en parallèle la composition et la composition combinée série de contrôleurs
- Identification et utilisation des modèles dans la conception du contrôle et la transformation automatique de modèles Event B en modèles Simulink.

Dans le chapitre 11, nous nous concentrons sur quelques défis futurs liés à l'application des techniques formelles pour améliorer le protocole médical. Nous avons montré que la mise en œuvre réussie d'amélioration du protocole médical, en utilisant des méthodes formelles, où nous avons terminé la totalité du processus de modélisation et de formalisation du protocole d'interprétation d'ECG et les propriétés, et nous nous sommes acquittés de toutes les obligations de preuve générées. Une structure hiérarchique pour le diagnostic ECG est découvert, ce qui contribue à améliorer le protocole de l'ECG. Comme un travail futur, il est possible d'utiliser la même technique pour enquêter sur d'autres types de protocoles médicaux utilisant notre approche de modélisation basée sur le raffinement.

CHAPTER 1

Introduction

"The man of science has learned to believe in justification, not by faith, but by verification."

(Thomas Huxley)

The primary goal of the thesis is to advance the use of formal techniques for the development computing systems with high integrity. Specifically, the thesis makes an analysis of critical system software that formal methods are not well integrated into established critical systems development processes. The thesis work suggests a formalism for a new development life-cycle, and proposes a set of associated techniques and tools to develop highly critical systems using formal techniques from requirements analysis to automatic source code generation using several intermediate layers with rigorous safety assessment approach. The approach has been verified using the Event-B formalism, and tools, with the addition of the following quite a few new aspects of real-time animator, refinement chart, a set of automatic code generation tools, and critical system development methodology, have been added. The efficacy of formalism has been evaluated by means of case studies in the area of medical and automotive domains. In this chapter, we present the motivation of this work and main concepts of our proposed approach for developing a new methodology for system development, and associated techniques and tools.

1.1 Motivation

Nowadays, software systems have penetrated into our daily life in many ways. Information technology is one major area, which provides powerful, and adaptable opportunities for innovation. However, sometimes computer-based developed systems are producing disappointed results and fail to produce the desired results according to work requirements and stakeholder needs. They are unreliable, and eventually dangerous. As a cause of system failure, poor developments practices [Leveson 1993; Zhang 2010; Gibbs 1994; Price 1995; Yeo 2002] are one of the most significant. This is due to the complex nature of modern software and lack of understanding. Software development provides a framework for simplifying a complex system to get a better understanding and to develop the higher-fidelity system at a lower cost. Highly embedded critical systems, such as automotive, medical, and avionic, are susceptible to errors, which are not sustainable in case of failure. Any failure in these systems may to two types of consequences: *direct consequences* and *indirect*

consequences. Direct consequences lead to finance, property losses, and personal injuries, while indirect consequences lead to income lost, medical expenses, time to retain another person, and decrease employee moral, etcetera. Additionally, and most significantly potential loss is customer trust for a product failure. In this context, a high degree of safety and security is required to make amenable to the critical systems. A system is considered to accomplish the current task safely in case of a system failure. A formal rigorous reasoning about algorithms and mechanisms beneath such a system is required to precisely understand the behavior of systems at the design level. However, to develop a reliable system is a significantly complicated task, which affects the reliability of a system.

Formal methods-based development [Woodcock 2007; Gaudel 1996; Jetley 2006] is a very standard and popular approach to deal with the increasing complexity of a system with assurance of correctness in the modern software engineering. Formal methods-based techniques increasingly control safety-critical functionality in the development of the highly critical systems. These techniques are also considered as a way to meet the requirements of the standardize certificates [ISO ; IEEE-SA ; CC ; FDA] to evaluate a critical system before use in practice. Furthermore, critical systems can be effectively analysed at early stages of the development, which allows to explore conceptual errors, ambiguities, requirements correctness, and design flaws before implementation of the actual system. This approach helps to correct errors more easily and with less cost. We formulate following objectives related to a critical system development, and propose formal methods-based development methodology and a new set of associated techniques and tools to meet the following objectives:

- Establishing a unified theory for the critical system development.
- Building a comprehensive and integrated suite of tools for the critical systems that support verification activities, including formal specification, model validation, real-time animation and automatic code generation.
- Environment modeling for closed-loop system development for verification purposes.
- Refinement-based formal development to achieve less error-prone models, easier specification for the critical systems and reuse of such specification for further designs.
- Decomposition of the complex system into different independent subsystems and re-composition after validation.
- Model-based development and component-based design frameworks.
- System integration of critical infrastructure. Possibility of annotating models for different purposes, (e.g., directing the synthesis or hooking to verification tools).
- Evidence-based certification through animation.
- Requirements and metrics for certifiable assurance and safety.

The enumerated objectives are covered in this thesis through developing a new development life-cycle methodology and a set of associated techniques and tools for developing the critical systems. The development life-cycle methodology is a development process for the systems to capture the essential features precisely in an intuitive manner. New sets of techniques and tools, and a development methodology are developed for handling stakeholder's requirements, refinement-based system specification, verification, model animation using real-time data set through a real-time animator, and finally automatic source code generation from the verified formal specification. All these approaches may also help to get certificates from the international standards [ISO ; IEEE-SA ; CC ; FDA].

1.2 Approach

In this thesis, we present a development life-cycle methodology and a set of associated techniques and tools to develop highly critical systems based on formal techniques from requirements analysis to code implementation. This thesis has mainly two parts: techniques and tools and case studies. The techniques and tools section consists of a development life-cycle methodology, a framework for real-time animator [Méry 2010b], refinement chart [Méry 2011e], a set of automatic code generation tools [Méry 2010a; Méry 2011b; EB2ALL 2011] and formal logic based heart model for close loop modeling [Méry 2011f; Méry 2011i]. The development methodology and associated tools are used for developing a critical system from requirements analysis to code implementation, where verification and validation tasks are used as intermediate layers for providing a correct formal model with desired system behavior at a concrete level. Introducing new tools help to verify desired properties, which are hidden at the early stage of the system development. For example, a real-time animator provides a new way to discover hidden requirements according to the stakeholders. It is an efficient technique to use the real-time data set, in a formal model without generating the source code in any target programming language[Méry 2010b], which also provides a way for domain experts (i.e. medical experts) to participate in the system development process (medical device development). In the following paragraph, the basic description about development methodology and all associated techniques and tools are provided as follows:

We propose a new methodology, which is an extension of the waterfall model [Acuña 2005; Schumann 2001; Wichmann 1992; Bell 1993] and utilises related to formal methods set rigorous approaches to produce a reliable critical system. This methodology combines the refinement approach with a verification tool, model checker tool, real-time animator and finally generates the source code using automatic code generation tools. The system development process is concurrently assessed by safety assessment approach [Leveson 1991] to comply with certification standards. This life-cycle methodology consists of seven main phases: first, informal requirements, resulting in a structured version of the requirements, where each fragment is classified according to a fixed taxonomy. In the second phase, informal requirements are represented in the formal modeling language, with a precise semantics, and enriched with invariants and temporal constraints. The third phase consists of refinement-based formal verification to test the internal consistency and correctness of the

specifications. The fourth phase is the process of determining the degree to which a formal model is an accurate representation of the real world from the perspective of the intended uses of the model using a model-checker. The fifth phase is used to animate the formal model with real-time data set instead of *toy-data*, and offers a simple way for specifiers to build a domain-specific visualization that can be used by domain experts to check whether a formal specification corresponds to their expectations. The six phase generates the source code from the verified system specifications and final phase is used for acceptance testing of the developed system. This kind of approach is very useful to verify complex properties of a system and to discover the potential problems like deadlock and liveness at an early stage of the system development.

According to the development life cycle of a critical system, first of all we emphasize the requirements traceability using a real-time animator [Méry 2010b]. Formal modeling of requirements is a challenging task, which is used to reasoning in earlier phases of the system development to make sure completeness, consistency, and automated verification of the requirements. The real-time animation of a formal model has been recognized to be a promising approach to support the process of validation of requirement's specification. The principle is to simulate the desired behaviors of a given system using formal models in the real-time environment and to visualize the simulation in some form which appeals to stakeholders. The real-time environment assists in the construction, clarification, validation and visualization of a formal specification. Such an approach is also useful for evidence-based certification.

Refinement techniques [Abrial 2010; Abrial 1996a; Back 1979] serve a key role for modeling a complex system in an incremental way. A refinement chart is a graphical representation of a complex system using a layering approach, where functional blocks are divided into multiple simpler blocks in a new refinement level, without changing the original behavior of the system. A final goal in using this refinement chart is to obtain a specification that is detailed enough to be effectively implemented, but also to correctly describe the system behaviors. The purpose of the refinement chart is to provide an easily manageable representation for different refinements of the systems. The refinement chart offers a clear view of assistance in "system" integration. This approach also gives a clear view about the system assembly based on operating modes and different kinds of features. This is an important issue not only for being able to derive system-level performance and correctness guarantees, but also for being able to assemble components in a cost-effective manner.

Another important step in the software-development life cycle is the code implementation. In this context, we have developed an automatic code generation tool [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b] for generating an efficient target programming language code (C, C++, Java and C#) from Event-B formal specifications related to the analysis of complex problems. This tool is a collection of plug-ins, which are used for translating Event-B formal specifications into different kinds of programming languages. The translation tool is rigorously developed with safety properties preservation. We present an architecture of the translation process, to generate a target language code from Event-B models using Event-B grammar through syntax-directed translation, code scheduling architecture, and verification of an automatic generated code.

A closed-loop model of a system is considered as a *de facto* standard for critical systems in the medical, avionic, and automotive domains for validating the system model at early stages of the system development, which is an open problem in the area of modeling. The cardiac pacemaker and implantable cardioverter-defibrillators (ICDs) are key critical medical devices, which require closed-loop modeling (integration of system and environment modeling) for verification purpose to obtain a certificate from the certification bodies. In this context, we propose a methodology to model a biological system related to the heart system, which provides a biological environment for building the close loop system for the cardiac pacemaker. The heart model is mainly based on electrocardiography analysis, which models the heart system at the cellular level. The main objective of this methodology is to model the heart system and integrate with a medical device model like the cardiac pacemaker to specify a closed-loop model. Industries have been striving for such a kind of approach for a long time in order to validate a system model under a virtual biological environment.

Assessment of the proposed framework, and techniques and tools are scrutinised through the development of two different case studies related to the medical and automotive domains. The cardiac pacemaker challenge is related to the medical domain, and adaptive cruise control (ACC) is related to the automotive domain. These two case studies assess main any one key features of the proposed frameworks and developed tools [Méry 2011g; Méry 2009; Méry 2010c] from requirements analysis to code implementation.

Formal techniques are useful not only for critical-systems, but are also for applying to verify required safety properties in other domains, for example, in the clinical domain to verify the correctness of protocols and guidelines [Méry 2011h; Méry 2011j]. Clinical guidelines systematically assist practitioners with providing appropriate health care for specific clinical circumstances. Today, a significant number of guidelines and protocols are lacking in quality. Indeed, ambiguity, and incompleteness are common anomalies in medical practices. Our main objective is to find anomalies and to improve the quality of medical protocols using well-known mathematical formal techniques, such as Event-B. In this study, we use the Event-B modeling language to capture guidelines for their validation for improving the protocols. An appropriateness of the formalism is given through a case study, relative to a real-life reference protocol (ECG Interpretation) which covers a wide variety of protocol characteristics related to several different heart diseases.

1.3 Contribution and Outline

1.3.1 Contribution

The following contributions have been made in this thesis: We propose a development methodology, and a set of associated techniques and tools to develop a highly critical system based on formal verification and validation approaches from requirements analysis to automatic source code generation using several intermediate steps [Méry 2010b; Méry 2011e; Méry 2011g; Méry 2011f; EB2ALL 2011; Méry 2011a; Méry 2011b]. Our approach of specification and verification is based on the techniques of abstraction and refinement.

This (formal) technique, supported in Event-B, consists of describing rigorously the problem in the abstract model and introducing the solution or design details in refinement steps. Here, we present an incremental proof-based development to model and verify such interdisciplinary requirements in Event-B [Cansell 2007; Abrial 2010]. Through the refinement, we verify that the detailed design of a system in the refinement conforms to the initial abstract specifications. Our approach is based on the Event-B modeling language which is supported by the Rodin platform integrating tools for proving models and refinements of models; moreover, we use the ProB tool [ProB ; Leuschel 2003] for analyzing the models and for validating these models. The Event-B tool generates proof obligations relating abstract and concrete variables for refinement checking. We have used the Rodin proof tools [RODIN 2004] for the generation of proof obligations and discharge them using the automatic and interactive prover. In order to discharge these proof obligations, we need to add a series of new gluing invariants to the model. These gluing invariants demonstrate the relationship between abstract and concrete variables. The discovery of these new gluing invariants provides a clear insight to the system and supports precise reasoning about why a specific solution proposed in the refinement is a correct solution to an abstract problem. Formal methods are usually used in analysing assumptions, relationships, and requirements of a system.

The proposed development life-cycle methodology and a set of associated techniques and tools [Méry 2010b; Méry 2010d; Méry 2011e; Méry 2011f; Méry 2011b] have been evaluated in two different kinds of critical systems: cardiac pacemaker and adaptive cruise control (ACC). The cardiac pacemaker as a "Grand Challenge", which has proposed by the Verified Software Initiative [Hoare 2009; Woodcock 2007]. ACC is a real application system, which has offered by the India Science Lab, General Motor. Both systems are complex and challenging in different domains related to the medical and automotive. These two case studies assess key features of the proposed frameworks and developed tools [Méry 2011g]. The case studies use a development framework and set of techniques and tools from requirements analysis to code implementation. The framework provides a platform for the integration of model-based design with a heterogeneous set of formal verification tools. The refinement based modeling techniques reduces the verification effort significantly by designing the whole system using stepwise refinement and each refinement is verified using theorem prover, model checker and animation tools. We have also used an environment model of the heart system for modeling the close loop system of the cardiac pacemaker. A real-time animator is used to verify formal models and the actual system behavior according to the domain experts. Moreover, a code generation tool [EB2ALL 2011; Méry 2011b] helps to generate the source code from the proved formal model of the system for implementation. The formal verification and model validation offers to meet the challenge of complying with FDA's QSR, ISO, IEEE, CC [ISO ; CC ; IEEE-SA ; FDA] quality system directives. According to the FDA QSR, validation is the "confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use can be consistently fulfilled". Verification is "confirmation by examination and provision of objective evidence that specified requirements have been fulfilled" [NITRD 2009; Méry 2010d].

Moreover, we have presented the formal representation of the medical protocol. The

formal model of a medical protocol [Méry 2011h] is verified, and this verified model is not only feasible but also useful for improving the existing medical protocol. We have fully formalised a real-world medical protocol (ECG interpretation) to analyse whether the formalisation complies with certain medically relevant protocol properties. The formal verification process has discovered a number of anomalies. We have also discovered a hierarchical structure that helps to discover a set of conditions that can be very helpful to diagnose particular disease at the early stage of the diagnosis without using multiple diagnosis. The main objective of this approach is to test correctness and consistency of the medical protocol.

1.3.2 Thesis outline

The thesis is structured in 12 Chapters. Chapter 2 presents state of the art. Chapter 3 describes modeling techniques using the Event-B modeling language, and then this thesis is divided into two parts.

Part I, which gives development methodology and associated list of formal methods based new techniques and tools, is arranged as follows. In Chapter 4, we propose a development life-cycle methodology for developing the highly critical software systems using formal methods from requirements analysis to code implementation using rigorous safety assessments. In Chapter 5, we propose a novel architecture to validate the formal model with real-time data set in the early stage of development without generating the source code in any target language. [Méry 2010b]. In Chapter 6, the refinement chart is proposed to handle the complexity and for designing the critical systems. In Chapter 7, we present a tool that automatically generates efficient target programming language code (C, C++, Java and C#) from Event-B formal specification related to the analysis of complex problems. In this chapter, the basic functionality as well as the design-flow is described, stressing the advantages when designing this automatic code generation tool; EB2ALL [EB2ALL 2011; Méry 2011d; Méry 2010a; Méry 2011c; Méry 2011b]. In Chapter 8, we present a methodology to model a biological system, like the heart. The heart model is mainly based on electrocardiography analysis, which models the heart system at the cellular level. The main objective of this methodology is to model the heart system and integrate it with the medical device model like the cardiac pacemaker to specify a closed-loop model.

Part II, which reports on results of the empirical evaluation of all proposed approaches and set of tools and techniques on two different kinds of case studies. In Chapter 9, we present the complete formal development of a cardiac pacemaker using proposed techniques and tools from requirements analysis to automatic code generation. Similarly, In Chapter 10, we present formal development of the adaptive cruise control (ACC) from requirements analysis to code generation. Moreover, in this case study, we generate a *simulink* model from the proved formal model. In Chapter 11, we present a new application of formal methods to evaluate real-life medical protocols for quality improvement. An assessment of the proposed approach is given through a case study, relative to a real-life reference protocol (ECG Interpretation) which covers a wide variety of protocol characteristics related to several heart diseases. We formalise the given reference protocol, verify

a set of interesting properties of the protocol and finally determine anomalies. Chapter 12 concludes this thesis with a summary of the contributions made and an outlook to future directions of research.

1.4 Publications

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

International Journals

- D. Méry and **N. K. Singh** "*Formal Specification of Medical Systems by Proof-Based Refinement*", ACM Transaction on Embedded Computing Systems, May, 2011, (*in Press*). (Part-I)
- D. Méry and **N. K. Singh** "*Functional behavior of a cardiac pacing system*", International Journal of Discrete Event Control System, 2010. (Part-II)
- D. Méry and **N. K. Singh** "*A Generic Framework: from Modeling to Code*", Journal of Innovations in Systems and Software Engineering (ISSE) NASA, Springer London Pages 1-9, 2011. (Part-I)

International Conferences

- D. Méry and **N. K. Singh** "*Automatic Code Generation from Event-B Models*", Proceedings of the 2011 Symposium on Information and Communication Technology, SoICT 2011, Hanoi, Vietnam, ACM, ACM International Conference Proceeding Series, October 13-14, 2011. (Part-I)
- D. Méry and **N. K. Singh** "*Analysis of DSR protocol in Event-B*", 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2011), Springer LNCS,10-12 October 2011.
- D. Méry and **N. K. Singh** "*Formal Development and Automatic Code Generation : Cardiac Pacemaker*", International Conference on Computers and Advanced Technology in Education (ICCATE 2011), Communications in Computer and Information Science (CCIS), Springer, 3-4 November 2011. (Part-II)
- D. Méry and **N. K. Singh** "*EB2J : Code Generation from Event-B to Java*", 14th Brazilian Symposium on Formal Methods (SBMF 2011), 26-30 September 2011, (*Short Paper*)) (Part-I).
- D. Méry and **N. K. Singh** "*Formalisation of the Heart based on Conduction of Electrical Impulses and Cellular-Automata*", International Symposium on Foundations of Health Information Engineering and Systems (FHIES 2011), Springer LNCS, 29-30 August 2011. (Part-I)
- D. Méry and **N. K. Singh** "*Medical Protocol Diagnosis using Formal Methods*", International Symposium on Foundations of Health Information Engineering and Systems (FHIES 2011),Springer LNCS, 29-30 August 2011. (Part-II)

- D. Méry and **N. K. Singh** "*Trustable Formal Specification for Software Certification*", in Proceeding ISoLA (2), Springer LNCS,Vol-6416, 312-326, 2010. (Part-I)
- D. Méry and **N. K. Singh** "*Real-time animation for formal specification*", in Proceeding Complex Systems Design & Management, Springer-Verlag, Paris, 27-29 October, 2010. (Part-I)
- D. Méry and **N. K. Singh** "*EB2C : A Tool for Event-B to C Conversion Support*", Poster and Tool Demo submission, and published in a CNR Technical Report in SEFM 2010. (Part-I)

Technical Reports

- D. Méry and **N. K. Singh** "*Technical Report on Formalisation of the Heart using Analysis of Conduction Time and Velocity of the Electrocardiography and Cellular-Automata* ", Technical Report (http://hal.inria.fr/inria-00600339/en/), 2011. (Part-I)
- D. Méry and **N. K. Singh** "*Technical Report on Interpretation of the Electrocardiogram (ECG) Signal using Formal Methods*", Technical Report (http://hal.inria.fr/inria-00584177/en/), 2011. (Part-II)
- D. Méry and **N. K. Singh** "*Pacemaker's Functional Behavior in Event-B*", Technical Report (http://hal.inria.fr/inria-00419973/en/), 2009. (Part-II)
- D. Méry and **N. K. Singh** "*Formal Development of Two-Electrode Cardiac Pacing System*" Technical Report, (https://hal.archives-ouvertes.fr/inria-00465061/en/), 2010). (Part-II)

CHAPTER 2

State of the Art

"Reason, observation, and experience;
the holy trinity of science."
(Robert Green Ingersoll)

2.1 Introduction

Critical systems are tremendously grown in functionality in both software and hardware, and due to increasingly the complexity of the critical systems it is very hard to predict an absence of failure. Moreover, some of these failures may cause catastrophic financial loss, time or even human life. One of the main objectives of software engineering is to provide a framework to develop a critical system that operates reliably despite this complexity. It has been shown in [Woodcock 2009; Rushby 1995] that the promising results are achievable only through the use of formal methods in the development process. More than a decade, several formal methods based techniques, and tools are used by industries as well as in academic projects [Woodcock 2007; Jetley 2006]. The backbone of formal methods is considered to be mathematics, which often supports related techniques and tools based on logico-mathematical theory for specifying and verifying the complex systems. The techniques and tools based on formal methods have provided a certain level of reliability under some constraints. Formal verification is considered as a benchmark technique, particularly in the area of safety critical systems, where important safety properties are required to prove rigorously before the system implementation. However, the use of formal methods helps to speculate the hidden peculiarity of the system like inconsistencies, ambiguities, and incompleteness.

In the past, formal method was not into practice in the software development life cycle due to use of complex mathematical notations; inadequate tools support and too hard to apply. Special training was required to use formal methods to apply in the system development process. Increasingly, number of successful development of techniques and tools related to the formal methods, the industries have started to adopt it for verifying safety properties of the complex systems [Clarke 1996; Bowen 1993; Rushby 1995; Bozzano 2010]. For verifying a critical system, industries are adopting formal methods-based techniques such as model checking and theorem proving in place of the traditional simulation techniques. In both areas related to the model checking and theorem proving,

researchers and practitioners are performing more and more industrial-sized case studies [Bowen 1993; Jetley 2004; Jetley 2006; Cousot 2007; Halbwachs 1991; Macedo 2008; Blanc 2000; Berry 2000], and thereby gaining the benefits of using formal methods.

This chapter contains a concise survey of the literature related to the proposition of this thesis. This survey reviews the existing literature relating to the development and analysis of software for safety critical systems, which identifies current valuable approaches for developing safety critical software, and reviews the methods and analysis techniques available to system developers. Remaining chapter is organized as follows. Section 2 presents the software used in the safety-critical systems and Section 3 presents traditional system engineering approach. Section 4 presents standard design methodologies in a system development process. Section 5 presents industrial application of formal methods, and finally, Section 6 discusses formal methods for safety critical systems.

2.2 Software in Safety-critical Systems

Software is a vital part of any system, especially in embedded systems, where it is used to control the whole functionality of the systems. The embedded systems have major role to control the behavior of the safety critical systems. When we use these systems, we consider that their risk has been minimized and uses of the systems are effectively safe. The system is not only safe, but we also expect other attributes like reliable and cost effective. Main safety-critical systems are commercial aircraft, medical care, train signaling systems, air traffic control, nuclear power, and weapons, where any kind of failure can quickly lead to human life in danger, loss of equipment, and so on. The industries are responsible for designing and delivering the safety-critical systems according to the standards authorities [ISO ; IEEE-SA ; CC ; FDA], which satisfy the requirements.

To address the problem of system's failure related to the software errors for example, overdoses from Therac-25 for treating cancer through radiation [Leveson 1993], the overshooting of the runway at Warsaw airport by an Airbus A320 [Commission 1994], Intel Pentium floating point divide [Price 1995], 5000 adverse events for Insulin Infusion Pump(IIP) reported by FDA [Zhang 2010; Xu 2011] and Ariane 5 flight 501 going off [72]. All these problems and many more are considered as a part of the "software crisis". The term "software crisis" has been introduced in late 1960s to describe the failures of the system in which software-development problems cause the entire system [Gibbs 1994]. In 1968, a meeting is organized by NATO related to the software crisis. This crisis had as its root cause the problem of complexity brought about in many cases by sheer length of programs combined with a poor control over how each line of code affects the overall system. Almost three decades later, this problem still remains as indicated in [Gibbs 1994].

Software crisis is a well-known problem for other engineering disciplines, and over the years of experience has been accumulated to provide effective solutions: the technology has been available, and it has been shown to work with a very high degree of confidence. Software is using frequently in the system development, which is also classified as an engineering discipline, so it would seem natural that one can apply the insights and quickly surmount any hurdles. However, it is true that the engineering insights are applicable to

modern the critical-system development to come over the traditional approach of the system development.

2.2.1 Safety Standards

It is perhaps best to start by considering the various standards that exist for industries, which develop the safety critical systems. Standards are documented agreements containing technical specifications, which produce precise criteria, consistent rules, procedures to ensure reliability, software processes, methods, products, services and use of products, are fit for their purpose in this world. Standards include a set of issues corresponding to the product functionality and compatibility, facilitate interoperability including designing, developing, enhancing, and maintaining. A set of protocols and guidelines, which are produced by the standards, are consistent and universally acceptable for the product development. The standards allow to understand the quality of different products for competing with them and provide a way to verify the credibility of a new product [ISO ; IEEE-SA].

Verification and validation (V & V) are part of the certification process for any critical system. There are several reasons, why certification is required for any critical system. For example, medical device like a cardiac pacemaker must obtain a certificate before implantation. Certification of the product not only assures about the safety, but also helps to a customer to gain confidence to buy and to use the product, which is also important for commercial reasons like having a sales advantage to industry. Certifications are usually carried out by some national and international authorities. Certification can be applicable to an organization, tools or methods, or systems or products. Main objectives of the certification bodies for providing assurance that the organization achieves a certain level of proficiency, and that they agree to the certain standards or criteria. In the case of product certification, there are always issues for the certification, whether a methodology or development process is certified or not.

There is a wide variety of standards bodies. More than 300 software standards and 50 organizations are developing software standards [Fries C. 2011]. Standards come in many different flavours, for example, de-facto standards, local, national and international standards. Some of the standards are more specific related to the defense, financial, medical, nuclear, and transportation etcetera (See in Appendix-A).

There are number of standards addressing safety and security of a system related to the software development. For example, avionics RTCA-Do-178B [RTCA 1992] or the IEC 61508 [IEC61508 2008; Gall 2008] as the fundamental standard for the functional safety of E/E/EP systems [IEC61508 2008; Gall 2008]. The IEC 62304 [IEC62304 2006] standard is for the software life-cycles of medical device development, which addresses to achieve more specific goals through standard process activity, and helps to design the safe systems. All necessary requirements for each life cycle process are provided by the IEC 62304. The process standard IEC 62304 [IEC62304 2006] is a collection of two other standards ISO 14791 and ISO 13485, where the ISO 14791 standard is for quality, and the ISO 13485 is for risk management.

The Institute of Electrical and Electronics Engineers (IEEE) standards [IEEE-SA] provides a safety assurance level for industries, including: power and energy, biomedical and

health care, information technology, transportation, nanotechnology, telecommunication, information assurance, and many more. The IEEE standard is approved by authority and considers the users recommendations before apply into the development process. All these standards are reviewed at least every five years to qualify the new amendments in the systems.

The Food and Drug Administration (FDA) [Keatley 1999] is established by US Department of Health and Human Services (HHS) in 1930 for regulating the various kinds of product like food, cosmetics, medical devices, etcetera. The FDA is now using standards in the regulatory review process to provide a safety to the public before using any product. The FDA provides some guidelines on the recognition to use of and consensus standards. The FDA is interested in standards because they can help to serve as a common yardstick to assist with mutual recognition, based on the signed Mutual Recognition Agreement between the European Union and United States. The FDA standard classifies the medical devices based on risk and the use of medical devices. The FDA provides some standard guidelines for the medical devices, and the medical devices have required to meet these standards. Time to time lots of amendments have been done in the FDA standards [FDA ; Keatley 1999] according to the use of medical devices to provide a safety.

Common Criteria (CC) [CC] is an international standard that allows an evaluation of security for the IT products and technology. The CC is an international standard (ISO/IEC 15408) [ISO] for computer security certification. CC is a collection of existing criteria: European (Information Technology Security Evaluation Criteria)(ITSEC)), US (Trusted Computer Security Evaluation Criteria (TCSEC)) and Canadian (Canadian Trusted Computer Product Evaluation Criteria (CTCPEC)) [CC 2009a; CC 2009b; CC 2009c]. The Common Criteria enable an objective evaluation to validate that a particular product or system satisfies a defined set of security requirements. The CC provides a framework for the computer users, vendors and testing organizations for fulfill their requirements and assures that the process of specification, implementation and testing of the product has been conducted in a rigorous and standard manner.

There are several ways to tackle the complexity issues of software, which major the software at industrial scales and usability of the softwares. The Software Engineering Institute, funded by the military, has produced a Capability Maturity Model (CMM) [Paulk 1995] by which may be assessed the quality of management in a software engineering team. The CMM broadly refers to a process improvement approach that is based on a process model. A process model is a structured collection of practices that describe the characteristics of effective processes; the practices included are those proven by experience to be effective. The CMM can be used to assess an organization against a scale of five process maturity levels. Each level ranks the organization according to its standardization of processes in the subject area being assessed.

2.3 Traditional System Engineering Approach

A critical system uses a standard life-cycle to achieve a certificate from the standard authorities [IEEE-SA ; ISO ; FDA ; CC]. A system can be considered safe if all hazards have

been eliminated, or the risk associated hazards have been reduced to an acceptable level. Software is a part of the system, which is used within the system to operate the system safely. The integrated software within the system does not show any kind of misbehavior. However, if the same software is used by multiple systems then the software must have similar behavior in each system. However, sometimes it is not true. It is believed that each system is different, with different requirements, different risk level with different hazard's characteristics, it is impossible to know if software is safe without considering the behaviour of the software as a part of the system which it is controlling. Therefore, when considering the process for developing a safe software, it is crucial that the whole system of which the software is a part is considered, as well as the software itself [Blanchard 2006]. In the following section, we review a typical safety-critical development process.

2.3.1 The Software Safety Life-cycle

In recognition of the distinctive nature of safety-related systems, there is a standard development process known as V-model, which is widely accepted by large companies and defense. It is an extension of the standard Waterfall model in engineers [Wichmann 1992; Bell 1993; Acuña 2005; Schumann 2001]. The V-model represents a software-development process, where the process steps are bent upwards after the coding phase to form the typical V shape. The V-model presents the relationships between each phase of the development life cycle and its associated phase of testing. V-model is also called verification and validation model (V & V). This process uses a very intensive testing for removing the bug or error, which may appear during any stage of the system development.

The typical process of developing a safety-critical software system is generally time-consuming. Most such processes are based on the V-model, which is illustrated diagrammatically in Fig. 2.1. This model identifies the major elements of the development process and indicates the structured, and typically sequential, nature of the development process. The sequential nature of development is generally considered essential for reasons of managing communication and scale, for scheduling different phases and disciplines, for managing traceability (which is mandated by relevant safety standards) and for the certification purposes.

In order to produce a safety-related software according to this framework, various techniques are recommended. These include the application of structured analysis techniques to generate a visible modular construction (the principles of modularity are expounded in [Parnas 1972]), and diversity in design, implementation and maintenance to avoid faults due to common mode failures. Many such techniques are very widely applicable, and although they are usefully brought into the safety-critical context, there is not so much literature devoted solely to their use in this specific area. Nevertheless, material is available: for instance, there have been reviews such as [Cullyer 1991; Trafford 1997] to help designers and managers as to the suitability of mainstream programming languages for the safety-critical systems.

Safety requires a lot of integrity, and this is recognised in the safety life cycle model which separates the specification of safety requirements into purely functional requirements and safety-integrity requirements. The safety integrity requirements are calculated

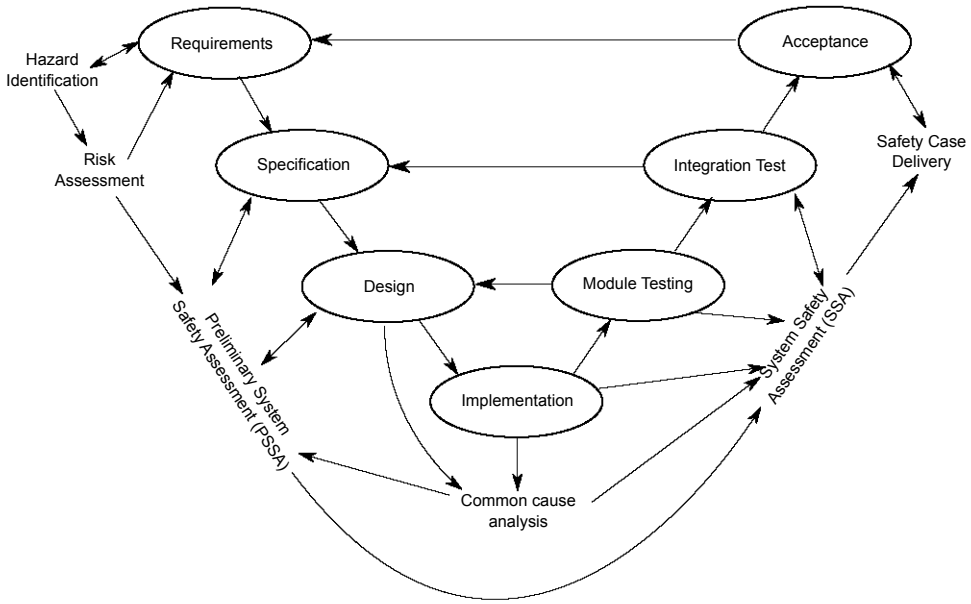


Figure 2.1: The V model of safety-critical system development

individually for each of the functions previously identified. Having done this, one may concentrate on providing the high levels of assurance on the safety-critical aspects. We intend using the safety life cycle model as a basis, with a view to ascertaining its suitability to support the production of formal models with high integrity. Our contention is that we treat carefully the non-functional requirements and to put forward a selection of viewpoints and methods highlighting further the safety concepts, which are often subtle, then the life cycle model can be effective [Trafford 1997]. A safe system can be characterised as one in which risks from hazards have been minimized throughout a system life. The process of providing hazard analyses and risk assessments are thus crucial activities to ensure safety of the system.

2.3.2 Hazards Analysis

Different kinds of techniques may be employed for safety assessment from hazard analysis. When a system has many components, then take a modular approach for analysing a system using System Hazard Analysis (SHA) and Subsystem Hazard Analysis (SSHA). The SHA discovers all associated hazards of the system, while the SSHA discovers how the operation of a given component affects the overall system.

The SHA and SSHA analyses are performed by several techniques, which are provided by the standard authorities. Hazard and operability studies (HAZOP) are more commonly used at the broadest level for analysing process plants like chemical and nuclear industries [Imperial Chemical Industries 1977]. The HAZOP supports the chemical process industry, takes a representation of a system and analyses how its operation may lead to an unsafe deviation from the intent of the system [Imperial Chemical Industries 1977] with special attention to the environment of operation. This technique is very popular in

industries because it aims to predict possible failures, and identify their impact.

Where a system is self-contained, having its boundaries well defined, one focuses on the hazards that are internal to the system, which may be termed faults. Thus, a fault is always a hazard, but not conversely. At this level, we have another technique to analyse the systems using fault tree analysis (FTA) [Leveson 1983]. Fault trees can also be used in a confirmatory role where they are particularly useful in showing that a probability requirement for a hazardous failure mode has been met by the system.

The operation of various components, which is also a HAZOP activity, can be used to determine the possible cause of fault in a fault tree. Failure mode, effects and critically analysis (FMEA and FMECA) are other techniques for hazard analysis, which facilitate the identification of potential problems in the design or process by examining the effects of lower level failures. A more detailed discussion of the system hazard analyses (SHA) with the software perspective is provided in [Imperial Chemical Industries 1977; Leveson 1991].

2.3.3 Risk Assessment and Safety Integrity

A risk assessment is simply a careful examination of the past data related to the hazard's analysis for the similar systems; from the reliability assessments of components of the system being developed; and other sources. The outcome of the risk assessment presents some kind of gradation and may be expressed in terms of what constitutes a tolerable and intolerable risk. This outcome results help for regulating industrial risk, and to determine whether a risk is unacceptable, acceptable or somewhere in between. Lots of factors are used for determining the risk based on quantitative and qualitative analyses [Bell 1993]. Using a risk classification of accidents according to the frequency and severity usefully serves as a relatively simple basis for its determination.

Assessment of the risk can decide a necessary level of safety that we expect the system to achieve from its various functions. This is the issue of safety integrity, which is defined as:

Safety integrity is the likelihood of a safety-related system achieving the required safety functions under all the stated conditions within a stated period of time [Wichmann 1992].

The system activities is contributing to the integrity may be characterised by two kinds of requirements:

1. Generation of the new safety requirements of a system is resulting from the design and development.

2. Ensuring that what is being built meets the requirements that have already been specified.

Here, the first requirement is related to the requirement analysis and hazard analyses of the system. The second requirement is related to the reliability engineering techniques, whose consideration may have to be sustained throughout the development as the design evolves with modification to interfaces, rearrangement of components or other kinds of

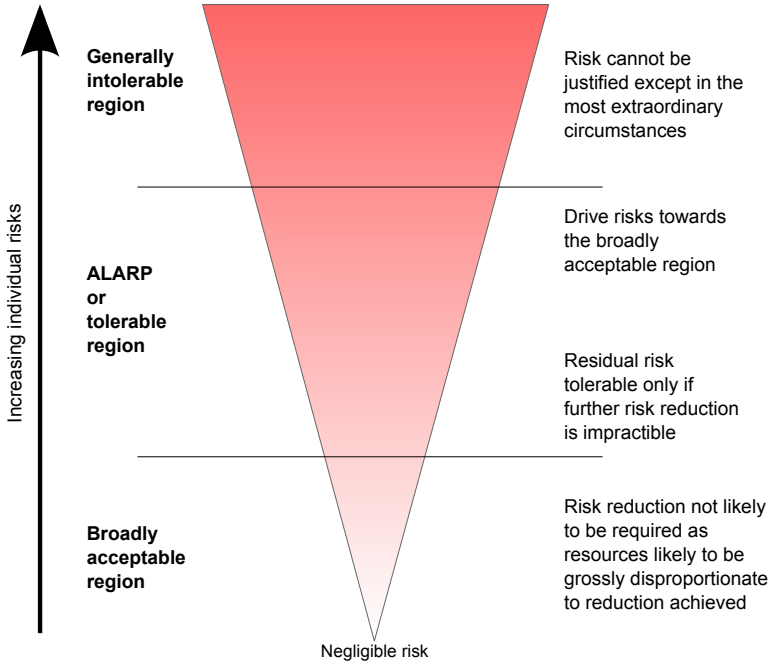


Figure 2.2: ALARP Model of risk level

changes. To apply the several techniques like FMEA/FMECA and FTA for a fault prediction, fault removal, fault avoidance and fault tolerance, and to achieve the system integrity require together with methods and design of the system, are main resources for measuring the system reliability [Trafford 1997].

A safety of the system may be simply characterised by a process of reducing risks to appropriate effect. The main objective of a qualitative or quantitative risk assessment is to establish the level of tolerability for any identified risk. If a risk falls in between the states of 'intolerable' and 'acceptable' then any risk must be reduced to 'as low as reasonably practicable'. This is known as the ALARP principle as illustrated in Fig. 2.2. The width of the triangle is proportionate to the level of risk and thus also to the amount of resources that can be justified to reduce it. A comprehensive survey of risks and safety integrity is provided in [Bell 1993].

In Fig. 2.1, Preliminary System Safety Analysis (PSSA) and System Safety Analysis (SSA) are the collection of various techniques like FTA, HAZOP, FMEA/FMECA, etc. The aim of all these techniques is to identify failures and derive the safety requirements, which prevent from the occurrence of the hazard. FTA focuses on the different components of a system, while HAZOP focus on the flow between components. There are also a number of other techniques, which are used in the PSSA for analysing failures, an overview can be found in [Neumann 1995].

2.3.4 Safety Integrity and Assurance

Finally, there is always a question in the development of the critical system, "What is the assurance level according to the certain level of integrity of the system?". In order to safety assurance of the developed system may be certified as safe, there must be a set of documents, which provides detail justification of the safety. This document contains a list of all hazard's cases with log details and various arguments for indicating that how the system has reached at the required safety levels. The safety case brings in all the aforementioned risk analyses, risk reductions and other integrity and reliability measures, often presenting various statistical evidence. It is a considerable huge amount of a task involves lots of documentation. A software SAM (Safety Arguments Manager) is recognised to support this process and allows to manage all the developing safety cases [Forder 1993].

2.4 Standard Design Methodologies in Development

A design is a meaningful engineering representation of a higher-level interpretation of the system, which is actually the part of the implementation in the source code. Design process is traceable using reverse engineering technique to the actual stakeholder's requirements. The quality of a system can be assessed through predefined criteria for a good design. Analysis and design methods for software have been evolving over the years, each with its approach to modeling the needed world-view into software [NASA Technical Team 2004]. The following methodologies are common, which are used in current practice.

- Structured Analysis and Structured Design (SA/SD)
- Object Oriented Analysis and Object Oriented Design (OOA/OOD)
- Formal Methods (FM) and Model-based Development

SA/SD techniques provide means to create and evaluate a good design of the systems. This technique covers functional decomposition, data flow and information modeling. OOA/OOD considers the whole system into abstract entities called objects, which can contain information (data) and have associated behavior. It is in practice from last 30 years, which is used in several big projects. It contains Object-Oriented Analysis and Design (OOA/OOD) method, Object Modeling Technique (OMT) Object-Oriented Analysis and Design with Applications (OOADA), Object-Oriented Software Engineering (OOSE) and UML. Formal Methods (FM) and Model-based development are a set of techniques and tools based on mathematical modeling and formal logic that are used to specify and verify requirements and designs for the systems and softwares [NASA Technical Team 2004]. Formal method is also a process that allows the logical properties of a computer system to be predicted from a mathematical model of a system by means of a logical calculation. Formal method can be used for formal specification, formal verification and software models (with automatic code generation) [NASA Technical Team 2004].

2.5 Industrial Application of Formal Methods

This section surveys previous works related to the critical system development. A common theme in much of this work is to use formal methods. Formal methods provide a large number of tools and techniques for solving the different kinds of problems. Mainly formal methods are applicable to verification and validation of the system. Formal methods are used to verifying the specification of the system. Although the safety-critical systems have got the confidence in the development due to use of formal methods, such techniques are applicable in a wide variety of application areas in industry. Formal methods have been used to improve the quality of the system as well as verifying the correctness of the system at the early stage of the system development. A set of examples that pioneered the application of formal methods, to more recent examples that illustrate the current state of the art. Here, we have given some list of industrial applications, where formal methods have been used in the projects. A detail survey of all these projects is presented in [Clarke 1996; Bowen 1993; Rushby 1995; Bozzano 2010]

2.5.1 IBM's Customer Information Control System

A successful application of formal methods was the verification of the Customer Information Control System (CICS) in 1980, which was collaborated between Oxford University and IBM Hursley Laboratories [Houston 1991]. The overall system contains more then 750,000 lines of code. Some part of the code was produced from Z specifications, or partially specified in Z, and the resulting specifications were verified using a rigorous approach. Some tools, related to type checking and parsing were developed during the project, which were used to assist the specifier and code inspector. More than 2000 pages of formal specifications were developed for verifying the system. Measurements taken by IBM throughout the development process indicated an overall improvement in the quality of the product, a reduction in the number of errors discovered, and earlier detection of errors found in the process [Clarke 1996]. Furthermore, it was estimated that the use of formal methods reduced 9% of the development cost for the new release of the software.

2.5.2 The Central Control Function Display Information System (CDIS)

The Center Control Function Display (CDIS) System was delivered from Praxis to the UK Civil Aviation Authority in 1992 for London's airspace as a new air traffic management system [Hall 1996]. The CDIS system consists of fault tolerant architecture of a distributed network, where more than 100 computers are linked together. Formal methods were used at various levels of the system development. The requirements analysis phase was represented by formal descriptions using structured notations. The VDM [Bjørner 1978] tool was used for specifying the whole system, which specified concurrent system behaviors. At the product design level, the VDM code was refined into more concrete specifications, and a lower level code was formally specified and developing using CCS [Milner 1982]. The productivity of the system was better than the traditional system development and quality of the system was improved through finding some faults.

2.5.3 The Paris Métro Signaling System (SACEM)

The SACEM system [Hennebert 1993] was developed by several industrial partners GEC Alsthom, MATRA Transport and CSEE (Compagnie des Signaux et d'Entreprises Électriques) in 1989. The system was responsible for controlling the RER commuter train system Paris. The existing system was made of embedded software and hardware, where software had 21000 lines of code. Some part of the SACEM software was formally specified in the B modeling language [Abrial 1996a], which was formally proved.The SACEM project is an example of "reverse engineering" process, where formal specification and verification were conducted after developing the code. Finally, the system was certified by the French railway authority.

ClearSy has developed the screen door controllers for paris metro line using B formal methods [Lecomte 2007]. The models are developed using correct by construction approach and prove the absence of failure in a system behaviour. The constructive process used during system specification and design leads to a high-quality system.

2.5.4 The Traffic Collision Avoidance System (TCAS)

Formal specification of the Traffic Collision Avoidance System (TCAS) [Britt 1994] is another interesting example of the application of formal methods in the air-traffic transport domain. The TCAS system is used by all commercial aircraft for reducing the chance of a midair collision. In early 1990s, the Safety-Critical Systems Research Group at the University of California, produced a formal requirements specification for the TCAS due to occurring some flaws in the original TCAS specification. The formal specification was developed into Requirements State Machine Language (RSML) [Leveson 1994], which is based on a variant of Statecharts [Harel 1987a]. The original specification was not supported by existing formal methods tools, but nevertheless, it was very useful for the project reviewers, in the sense of improving the original specification. Heimdahl et al. [Heimdahl 1996] successfully checked the consistency and completeness of the TCAS specification and provably-correct code generated from the RSML specification.

2.5.5 The Rockwell AAMP5 Microprocessor

The microcode of AAMP5 microprocessor was formally specified and verified, which was produced by Rockwell [Miller 1995]. This project was undertaken by Collins Commercial Avionics (CCA) and SRI. The AAMP5 microprocessor has a complex architecture, designed for ADA language and implements floating-point arithmetic in the microcode. PVS theorem prover [Crow 1995] was used for specifying and verifying the microcode of the AAMP5 instructions.

2.5.6 The VIPER Microprocessor

VIPER microprocessor was developed with a simple architecture, specifically for the safety critical applications [Cullyer 1989]. Formal methods were used throughout the development cycle of VIPER, at the different level using different techniques. This work was

conducted by the Royal Signals and Radar Establishment (RSRE). Some parts of the system were specified by the HOL theorem prover and LCF-LSM language [Gordon 1983]. Mainly top level specification and abstract level view for register transfer level were carried out in the HOL. There was not any significant result through this formal verification except finding some minor flaws in the system, which were no concerns for the fabricators of the chip.

2.5.7 The Mondex Electronic Purse

In this section, we have mentioned the Mondex Electronic Purse as a significant example of the use of formal methods in an industrial-scale application. The Mondex Electronic Purse [Stepney 2000; Butler 2008; Woodcock 2008] is an electronic system for e-commerce, based on smartcard, produced by NatWest Development Team. Electronic purse must ensure the security of each transaction. Formal methods were used by a several group of researchers for verifying the protocol of money transfer over an insecure and lossy medium. The whole formal specification of the Mondex system was developed and proved from the abstract model to concrete model using refinement approach. The abstract model was focused specially on the securities properties of the system.

2.5.8 The BOS Control System

The BOS Software is an automatic system, which is used to protect the harbor of Rotterdam from flooding, while concurrently also controls the ship traffic [Tretmans 2001]. BOS controls a movable barrier, taking decisions of when and how the barrier has to move, based on chaotic behavior of water level, tidal info, and weather preconditions. BOS is a highly critical system, which is characterized by IEC 61508 [IEC61508 2008]. The design and implementation of the BOS were undertaken by CMG Den Haag B.V, in collaboration with a formal methods team at University of Twente. Different kinds of methodologies were applied during development of the system. Mainly formal methods were used to specify the crucial part of the system for validating the system specification. The control part of the system was formally specified in PROMELA and data part into Z specification language. The formal validation of the design focused on the communication protocol between BOS and the environment. The final implementation of the system was done in C++ which was generated from Z specification. At the initial level of the system development, formal methods helped to uncover several issues in the existing system. Overall use of the formal methods improves the quality of the system.

2.5.9 The Intel® Core^{TM} i7 Processor Execution Cluster

Intel Core i7 processor [Kaivola 2009] is used to verify using formal methods. The Intel Code i7 processor is a multi-core processor, where formal methods were used for pre-silicon validation of the execution cluster EXE, a component that is responsible for carrying out data computations for all microinstructions. The EXE cluster implements more than 2,700 microinstructions and supports multi-threading. The formal methods were used here to verify the data-path, control logic, and the state of the components. Formal methods

based on symbolic simulation, and inductive invariants were used in the validation process of the processor. The significant contribution was of this project that the formal verification completely replaced traditional coverage driven testing and proved that formal verification was a viable alternative approach for traditional testing techniques in terms of time and costs with respect to quality of the system.

Here, we have presented a list of projects related the critical system development, which have used the formal methods. All these projects have used different kinds of formal techniques for discovering the bugs at the early stage of the system development and have shown that formal methods could be a significant approach for verifying the systems. Formal method techniques are very expensive and hard to apply in the system development process due to complexity of mathematics and the limitations of existing tools [Overture ; Jones 1986b; Crow 1995]. Main limitations are, each tool based on formal method can be used for only specific purpose, and a formal model developer requires good experience for using formal methods and knowledge of related mathematics. To know the significant use of formal methods [Clarke 1996; Bowen 1993; Rushby 1995; Bozzano 2010] as well as handling its complexity, in this thesis, we propose a new development life-cycle methodology, where each step is based on formal techniques. In this context, we develop a chain of techniques and tools for supporting the system development life-cycle using formal techniques from requirement analysis to code generation.

2.6 Formal Methods for Safety-critical systems

In this section we show the need for the use of formal methods, providing some informal definitions of the main concepts.

2.6.1 Why Formal Methods?

Providing a high integrity system with the embedded softwares requires a careful argument for its justification. Demonstrating the requirements through sufficient statistical evidence based on testing, and other general reliability measures has been shown to be doubtful. Thus, some other kinds of arguments have to be written, which must be precise - in language that is well-defined, whose meaning is clear, and with the ability to prove statements without doubt. Since natural language is unable to fulfill such demands, the only possible solution is to use a mathematical approach - formal methods [Trafford 1997].

A formal approach is ideal for verification, the activity guaranteeing correctness, that we are building the system right and particularly, that successive refinements of a specification are consistent with each other. More than that, the discipline which they encourage often leads to a more careful analysis of the most basic assumptions and definitions in the design, a benefit which is often understated [Trafford 1997]. In particular, they may point to ambiguities in the requirements' definition. Formal methods is thus effective for validation - making sure that we are building the right system [Bowen 1993; Thomas 1993; Hinchey 1995].

The main objective of formal methods is to help developers to build reliable systems. Formal methods is a cutting-edge technology for developing the critical systems, where

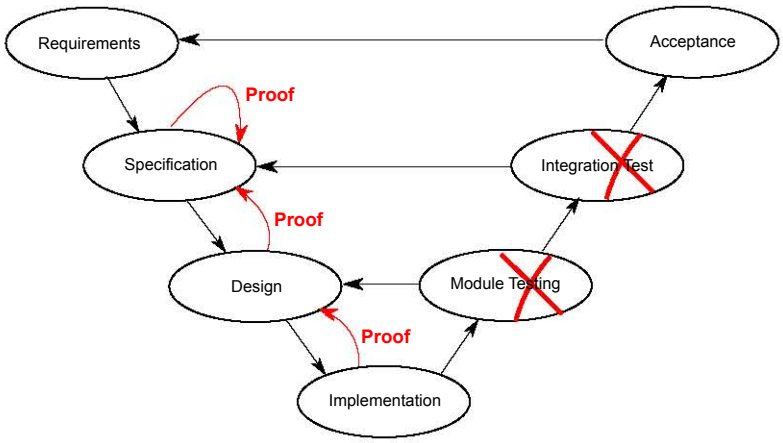


Figure 2.3: The V-model of safety-critical system development using Formal Methods

high safety and security are required. Mathematics is a basic foundation for formal logic, which provides some ways to discover potential errors at the early stage of the development. Fig. 2.3 presents modified V-model after introducing formal methods in a development process. This figure shows that module testing and integration testing are not required due to formally verified system at the specification and design level. Formal methods help to reduce the burden of exhaustive testing, which are used by the traditional development. Formal techniques verify the whole system at the early stage of the system development during specification and design, and to prove the correctness of the system. We cannot say that formal method is a silver bullet, but it is more reliable than the other traditional development approaches. Now formal methods' techniques are feasible to apply for any larger and complex problems.

2.6.2 Motivation for their use

The use of formal methods is very limited in current industrial practice. It is mainly used for verification and validation of any specific part of the system. Specifically, it addresses that the formal methods are not well integrated into established critical system development processes. There are a number of reasons for this. First, the application of formal methods requires high abstraction and mathematical skills to write specifications and conduct proofs, and to read and understand formal specifications and proofs, especially when they are very complex. Second, existing formal methods do not offer usable and effective methods for use in the well-established industrial software process. There are lots of effective tools are available, which are crucial for formal methods application, but existing tools are not able to support a complete formal software-development process, although tools supporting the use of formal methods in limited areas are available [Overture ; Jones 1986b; Crow 1995]. To make formal methods more practical and acceptable in industry, some substantial changes must be made.

This thesis proposes a development life-cycle and a set of associated techniques and tools to develop the highly critical systems using formal techniques from requirements

analysis to automatic source code generation. In this context, we have developed a set of techniques and tools related to Event-B modeling language [Abrial 2010]. Event-B modeling language is only used for verifying the part of the system. There is not a set of supporting tools, which can be used for the formal software development. Our proposed techniques and tools have filled all missing tools and provide a rigorous framework for the system development process. We have applied our approach on two large industrial-scale case studies related to the cardiac pacemaker and adaptive cruise control. These two case studies are related to the medical and automotive domains. Our main objective is to use these two case studies to show the effectiveness of our proposed approach and give the evidence that developed techniques and tools are applicable to any kind of critical system.

In this thesis, we have provided some possible solutions for the emerging problems in the area of software engineering related to the development of the critical systems. We have captured some missing things in the existing tools related to the formal methods, which are essentially required for developing any highly critical system. We have proposed a set of new techniques and tools to model the critical systems, which cover some set of weakness in the existing approach. No one method or tool can serve all purposes. We need to support all different kinds of requirements. From past experience, we have learned what kinds can have the most impact. To be attractive to the practitioners, methods and tools should satisfy the following criteria, where we realize that some of these criteria are ideals, but it is still good to strive for them and some of the basic criteria [Clarke 1996] are required in the development of methods and tools:

1. Methods and tools should provide significant benefits for developing a system, when starting to use them.

2. Helps for writing clear, consistent and unambiguous specifications.

3. It should be possible to amortize the cost of a method or tool over many uses. For example, it should be possible to derive benefits from a single specification at several points in a programme life cycle: in design analysis, code optimization, test case generation, and regression testing. Moreover existing developed specification can be reused for other development processes.

4. Methods and tools should work in conjunction with each other and with common programming languages and techniques. Developers should not have to “buy into” a new methodology completely to begin receiving benefits. The use of tools for formal methods should be integrated with that of tools for traditional software development, for example, compilers and simulators.

5. Notations and tools should provide a starting point for writing formal specifications for developers who would not otherwise write them. The knowledge of formal specifications needed to start realizing benefits should be minimal.

6. Methods and tools should support evolutionary system development by allowing partial specification and analysis of selected aspects of a system.

A new method or tool should have precise strengths and weakness, limitations, modeling assumptions and to support for ease integration with other technique's etcetera. Clear selection criteria help potential users to decide what method or tool is most appropriate for the problem at hand. Given that no formal method is likely to be suitable for describing and analyzing every aspect of a complex system, a practical approach is to use different methods in combination. Based on the results of the survey performed in this chapter it is possible to identify the contribution that this thesis makes. We have given our motivation for developing new techniques and tools as follows:

- **Development Life-cycle Methodology:** This is the heart of the thesis, which presents a methodology for the critical system development from requirement analysis to automatic code generation with standard safety assessment approach. It is an extension of the waterfall model [Wichmann 1992; Bell 1993] with some rigorous approaches to produce a reliable critical system. This methodology combines the refinement approach with a verification tool, model checker tool, real-time animator and finally generates the source code using automatic code generation tools. This kind of approach is very useful to develop the whole system using formal techniques and to verify the complex properties of a system and to discover the potential problems.

- **Refinement Chart:** There are several ways to handle the design complexity of a system. Refinement technique is the most common approach, which facilitates to build a system gradually. We have discovered a very simple way to present the whole system based on operational behavioral using a refinement chart. The refinement chart is a graphical representation of a complex system using layering approach, where functional blocks are divided into multiple simpler blocks in a new refinement level, without changing the original behavior of the system. The final goal to use this refinement chart is to obtain a specification that is detailed enough to be effectively implemented, but also to correctly describe the requirements of the system. The purpose of the refinement chart is to provide an easily manageable representation for different refinements of the system. The refinement chart offers a clear view of assistance in "system" integration. This approach also gives a clear view about the system assembling based on the operating modes and different kinds of features. For example, if a developer does not want to provide any particular feature in any system, then using the refinement chart, it is possible to find that removable feature easily and not to include in the final development system. However, it can also provide the information that, which other parts will be affect-able, when we remove the particular operating modes.

- **Environment Modeling:** The most challenging problem is an environment modeling, for instance, to validate and verify the correct behavior of the system model requires an interactive formal model (an environment formal model). For example, a cardiac pacemaker or cardioverter-defibrillators (ICDs) formal models require a heart model to verify the correctness of the developed system. No any tools and techniques are available to provide an environment modeling to verify the developed system model. The main objective is to use formal approach for modeling the

medical device and biological environment to verify the correctness of the medical systems.

To model a biological environment (the heart) for a cardiac pacemaker or cardioverter-defibrillators (ICDs), we propose a method for modeling a mathematical heart model based on logico-mathematical theory. The heart model is based on electrocardiography analysis [Harrild 2000; Khan 2008; Bayes 2006], which models the heart system at cellular level [Neumann 1966]. The main key feature of this heart model is the representation of all the possible morphological states of the electrocardiogram (ECG) [Bayes 2006; Artigou 2007]. The morphological states represent the normal and abnormal states of the electrocardiogram (ECG). The morphological representation generates any kind of heart model (patients model or normal heart model using ECG). This model can observe a failure of impulse generation and failure of impulse propagation.

- **Automatic source code generation form formal models:** Different kinds of code generation tools are available, and can generate source code into any programming languages but the main constraint is that all those tools are not applicable to generating the codes from Event-B modeling language. Here, our main objective is to develop a set of code generation tools, which can support automatic code generation into several programming languages from Event-B modeling language and supports in the development life-cycle of a critical system from requirement analysis to code generation.

- **Real-time animator:** Lots of formal methods based animator are available for different formal languages. But all kinds animator use only *toy-data* sets. No any tool is available for real-time data testing without generating the source code of the system. We have provided an architecture to use the real-time data set for animation using formal specification. Here, we have discovered that the medical experts are unable to understand the complex formal specification, so that we have proposed a new technique to apply the real-time data set to animate the formal specification and a domain expert can anticipate in the system development. Another objective is to develop this technique also for requirement traceability according to the domain experts. For example, through the animation of the formal model using real-time data set, domain experts can help to find the missing behavior of the system.

- **Integration of different approaches:** We proposed a new framework to compose different kinds of formal techniques to model a critical system to overcome the existing problems. An integration of formal techniques in the development process of a critical system provides the modeling concepts with formal semantics that captures at a high-level of abstraction. Modelling concepts should not be restricted due to apply the verification techniques for checking the correctness of a system. However, the system specifier should have the freedom to use the intuitive modelling concepts neglecting the complexity they impose on verification. The integration framework should bridge the gap between modelling concepts and input required for verification tools. For instance, integration of theorem prover and model checker can be

used for verifying the essential properties. The compositional reasoning strategies using theorem prover and model checking can reduce the verification effort and to verify the required safety properties. The model checker helps to discover lots of errors and strengthening the safety properties through careful cross analysis of the model animation. System specifications are verified by both the model checker and theorem prover tools to prove the absence of any error.

CHAPTER 3

The Modeling Framework : Event-B

"Methods and means cannot be separated from the ultimate aim."
(Emma Goldman)

In this section, we present an overview of the Event-B notation, which has been used for formal development throughout in my case studies. Event-B [Abrial 2010; RODIN 2004] has evolved from Classical B [Abrial 1996a] for specifying and reasoning about reactive systems. Main motivation to select Event-B is targeted at an incremental modelling style [Back 1998a; Back 1979] where a system is defined abstractly, and later more properties are introduced in an incremental fashion with a stepwise introduction of safety properties. The use of refinement represents systems at different levels of abstraction and the use of mathematical proof verifies consistency between refinement levels. Event-B is an event-based approach which is defined in terms of a few simple concepts describing a discrete event system and proof obligations that permit verification of properties of the event system. This chapter explains the fundamental concepts and formal notations of Event-B modelling language. Event-B is provided with tool support in the form of an open and extensible Eclipse-based IDE called Rodin [RODIN 2004] which is a platform for Event-B specification and verification.

3.1 Introduction

3.1.1 Overview of B

Classical B is a state-based method developed by Abrial for specifying, designing and coding software systems. It is based on Zermelo-Fraenkel set theory with the axiom of choice. Sets are used for data modelling, *Generalised Substitutions* are used to describe state modifications, the refinement calculus is used to relate models at varying levels of abstraction, and there are a number of structuring mechanisms (machine, refinement, implementation), which are used in the organisation of a development. The first version of the B method is extensively described in The B-Book [Abrial 1996a]. It is supported by the Atelier B tool [ClearSy] and by the B Toolkit [Ltd 1996].

Central to the classical B approach is the idea of a software operation which will perform according to a given specification if called within a given pre-condition. Subsequent

to the formulation of the classical approach, Abrial and others have developed a more general approach in which the notion of *event* is fundamental. An event has a firing condition (a guard) as opposed to a pre-condition. It may fire when its guard is true. Event based models have proved useful in requirement analysis, modelling distributed systems and in the discovery/design of both distributed and sequential programming algorithms.

After an extensive experience with B, current work by Abrial is proposing the formulation of a second version of the method [Abrial 2003a]. This distills experience gained with the event-based approach and provides a general framework for the development of *discrete systems*. Although this widens the scope of the method, the mathematical foundations of both versions of the method are the same.

3.1.2 Proof-based Development

Proof-based development methods [Back 1979; Abrial 1996a; Morgan 1990] integrate formal proof techniques in the development of software systems. The main idea is to start with a very abstract model of the system under development. Details are gradually added to this first model by building a sequence of more concrete ones. The relationship between two successive models in this sequence is that of *refinement* [Back 1979; Abrial 1996a; Chandy 1988; Back 1998b]. The essence of the refinement relationship is that it preserves already proved *system properties* including safety properties and termination.

A development gives rise to a number of, so-called, *proof obligations*, which guarantee its correctness. Such proof obligations are discharged by the proof tool using automatic and interactive proof procedures supported by a proof engine [ClearSy ; Cle 2004].

At the most abstract level it is obligatory to describe the static properties of a model's data by means of an *invariant* predicate. This gives rise to proof obligations relating to the consistency of the model. They are required to ensure that data properties which are claimed to be invariant are preserved by the events or operations of the model. Each refinement step is associated with a further invariant which relates the data of the more concrete model to that of the abstract model and states any additional invariant properties of the (possibly richer) concrete data model. These invariants, so-called *gluing invariants* are used in the formulation of the refinement proof obligations.

The goal of a B development is to obtain a *proved model*. Since the development process leads to a large number of proof obligations, the mastering of proof complexity is a crucial issue. Even if a proof tool is available, its effective power is limited by classical results over logical theories and we must distribute the complexity of proofs over the components of the current development, e.g. by refinement. Refinement has the potential to decrease the complexity of the proof process whilst allowing for traceability of requirements.

B Models rarely need to make assumptions about the *size* of a system being modelled, e.g. the number of nodes in a network. This is in contrast to model checking approaches [Clarke 2000]. The price to pay is to face possibly complex mathematical theories and difficult proofs. The re-use of developed models and the structuring mechanisms available in B help in decreasing the complexity. Where B has been exercised on known difficult problems, the result has often been a simpler proof development than has been

achieved by users of other more monolithic techniques [Moreau 2001].

3.1.3 Scope of the B Modelling

The scope of the B method concerns the complete process of software and system development. Initially, the B method was mainly restricted to the development of software systems [Behm 1999; Lano 1999; Hoare 1996] but a wider scope for the method has emerged with the incorporation of the event based approach [Abrial 1996b; Abrial 1998; Abrial 2003a; Butler 1998; Butler 2000; Sekerinski 1998] and is related to the systematic derivation of reactive distributed systems. Events are simply expressed in the rich syntax of the B language. Abrial and Mussat [Abrial 1998] introduce elements to handle liveness properties. The refinement of the event-based B method does not deal with fairness constraints but introduces explicit counters to ensure the happening of abstract events, while new events are introduced in a refined model. Among case studies developed in B, we can mention the METEOR project [Behm 1999] for controlling train traffic, the PCI protocol [Cansell 2002], the IEEE 1394 Tree Identify Protocol [Abrial 2003b]. Finally, B has been combined with CSP for handling communications systems [Butler 2000; Butler 1996a] and with action systems [Butler 1998; Sekerinski 1998]. The proposal can be compared to action systems [Back 1998a], UNITY programs [Chandy 1988] and TLA [Lamport 1994] specifications but there is no notion of abstract fairness like in TLA or in UNITY.

3.1.4 Related Techniques

The B method is a state-based method integrating set theory, predicate calculus and generalized substitution language. We briefly compare it to related notations.

Like Z [Spivey 1987; Henson 2007], B is based on the ZF set theory; both notations share the same roots, but we can point to a number of interesting differences. Z expresses state change by use of before and after predicates, whereas the predicate transformer semantics of B allows a notation which is closer to programming. Invariants in Z are incorporated into operation descriptions and alter their meaning, whereas the invariant in B is checked against the state changes described by operations and events to ensure consistency. Finally, B makes a careful distinction between the logical properties of pre-conditions and guards, which are not clearly distinguished in Z.

The refinement calculus used in B for defining the refinement between models in the event-based B approach is very close to Back's action systems, but tool support for action systems appears to be less mechanized than B.

TLA$^+$ [Lamport 2002; Merz 2007] can be compared to B, since it includes set theory with the ε operator of Hilbert. The semantics of TLA temporal operator is expressed over traces of states whereas the semantics of B actions is expressed in the weakest precondition calculus. Both semantics are equivalent with respect to safety properties, but the trace semantics of TLA$^+$ allows an expression of fairness and eventuality properties that is not directly available in B.

VDM [Jones 1986a; Fitzgerald 2007] is a method with similar objectives to classical

B. Like B it uses partial functions to model data, which can lead to meaningless terms and predicates, e.g. when a function is an applied outside its domain. VDM uses a special three valued logic to deal with indefiniteness. B retains classical two valued logic, which simplifies proof at the expense of requiring more care with indefiniteness. Recent approaches to this problem will be mentioned later. ASM [Gurevitch 1995; Börger 2003; Reisig 2007] and B share common objectives related to the design and the analysis of (software/hardware) systems. Both methods bridge the gap between human understanding and formulation of real-world problems and the deployment of their computer-based solutions. Each has a simple scientific foundation: B is based on set theory, and ASM is based on the algebraic framework with an abstract state change mechanism. An Abstract State Machine is defined by a signature, an abstract state, a finite collection of rules and a specific rule; rules provide an operational style very useful for modelling specification and programming mechanisms. Like B, ASM includes a refinement relation for the incremental design of systems; the tool support of ASM is under development, but it allows one to verify and to analyse ASMs. In applications, B seems to be more mature than ASM, even if ASM has several real successes like the validation [Stärk 1998] of Java and the Java Virtual Machine.

3.2 The Event-B Modelling Notation

Event-B [Abrial 2010], unlike Classical B [Abrial 1996a], does not have a fixed syntax. We summarize the concepts of the Event-B modeling language [Cansell 2007; Abrial 2010] developed by Abrial and indicate the links with the tool called Rodin [RODIN 2004]. Here, we present the basic notation for Event-B using some syntax. We proceed like this to improve legibility and help the reader remembering the different constructs of Event-B. The syntax should be understood as a convention for presenting Event-B models in a textual form rather than defining a language.

Event-B [Abrial 2010] modeling language has mainly two main constructs *contexts* and *machines*, which are used to model the whole system. Contexts is used to formalise the static parts of the system, while Machines is used to specify dynamic behavior of the system. Context can be extended by other contexts and referenced by machines. A dynamic part of the system, machines can be refined by machines. Fig. 3.1 depicts basic constructs and their relationship.

3.2.1 Contexts

Contexts express axiomatic static properties of the models. Contexts may contain carrier sets, constants, axioms, and theorems. Axioms describe properties of carrier sets and constants. Theorems are derived properties that can be proved from the axioms. Proof obligations associated with contexts are straightforward: the stated theorems must be proved, which follow from the predefined axioms and theorems. Additionally, a context may be indirectly seen by machines. Namely, a context C can be seen by a machine M indirectly if the machine M explicitly sees a context which is an extension of the context C.

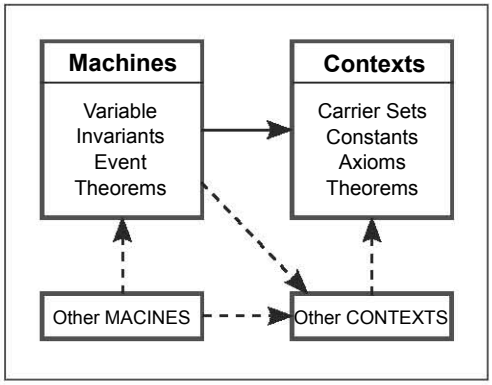


Figure 3.1: Relationship between constructs: Machine and Contexts

3.2.2 Machines

Machines express dynamic behavioural properties of the models, which may contain variables, invariants, theorems, events, and variants. Variables v represents the state of the machine. Variables, like constants, correspond to simple mathematical objects: sets, binary relations, functions, numbers, etcetera. They are constrained by invariants $I(v)$. Invariants are supposed to hold whenever variable values change.

A machine is organizing events modifying state variables, and it uses static informations defined in a context. These basic structure mechanisms are extended by the refinement mechanism which provides a mechanism for relating an abstract model and a concrete model by adding new events or by adding new variables. This mechanism allows us to develop gradually Event-B models and to validate each decision step using the proof tool. The refinement relationship should be expressed as follows: a model M is refined by a model P, when P is simulating M. The final concrete model is close to the behavior of the real system that is executing events using real source code. We give details now on the definition of events, refinement and guidelines for developing complex system models.

3.2.3 Modeling actions over states

The event-driven approach [Cansell 2007; Abrial 2010] is based on the Classical B notation [Abrial 1996a]. It extends the methodological scope of basic concepts to take into account the idea of *formal reactive models*. Briefly, a formal reactive model is characterized by a (finite) list x of *state variables* possibly modified by a (finite) list of *events*, where an invariant $I(x)$ states properties that must always be satisfied by the variables x and *maintained* by the activation of the events. Table-3.1 presents some set-theoretical notations of Event-B [Abrial 2010] and Classical B [Abrial 1996a], which are used to formalise the system. We summarize the definitions and principles of formal models and explain how they can be managed by tools [RODIN 2004].

Each event is composed of a guard $G(t, x)$ and an action $R(t, x)$, where t are local variables the event may contain. The guard states the necessary condition under which an event may occur, and the action describes how the state variables evolve when the event

Name	Syntax	Definition
Binary relation	$s \leftrightarrow t$	$(\mathcal{P})(s \times t)$
Composition of relations	$r_1 ; r_2$	$\{x, y \mid x \in a \ \wedge \ y \in b \wedge$ $\exists z.(z \in c \ \wedge \ x, z \in r_1 \ \wedge \ z, y \in r_2\}$
Inverse relation	r^{-1}	$\{x, y \mid x \in \mathcal{P}(a) \ \wedge \ y \in \mathcal{D}(b) \ \wedge \ a \mapsto b \in r)\}$
Domain	$dom(r)$	$\{a \mid a \in s \ \wedge \ \exists b.(b \in t \ \wedge \ a \mapsto b \in r)\}$
Range	$ran(r)$	$dom(r^{-1})$
Identity	$id(s)$	$\{x, y \mid x \in s \ \wedge \ y \in s \ \wedge \ x = y\}$
Restriction	$s \lhd r$	$id(s); r$
Co-restriction	$r \rhd s$	$r; id(s)$
Anti-restriction	$s \lhd r$	$(dom(r) - s) \lhd r$
Anti-co-restriction	$r \rhd s$	$r \rhd (ran(r) - s)$
Image	$r[w]$	$ran(w \lhd r)$
Overriding	$q \lhd\!\!+ r$	$(\text{dom}(r) \lhd q) \cup r$
Partial function	$s \nrightarrow t$	$\{r \mid r \in s \leftrightarrow t \ \wedge \ (r^{-1}; r) \subseteq id(t)\}$

Table 3.1: Set-theoretical notation

occurs. The first general form for an event is as follows:

$$\textbf{ANY } t \textbf{ WHERE } G(x,t) \textbf{ THEN } x :|(R(x,x',t) \textbf{ END}$$

Second form of an event (e), when event (e) has not any local variable (t), then an event is represented as follows:

$$\textbf{WHEN } G(x) \textbf{ THEN } x :|(R(x,x') \textbf{ END}$$

Third form of an event (e), when event (e) has not any guard (G) and local variable (t), then an event is represented as follows:

$$\textbf{BEGIN } x :|(R(x,x') \textbf{ END}$$

The first form for an event means that it is guarded by a guard that states the necessary condition for this event to occur. The guard is represented by $\exists\, t \cdot G(t,x)$. It defines a possibly nondeterministic event where t represents a vector of distinct local variables. It is also semantically equivalent to $\exists\, t \cdot (G(t,x) \ \wedge \ R(x,x',t))$. In the first, second and third forms, *before-after* predicate $BA(e)(x,x')$, associated with each event e, describes the event as a logical predicate expressing the relationship linking the values of the state variables just before (x) and just after (x') the *execution* of event e. The third form of the event (e) is used for initialisation.

Generalized substitutions are also borrowed from the B notation [Abrial 1996a]. They provide a means to express changes to state variable values. The action of an event is composed of mainly three kinds of assignments : $skip$ (do nothing), deterministic assignment and non-deterministic assignment. Where x is a variable, E is an expression and P is a predicate. The value of x in each case depends on its corresponding expression/predicate. For example, $x :\in E(x,t)$, x will be assigned as an element of $E(x,t)$. In the case of

$x :|(P(x, x')$, x will be assigned as a value satisfying the predicate P. $x :|(P(x, x')$ is a more general substitution form of an assignment predicate. This should be read as *x is modified in such a way that the value of x afterwards, denoted by x', satisfies the predicate* $P(x, x')$, where x' denotes the *new value* of the vector and x denotes its *old value*.

Type	Generalized Substitution
Empty	$skip$
Deterministic	$x := E(x, t)$
Non-deterministic	$x :\in E(x, t)$
	$x :\mid (P(x, x')$

Table 3.2: List of Generalized Substitutions

3.2.4 Proof Obligations

Proof obligations are generated by Rodin tool [RODIN 2004]. Different kinds of proof obligations are produced by Rodin tools, which are as follows: WD (well- definedness), INV (Invariant Preservation), GRD (Guard Strengthening), SIM (Action Simulation), FIS (Feasibility), etcetera. WD proof obligations are generated to ensure that formal predicates and expressions are well defined, which covers generally axioms, invariants, event guards/actions. The Rodin tool supports well-definedness to aid the activities of modelling and proving [Abrial 1996a]. INV proof obligations are generated to guarantee that the invariants are always preserved whenever the machine state changes. Proof obligations (INV 1 and INV 2) are produced by the Rodin tool [RODIN 2004] from events to state that an invariant condition $I(x)$ is preserved. Their general form follows immediately from the definition of the before-after predicate $BA(e)(x, x')$ of each event e and $grd(e)(x)$ is safety of the guard $G(t, x)$ of event e:

(INV1) $Init(x) \Rightarrow I(x)$
(INV2) $I(x) \wedge BA(e)(x, x') \Rightarrow I(x')$

The generated GRD proof obligation ensures that the guard of a concrete event is a correct refinement of the corresponding guard of the abstract event. Finally, the generated SIM proof obligations aim to ensure that the abstract actions are refined correctly by the action of the corresponding concrete event as specified by any gluing invariants. Note that it follows from the two guarded forms of the events that this obligation is trivially discharged when the guard of the event is false. Whenever this is the case, the event is said to be *disabled*.

The proof obligation FIS expresses the feasibility of the event e with respect to the invariant I. By proving feasibility we achieve that $BA(e)(x, y)$ provides an after state whenever $grd(e)(x)$ holds. This means that the guard indeed represents the enabling condition of the event. The intention of specifying a guard of an event is that the event may always occur when the guard is true. There is, however, some interaction between guards and nondeterministic assignments, namely $x :|BA(e)(x, x')$. The predicate $BA(e)(x, x')$

of an action $x :| BA(e)(x, x')$ is not satisfiable or the set S of an action $v :\in S$ is empty. Both cases show violations of the event feasibility proof obligation.

$$(\mathsf{FIS})\ I(x) \ \wedge\ \mathsf{grd}(e)(x) \ \Rightarrow \exists y.BA(e)(x, y)$$

We say that an assignment is feasible if there is an after-state satisfying the corresponding before-after predicate. For each event its feasibility must be proved. Note, that for deterministic assignments, the proof of feasibility is trivial. Furthermore, note that feasibility of the initialisation of a machine yields the existence of an initial state of the machine. It is not necessary to require an extra initialisation.

It is sometimes useful to state that the model which has been defined is deadlock free, that it can run for ever. This is very simply done by stating that the disjunction of the event guards always hold under the properties of the constant and the invariant. This is shown as follows, where $\mathsf{G_1}(e)(x), ..., \mathsf{G_n}(e)(x)$ denote the guards of the events.

$$(\mathsf{DKLF})\ I(x) \Rightarrow \mathsf{G_1}(e)(x) \vee \mathsf{G_2}(e)(x), ..., \mathsf{G_n}(e)(x)$$

3.2.5 Model refinement

The refinement of a formal model allows us to enrich the model via a *step-by-step* approach and is the foundation of our correct-by-construction approach [Leavens 2006]. Refinement provides a way to strengthen invariants and to add details to a model. It is also used to transform an abstract model to a more concrete version by modifying the state description. This is done by extending the list of state variables (possibly suppressing some of them), by refining each abstract event to a set of possible concrete version, and by adding new events. The abstract (x) and concrete (y) state variables are linked by means of a *gluing invariant* $J(x, y)$. A number of proof obligations ensure that,

- each abstract event is correctly refined by its corresponding concrete version
- each new event refines $skip$
- no new event takes control for ever
- relative deadlock freedom is preserved.

Details of the formulation of these proofs follow. We suppose that an abstract model AM with variables x and invariant $I(x)$ is refined by a concrete model CM with variables y and gluing invariant $J(x, y)$. Event e is in abstract model AM and event f is in concrete model CM. Event f refines event e. $BA(e)(x, x')$ and $BA(f)(y, y')$ are predicates of events e and f respectively, we have to prove the following statement, corresponding to proof obligation (1):

$$I(x) \ \wedge\ J(x, y) \ \wedge\ BA(f)(y, y') \ \Rightarrow\ \exists x' \cdot (BA(e)(x, x') \ \wedge\ J(x', y'))$$

The new events introduced in a refinement step can be viewed as hidden events not visible to the environment of a system and are thus outside the control of the environment. In Event-B, requiring a new event to refine $skip$ means that the effect of the new event is not observable in the abstract model. Any number of executions of an internal action may occur in between each execution of a visible action. Now, proof obligation (2) states that $BA(f)(y, y')$ must refine $skip$ $(x' = x)$, generating the following simple statement to prove (2):

$$I(x) \ \wedge \ J(x, y) \ \wedge \ BA(f)(y, y') \ \Rightarrow \ J(x, y')$$

In refining a model, an existing event can be refined by strengthening the guard and/or the before–after predicate (effectively reducing the degree of nondeterminism), or a new event can be added to refine the skip event. The feasibility condition is crucial to avoiding possible states that have no successor, such as division by zero. Furthermore, this refinement guarantees that the set of traces of the refined model contains (up to stuttering) the traces of the resulting model. The refinement of an event e by an event f means that the event f simulates the event e.

The Event-B modeling language is supported by the Rodin platform [RODIN 2004] and has been introduced in publications [Abrial 2010; Abrial 1996a; Cansell 2007], where there are many case studies and discussions about the language itself and the foundations of the Event-B approach. The language of *generalized substitutions* is very rich, enabling the expression of any relation between states in a set-theoretical context. The expressive power of the language leads to a requirement for help in writing relational specifications, which is why we should provide guidelines for assisting the development of Event-B models.

3.2.6 Tools Environments for Event-B

The Event-B modeling language is supported by the Atelier B [ClearSy] environment and by the Rodin platform [RODIN 2004]. Both environments provide facilities for editing machines, refinements, contexts and projects, for generating proof obligations corresponding to a given property, for proving proof obligations in an automatic or/and interactive process and for animating models. The internal prover is shared by the two environments and there are hints generated by the prover interface for helping the interactive proofs. However, the refinement process of machines should be progressive when adding new elements to a given current model and the goal is to distribute the complexity of proofs through the proof-based refinement. These tools are based on logical and semantical concepts of Event-B models (machines, contexts, refinement) and our methodology for modeling medical devices can be built from them.

Part I

Techniques and Tools

CHAPTER 4

Critical System Development Methodology

"The most important single aspect of software development is to be clear about what you are trying to build."

(Bjarne Stroustrup)

Formal methods have emerged as an alternative approach to ensuring the quality and correctness of the high confidence critical systems, overcoming limitations of the traditional validation techniques such as simulation and testing. This chapter presents a methodology for developing a critical system from requirement analysis to automatic code generation with standard safety assessment approach. This methodology combines the refinement approach with a verification tool, model checker tool, real-time animator and finally generates the source code using automatic code generation tools. This approach is very useful to develop the complete system using formal techniques and to verify the complex properties of a system, which helps to discover the potential problems. Remaining chapters of thesis (in Part-I) presents associated techniques and tools to accomplish the requirements of this methodology. Part-II of this thesis use this methodology for the development of the case studies: the cardiac pacemaker and adaptive cruise control(ACC).

4.1 Introduction

Software quality assurance for critical systems is an emerging market. New tools and techniques are developed to provide an assurance that the system will never show any kind of device failures. These tools and techniques are used for designing critical systems like avionics, medical devices and automotive. New developed tools and techniques are varied according to the diversity in the field of critical systems. For example, in the medical domain, small systems like a pacemaker have required different kinds of tools and techniques than the other large systems like imaging for diagnostics or surgery navigation, patient monitoring system etcetera.

Software is an essential part of any critical system, which realizes system's functionality and software reliability for gaining confidence. From last few years, the use of critical systems has been increased [Woodcock 2009]. These devices may sometimes malfunction. Device-related problems are responsible for a large number of accidents. A lot of deaths and injuries have been reported by the US Food and Drug Administration's (FDA) due to failure of the medical devices [Maisel 2001], which advocate safety and security issues for

using it. Certification bodies have found that many accidents and failure of a system, related to the devices, are caused by product design and engineering flaws, considered as the firmware problems [CDRH 2006; NITRD 2009].

Manufacturers have the freedom to tailor the process and to select appropriate methodology according to their specific needs. A lack of information about process and product qualities leads to uncertainness about the appropriateness of the methodology. Software development measures both processes in the quality management plan and associated safety cases related to the approval of the products. Formal methods are usually used for analysing assumptions, relationships, and requirements of a system.

Software certification is performed by certification standards, like FDA, IEC/ISO, IEEE [Keatley 1999; ISO ; IEEE-SA], which do not prove correctness of the system. If a product receives certification, it simply means that it has met all the requirements needed for certification. It does not mean that the product is *fault free*. Therefore, the manufacturer cannot use certification to avoid assuming its legal or moral obligations. A lot of standards consist of functional requirements on particular medical products; there is also a number of standards, which address system safety and software development. For example, IEC-62304 [IEC62304 2006] process standard for quality and risk management of medical devices.

Use of formal methods is very limited in current industrial practice, which addresses that formal methods are not well integrated into established critical system development processes. Formal methods require high abstraction and mathematical skills to write specifications and conduct proofs, and to read and understand formal specifications and proofs, especially when they are very complex are main reasons for not using in practice. Another important cause is that existing formal methods do not offer usable and effective methods for use in the well-established industrial software process. None of the existing tools are able to support a complete formal software-development process, although tools are supporting the use of formal methods in limited areas are available [Overture ; Jones 1986b; Crow 1995]. To make formal methods more practical and acceptable in industry, some substantial changes must be made.

Although formal methods are part of the standard recommendations [NITRD 2009] for developing and certifying critical systems, how to integrate formal methods into the certification process is, in large part, unclear. Especially challenging is how to demonstrate that the final developed system, behaves safely. This chapter describes formal methods based development process that we have applied to produce evidence for the certification, based on the certification bodies [CC ; ISO ; FDA], of a software based critical system. It also describes the most effective aspects of our methodology for certification and research that could significantly increase the utility of formal methods in software certification.

The main contribution of this chapter is to propose a development life-cycle methodology for developing the highly critical software systems using formal methods from requirements analysis to code implementation using rigorous safety assessments. There are not exiting a set of supporting tools, which can be used for developing a system using formal methods. Our proposed techniques and tools have filled all missing tools and provide a rigorous framework for the system development, and to produce evidence for the certification, based on the certification bodies [CC ; ISO ; FDA]. We have applied our proposed

approach on two large industrial-scale case studies related to the cardiac pacemaker and adaptive cruise control for showing the effectiveness of this development life-cycle.

This chapter is organized as follows. Section 2 presents related work and Section 3 describes the heart of the methodology for critical software system development. Section 4 presents benefits of proposed approach. Finally, Section 5 presents concluding remarks of this chapter.

4.2 Related Work

During 1950's and 1960's [Hosier 1961; Royce 1987], main purpose of the software life cycle was to provide a conceptual idea for managing the development of software systems. The conceptual idea was related to the planning, organizing, coordinating, staffing, budgeting and directing the software-development activities. Since the 1960's, different kinds of descriptions and characterizations of the software-development life cycle have emerged [Hosier 1961; Royce 1987; Boehm 1976; Dis ; Scacchi 1984; Sommerville 1995; Royce 1987].

Mainly, four traditional models of software evolution are very popular from earliest days of software engineering. These four models are waterfall model, stepwise refinement model, incremental release model and military standards based model. The most familiar software-development model is the waterfall model, which was originated by Royce [Royce 1987]. This model presents a way for developing the large complex software systems using an iterative process. Stepwise refinement model develops the software system through the progressive refinement and helps in enhancement of the high-level system specifications into source code components [Wirth 1971; Mili 1986]. The refinement process is undefined, while it is used during the development process, and formalization is expected to apply heuristically according to the expertise and acquired skills. These two models are effective and widely applied in the current practices of the software engineering [Marciniak 2002]. The incremental release model is mostly applied into industrial practices. Developing systems through incremental release provides a foundation level for essential operating functions, then enriching the system functionalities at the regular intervals [Basili 1975]. This model combines the classic software life-cycle with an iterative enhancement at the level of system development organization. Periodical software maintenance and services are also supported by this incremental release model. Military standards based models are the refined form of the classical life-cycle models, which eliminate complications that emerge during large software development. Since 1970's many government contractors use military standards for developing the large software systems [Moore 1997]. Military software system is not commonly used in the industrial and academic practices. Mainly, it is used for : (1) meeting required military standards; (2) developing complex embedded systems (e.g., airplanes, submarines, missiles, command and control systems) which are mission-critical; and (3) developing under contract to private firms through cumbersome procurement and acquisition procedures that can be subject to public scrutiny and legislative intervention [Moore 1997].

All these four models are used for coarse-grain characterizations of the software evo-

lution. The primary progressive steps of software evolution are requirements specification, design, and implementation. Moreover, these models are independent of any organizational development setting, software application domain, choice of programming language, etc. However, all of these life cycle models have been in use for some time, we refer to them as the traditional models, which are used for software evolution [Marciniak 2002].

There have been several efforts involving the use of formal methods to verify safety-critical systems. Formal methods have been used to handle complex safety-critical systems, for instance, steam boiler control [Abrial 1996c], Siemens Transportation [Badeau 2005], and space and avionic system [Butler 1996b; Miller 1998; Cousot 2007; Halbwachs 1991; Joshi 2005] so on. Various formalisms and rigorous techniques (VDM, Z, ASTRÉE, SCADE Event-B, Alloy, CSP, etc.) have been used in the development process of safety-critical systems. These approaches provide a given level of reliability and confidence to develop an error-free system. Few case studies show that the formal methods have been used to check correctness of operating modes, functions and desired behaviors of the medical devices [Gomes 2009; Macedo 2008; Bowen 1993; Jetley 2004; Jetley 2006; Magee 2003].

C.L. Heitmeyer et al. [Heitmeyer 2009; Heitmeyer 2008] have presented an approach for software certification using formal methods. They describe how formal methods are used to produce evidence in a certification, based on facts of a safety-critical software system. The evidence includes a top-level specification (TLS) of the safety-relevant software behavior, a formal statement of required safety properties, proofs that the specification satisfied properties, and a demonstration that the source code, which has been annotated with preconditions and post-conditions, is a refinement of the top-level specification (TLS) [Heitmeyer 2008]. A research report [Rushby 1995] is presented by John Rushby, which is based on certification issues for advanced technology. Its purpose is to explain the use of formal methods in the specification and verification of software and hardware requirements, designs, and implementations, to identify benefits, weaknesses, and difficulties in applying these methods to digital systems used in critical applications, and to suggest factors for consideration when formal methods are offered in support of certification.

Maibaum and Wassyng [Maibaum 2008] have proposed that the assessment procedure should focus on activities and as well as a product should be reviewed according to the domain requirements based on profound engineering expertise. Assessment procedure should consider all relevant risks, including suitable activities in the development plan, selected techniques should be appropriate for activities and safety classification, and acquired evidence of software development supports validity of the arguments.

Intermediate steps during the development of the final product require several kinds of methodologies and approaches, which are also useful for providing certain facts that help for certifying the final product. For example, a requirement specification, a design specification, a document describing validation of the design against the requirements, documents relating to testing, and documents proving correctness [Huhn 2010; NITRD 2009].

According to the literature survey, different kinds of arguments insist to design a new methodology, which may be able to design a consistent formal model based on domain-specific requirements and helps to certify a system. In this context, we proposed a development life-cycle methodology for developing a critical system using formal methods. We believe that our methodology adds value with its comprehensiveness; it focuses on

correctness by establishing the standards to perform software certification. It uniformly establishes what to check and how to check it and provides certain evidence of correctness.

4.3 Overview of the Methodology

In recent years, the critical systems have grown more complex and providing certification assurance, is a common crucial issue for certification bodies [Keatley 1999; ISO ; IEEE-SA ; FDA ; NITRD 2009]. Under consideration of all kinds of requirements of certification bodies, we propose a novel development methodology that addresses the issue of certification for all kinds of critical systems, which is an extension of the waterfall model [Acuña 2005; Schumann 2001; Wichmann 1992; Bell 1993] for developing a critical software system using formal methods and standard safety assessment approaches [Leveson 1991] from requirement analysis to final system implementation.

This development process is based on refinement approach, where we have introduced some new steps for designing the complete system using formal verification, validation and real-time animation [Méry 2010b]. All these steps are not only used in the development life-cycle, but they are also validating the correctness of a system, and all these processes are moreover verified by safety assessment techniques, which comply with software standards. Basic architecture of the methodology is depicted in Fig. 4.1, which may be used in the development process of a critical system.

In this development methodology, we have considered mainly two types of development: static development and dynamic development. Each phase includes capturing of requirements. The static development refers to the straight-forward process, which produces a program, and dynamic development refers to the activities that improve the quality of the program using refinement approach until it satisfies user requirements. In order to reach the required safety level and gain reliability, we have used standard safety assessment approaches in the development process, and also ensuring traceability between the different stages of the system development in order to reduce the validation effort. Different phases of the methodology are shown in Fig. 4.1 which is used in the development process of a critical system. Seven main phases of proposed methodology are described as follows:

4.3.1 Informal Requirements

The first activity of static development captures user requirements as completely as possible, which is an initial phase of the proposed methodology, presents an informal requirements of a given system. Software requirements specifications are widely used in a restricted form of natural language. Natural language is convenient because it allows non-technical users to understand systems requirements. On the other hand, the lack of precise semantics increases the possibility of errors being introduced due to interpretation mistakes and inherent ambiguities. Under or over specification are also common problems when using a natural language. Software requirements specification consists of the categorization and structuring of the informal requirements fragment described in the requirements document to produce categorized requirements fragments. The main objective of informal requirements is to provide a precise, yet understandable description of the safety-relevant

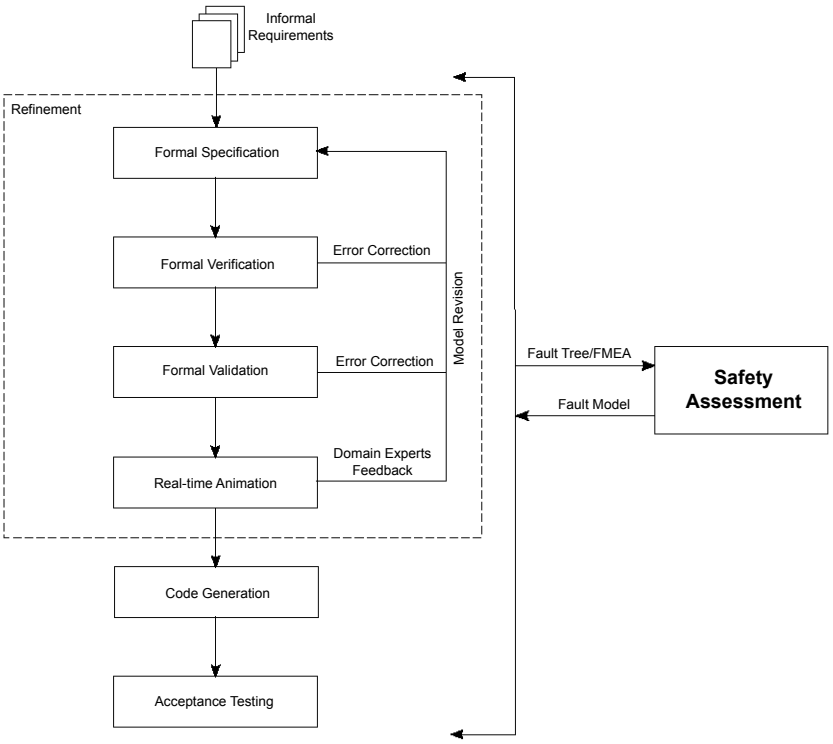


Figure 4.1: Formal methods based development methodology of critical system

behavior of a system and to make explicit assumptions on which the safety of a system is based. Once the developer reaches a general understanding of the problem, static development proceeds to the formal requirements specification.

4.3.2 Formal Specification

In our methodology, the required security requirements are formally expressed as properties of the state-based model that underlies the informal requirements. The categorized requirements fragments are described through the set of formal notations in any modeling language like Event-B, Z, ASM, VDM, TLA$^+$ etcetera. Formal specification languages have a mathematical (usually formal logic) basis and employ a formal notation to express system requirements. The formal specification is typically a mathematical based description of the system behavior, using state tables or mathematical logic. In this stage, more detailed requirements analysis is carried out by building a formal specification. Using the formal notation, precision and conciseness of specifications can be achieved. Formal specification will not normally describe lowest level software, such as mathematical subroutine packages or data structure manipulation, but will describe the response of the system to events and inputs, to a degree, necessary to establish the critical properties. For instance, in a cardiac pacemaker the sensor and actuator are functioning correctly. This specification reflects the primary user requirements derived through successive evolutions, each of which transforms an abstract specification to become more concrete. Evolution steps may

involve the usual notion of formal refinement, and may also involve introducing additional constraints required in the final solution. Hence, a specification during the evolution process is considered to provide the functional constraints on the final concrete specification rather than a complete description.

4.3.3 Formal Verification

This phase has a very important role in the formal development. To demonstrate that the informal requirements satisfy the safety properties of interest, the informal requirements and the properties are passed to a theorem prover and then prover is applied to prove formally that the informal requirements satisfy the properties. A formal notation can be analysed and manipulated using mathematical operators, mathematical proof procedures can be used to test (and prove) the internal consistency (including data conservation) and syntactic correctness of the specifications. Furthermore, the completeness of the specification can be checked in the sense that all enumerated options and elements have been specified. However, no specification language can ensure completeness in the sense that all the user's requirements have been met, because of the informal human-intention nature of the requirements specifications [Jackson 2007]. Finally, the implementation of the system will be in a formal language (i.e., the programming language), it is easier to avoid misconceptions and ambiguities in crossing the divide from formal specifications to formal implementations. A formal verification phase is done to ensure that,

- The model is designed correctly
- The algorithms have been implemented properly
- The model does not contain errors, oversights, or bugs

In summary, we can say that verification ensures that the specification is complete and that mistakes have not been made in implementing the model. But verification does not ensure the model,

- Solves an important problem
- Meets a specified set of model requirements
- Correctly reflects the working of a real world process

In Fig. 4.1, this phase gives the feedback to the formalization phase in case of not satisfying given properties of the system. The feedback approach is allowed to modify the formal model and verify again it through the formal verification phase. The verification process is applied to continue until not to find the correct formal model according to the expected behavior of the system.

4.3.4 Formal Validation

Formal validation phase is the process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model that is not covered by the formal verification. It consists of the identification of a subset of the formalized requirements fragments for an automatic validation analysis:

- Validation ensures that the model meets its intended requirements in terms of the methods employed and the results obtained.
- The ultimate goal of model validation is to make the model useful in the sense that the model addresses the right problem, provides accurate information about the system being modeled, and to make the model actually used.

Model checking [Clarke 1999] is a complementary technique for validation and verification of a formal specification. Model checkers attempt to make formal techniques easier to use by providing a high degree of automation at the expense of generality. Inputs to a model checker are typically a finite state model of a system, along with a set of properties that are expected to be preserved by the system. Properties to be verified can be usually categorized as one of the following:

1. Correct sequences of events
2. Proper consequences of activities
3. Simultaneous occurrences of particular events
4. Mutual exclusion of particular events
5. Required precedence of activities

The model checker explores all the possible event sequences of a model to determine that system is always holding required safety properties. If properties hold, the model checker confirms the correctness of a system. If a property fails to hold for some possible event sequences, the tool produces counter-examples, i.e., traces of event sequences that lead to failure of the property [ProB].

In Fig. 4.1, this phase also presents that it provides a feedback information to the formalization phase when a model is not satisfying expected behavior of a system. This feedback information helps to modify the developed formal model and verify it through the formal verification phase and validation through a model checker tool [Clarke 1999; ProB ; Leuschel 2003]. This cyclic process for finding a correct model is applied to continue until not to find the correct formal model according to the expected system behavior.

4.3.5 Real-time Animation Phase

This phase is a new validation technique to verify the formal model in the real-time environment using *real-time data* set instead of using *toy-data* set. This phase is applied for

rigorous testing of the formal model under domain expert's reviews. Real-time animation shows the behaviors of the system using real environment in early phase of the development without generating the source code. Such kind of techniques are very useful when domain experts are also involved in the system development [Méry 2010b]. Formal models are used to animate with the real-time animator [Méry 2010b], in order to verify on given scenarios according to the real system. This model animator is not part of the validation process, as this can be required to qualify as software requirements, but it helps us to check models against reality and to internally verify their suitability.

In Fig. 4.1, this step also presents feedback loop into the formalization phase to correct an unexpected error of the system. The feedback approach is allowed to modify the formal model and verify it using any theorem prover tool and finally validate it using a model checker tool. In this phase of the formal development, most of the errors are discovered by the domain experts. The verification, validation and real-time animation processes are applied to continue until not find the correct formal model according to the domain experts.

Most simulation researchers agree that animation may be dangerous too, as the analysts and users tend to concentrate on very short simulation runs so the problems that occur only in long runs go unnoticed. Of course, good analysts, who are aware of this danger, will continue the run long enough to create a rare event until not cover all possible events, which is then displayed to the users. Each phase of the methodology is supported by a specific tool.

4.3.6 Code Generation

The final stage of static development is an implementation, in which a program is constructed to realize the design. This involves the realization of the major executable components, definition of the concrete data structures, and an implementation of any minor auxiliary structures or functions that may be assumed in the design. It is important to verify the design and program against the requirements specification and design through rigorous reviews. Automatic code generation [Berry 2007] is well known process to get highly verified codes according to the requirements analysis. It is an important part of the software-development process, for implementing the final system. There are several reasons for using an automatic code generator in the development of the safety-critical systems. Automatic code generation techniques produce the executable specification with fewer numbers of implementation errors than a human programmer. Manual translation process can be error prone and time consuming. An iterative process for successive changes in the specification and manual code translation can introduce errors by various ways. The consistency between the specification and code translation is often lost. This is not the case when an automatic code generator is used to obtain a source code from verified specification. The code generator translates a proved formal model directly into a desired programming language. It ensures that the generated code is always consistent with the formal model. Code generation from the verified formal model is our final objective of this methodology.

4.3.7 Acceptance Testing

In the system development process, acceptance testing are used to determine if the requirements of a specification are met. Acceptance tests represent the customer's interests. The acceptance tests give the customer confidence that the application has the required features and that they behave correctly. In theory when all the acceptance tests pass the project is done [Software 2001]. Acceptance testing in software engineering generally involves execution of number of test cases which constitute to a particular functionality based on the requirements specified by the user. In system engineering process it may involve black-box testing performed on a system. It is also known as functional testing, black-box testing, QA testing, application testing, confidence testing, final testing, validation testing, or factory acceptance testing [Bertolino 1997].

Dynamic development is the process to discover hidden features of the system through applying several kinds of tools like formal verification, model checker and real-time animator in the system development process. This is an iterative way for acquiring more requirements according to the stakeholders and to verify the correctness of the system. Real-time animator based on formal methods is used here as a prototype model in the early stage of the system development. Prototyping can also be used for risk analysis, but the use of formal methods can improve the quality of prototypes. After code generation, the final system can be used to verify the system against both the informal user requirements (system testing) and the end-user (acceptance testing).

In this proposed methodology, we have used combined approach of formal proofs and rigorous reviews for a system development. The purpose of formal proof and rigorous reviews is to ensure the internal consistency of the specifications at different levels of development, to validate the specification against user requirements and certification standards, and to ensure that the designs and programs satisfy their requirements specifications.

4.4 Benefits of Using our Proposed Approach

In this methodology, we have provided an architecture to develop a critical system (see Fig. 4.1). Our methodology has the potential for improving quality and increasing safety for certifying the highly critical systems. Specific benefits include improving requirements, reducing error introduction, improving error detection, and reducing cost. Secondly, the proposed architecture of methodology allows us to carry out rigorous analyses. Such analyses can verify useful properties such as consistency, deadlock-freedom, satisfaction of high level requirements, correctness of a proposed system design and expected system behavior according to the domain experts using real-time environment in early phase of the development without generating the source code.

4.4.1 Improving Requirements

Using our methodology to capture requirements provides a simple validation check in early stage of critical-system development. Requirements expressed in a formal notation can also

be analysed early to detect inconsistency and incompleteness for removing errors that are normally found later in the development process.

4.4.2 Reducing Error Introduction

Formalized requirements prevent misunderstandings due to ambiguities that lead to an error introduction. As development proceeds, compliance can be continually checked using a formal analysis to ensure that errors have not been introduced. A further advantage of using our methodology at the requirements level is the ability to derive or refine from these requirements the code itself, thus ensuring that no error is introduced at this stage. Alternatively their use at the requirements level allows formal analysis to establish correctness between requirements and final generated source code of the complex system.

4.4.3 Improving Error Detection

Out methodology can provide exhaustive verification at whatever levels it is applied: high level requirements or low level requirements. Exhaustive verification means that the whole structure is verified over all possible inputs and states. This can detect errors that would be difficult or impossible to find using only a test based approach.

4.4.4 Reducing Development Cost

Our proposed methodology is based on formal techniques. In general, software errors are less expensive to correct the earlier in the development life-cycle they are detected in critical systems. The effort required to generate formal models is generally more than offset by the early identification of errors. That is, when formal methods are used early in the life-cycle, they can reduce the overall cost of the project development. When requirements have been formalized the costs of downstream activities are reduced. Formal notations also reduce cost by enabling the automation of verification activities.

4.5 Conclusion

One valuable byproduct of applying formal methods in software certification is that the process produces a formal specification of the required software behavior. Developing this specification has at least two benefits. First, a formal specification can be valuable when a new version of the software is developed. Second, the process of developing a formal specification by itself may expose errors.

A new methodology for developing a critical system using formal methods, which is an extension of waterfall model [Acuña 2005; Schumann 2001; Wichmann 1992; Bell 1993] has been described to applying formal methods in critical software development process from requirement analysis to automatic source code generation under rigorous analysis of the development process using standard safety assessment techniques. This development methodology combines the refinement approach with a verification tool, model checker

tool, real-time animator and finally generates the source code using automatic code generation tool. System development process is concurrently assessed by safety assessment approach [Leveson 1991] to comply with certificate standards. This life-cycle methodology consists of seven main phases: informal requirements, formal specification, formal verification, formal validation, real-time animation, automatic code generation and acceptance testing. This kind of approach is very useful to verify complex properties of a system and to discover the potential problems like deadlock and liveness at early stage of the system development.

Building on existing software certification standards, such as IEC-62304 and the Common Criteria [IEC62304 2006; CC ; Farn 2004], more and improved approaches which use formal methods in software certification are needed. Applying these new approaches for highly critical systems should have many benefits; the exposure of errors which might have not been detected without formal methods. That guidance, as proposed by NITRD, IEEE, and IEC/ISO [NITRD 2009; IEEE-SA ; ISO], allows adoption of formal methods into an established set of processes for the development and verification of a critical system to be an evolutionary refinement rather than an abrupt change of methodology. Formal methods might be used in a very selective manner to partially address a small set of objectives, or might be the primary source of evidence for the satisfaction of many of the objectives concerned with development and verification.

Two reliable facts of formal methods have demonstrated by last decades of research and experience - they are not the "*silver bullet*" to eliminate all software failures, but neither are they beyond the budget constraints of software developers. In critical system, formal methods are commonly demonstrating the absence of undesired behaviors and preserving essential properties. Model checkers, theorem provers, real-time animation and automatic code generation make it possible to analyse the complexity of the system and produce the final implemented system. On the other hand, the ability to generate complete test cases from formal specifications can result in overall savings, despite the cost of developing the specification. The process of developing a specification is often the most valuable phase of a formal verification, and "lightweight formal methods" approaches make it possible to formally analyse partial specifications and early requirements definitions. Experience with mandated use of formal techniques and other standards provides empirical evidence that these methods can be successfully incorporated into the development process for the critical systems. Remaining chapters include detailed descriptions of the new associated techniques and tools, which are used in this development methodology for critical system development.

CHAPTER 5

Real-Time Animator and Requirements Traceability

"The soul never thinks without a picture."
(Aristotle)

According to the development life cycle of a critical system, first of all we emphasize on requirements traceability using a real-time animator [Méry 2010b]. Formal modeling of requirements is a challenging task, which is used to reasoning in earlier phases of the system development and to make sure that the completeness, consistency, and automated verification of the requirements. This is an initial step in the proposed development methodology of the critical system development. The real-time animation of a formal model has been recognized to be a promising approach to support the process of validation of requirements specification. It is crucial to get an approval and feedback when domain experts have a lack of knowledge of any specification language, to avoid the cost of changing a specification at the later stage of development. This chapter introduces a new architecture, together with a direct and an efficient method of using real-time data set, in a formal model without generating the source code in any target language. This is a phase for validating a system through domain experts in our development life-cycle methodology. The principle is to simulate the desired behaviors of a given system using formal models in the real-time environment and to visualize the simulation in some form appealing to stakeholders. The real-time environment assists in the construction, clarification, validation and visualization of a formal specification.

5.1 Introduction

Formal methods aim to improve software quality and to produce *zero-defect* software, by controlling the whole software-development process, from specifications to implementations. In formal model development, they use top-down approaches and start from high-level and abstract specifications, by describing the fundamental properties of the final system. Requirements Engineering (RE) provides a framework for simplifying a complex system to get a better understanding and to develop the quality systems. The role of verification and validation is very important in the development of safety critical systems. Ver-

ification starts from the requirements analysis stage where design reviews and checklists are used for validation where functional testing and environmental modelling are done.

There are several ways to validate a specification: to model a prototype of a system, structured walk-through, transformation into a graphical language, animation, and others. Each technique has a common goal, to validate a system according to the operational requirements. Animation focuses on the observable behaviour of the system [Van 2004]. The principle is to simulate an executable version of the requirements model and to visualize exact behaviours of the actual system. Animators use finite state machines to generate a simulation process which can be then observed with the help of UML diagrams, textual interfaces, or graphical animations [Ponsard 2005]. Animation can be used in the early stage of development during the elaboration of the specification. As a relatively low cost activity, animation can be frequently used during the process to validate important refinement steps.

The final code generation process consists of two stages: final level formal specifications are translated into programs in a given programming language, and then these programs are compiled. Nevertheless, all approaches which support a formal development from specification to code must manage several constraining requirements, particularly in the domain of embedded software where specific properties on the code are expected likes timeliness, concurrency, liveness, reactivity, and heterogeneity [Lee 2002]. All these properties can be represented abstractly. Finally, it is impossible to use the real-time data in the early stage of formal development without compiling the source code in any target language.

Based on our various research experience using formal tools in industrial requirements (*verification* and *validation*) and our desire to disseminate formal methods, we have imagined a new approach to present an animated model of specification using real-time data set, in the early stage of formal development [Méry 2010b]. Animation of formal specification is not a new idea, but capture the real-time data and perform animation of formal methods in the real environment is a new idea. In this work, we present an architecture which allows to easily develop the visualizations for a given specification with support of existing tool Brama [Servat 2006]. Here, we describe an approach to extend an animator tool which will be useful to use the real-time data set. Now, present time all the animation tools use a *toy data* set to test the model while we are proposing a *key idea* to use the real-time data set with the model without generating the source code in any target language (C, C++, VHDL, etc.). It can help a specifier to gain confidence that the model that is being specified, refined and implemented, does meet the domain requirements.

This architecture supports state-based animations, using simple pictures to represent a specific state of a Event-B [Abrial 2010] specification, and transition-based animations consisting of picture sequences by using real-time data set. Picture sequences is controlled by the real-time data set, and it presents an actual view of a system. Before moving on we should also mention that there are scientific and legal applications as well, where the formal model based animation can be used to simulate (or emulate) certain scenarios to glean more information or better understandings of the system requirements. The real-time animation tool is very helpful for evidence based validation as well as in the certification process, which is provided by FDA's QSR, ISO/IEC' and IEEE standards quality system directives. This technique uniformly establishes what to check and how to check it and

provides certain evidence of correctness.

This chapter is organized as follows. Section 2 presents motivations behind this work. Section 3 presents basic definition of traceability and Section 4 presents related work. Section 5 presents basic details about animation and their benefits and limitations. Section 6 presents the functional architecture which enables the animation of a proved specification with a real-time data set. The functional architecture is then illustrated in Section 7 for applications and case studies. Section 8 presents limitations of this tool and finally Section 8 concludes the chapter.

5.2 Motivation

To discover the real requirements, discover errors in the early stage of the system development and design a quality system, we need to look beyond the system itself, and into the human activities that it will support. For example, medical systems are mainly used by doctors, physicians, medical practitioners and patients in more convenient ways for their own purpose. The medical device manufacturing companies are providing safe, secure and profitable services to stakeholders. Such human activities may be complex due to several involvements of many people with different types of conflicts of interests. In this situation, it is hard to handle any problem and to reach final agreement among the stakeholders. Requirements engineering techniques offer ways to handle complex problems by decomposing into simple ones, so that we can understand them better. Complexity of a system classifies it into a specific class of problems known as *wicked problems* [Rittel 1973]. This term was introduced by Rittel and Webber [Rittel 1973] for problems that have the following characteristics:

- There is no proper definition of the problem- such as each stakeholders have their own definition of the same problem.
- Wicked problems have no stopping rule - each solution is likely to extend into a new set of problems, and the problem is never likely to be solved entirely.
- Solutions are not exactly in the form of right or wrong, but it provides for better or worse solutions.
- There is no any fixed standard for a particular solution. Solution results are depended on the judgment of various stakeholders according to their needs.
- For wicked problems, there is no any fixed enumerable solutions. The solutions are discovered during problem analysis.
- Every wicked problem is unique and considered as sufficiently complex and different from others.
- Every wicked problem is a symptom of another problem, which makes difficult to choose an appropriate level of abstraction for describing the problem.

- The designer has no 'right' to be wrong. In other words, designers are liable for the consequences of the actions they generate.

Requirement Engineering (RE) techniques are used at the early stages of the system development life cycle, which are crucial for successful development of a system. As the computer systems play increasingly important roles in organizations, it seems to pay more attention towards the early stages of Requirement Engineering (e.g., [Bubenko 1995]). The cost of the system development increases more when errors are discovered later phases of the system development [Boehm 1981]. The basic beginning objective of stakeholders and the initial requirements statement of requirement engineering are "what the system should do?". Incompleteness, ambiguity, inconsistencies, and vagueness, are most common problems encountered when eliciting and specifying requirements and for finding these common problems are main goals of any requirement engineering tool [Bubenko 1995].

A model captures a view of a physical system. It is an abstraction of the physical system, with a certain purpose. Thus the model completely describes those aspects of the physical system that are relevant to the purpose of the model, at an appropriate level of detail. Not only a system developer is required to view a system from several angles but stakeholders want the same view of the system from different angles according to the requirements. Requirements traceability is a branch of requirements management within software development. Requirements traceability is concerned with documenting the life of a requirement and to provide bi-directional traceability between various associated requirements. It enables users to find the origin of each requirement and track every change, which was made to this requirement.

Validation of the requirements specification is an integral and indispensable part of the Requirements Engineering (RE). Validation is the process of checking, together with the stakeholders, whether the requirements specification meets the stakeholders' intentions and expectations [McDermid 1991]. Animation of a formal specification is one of the well-known approaches in the area of verification and validation, which provides visual animation of the formal models. An architecture of the real-time animation tool is presented in [Méry 2010b], that allows to check the presence of desired functionality and to inspect the behavior of a specification according to the stakeholders in the real-time environment.

The contribution of this chapter is to propose a new functional architecture, together with a direct and an efficient method of using real-time data set, in a formal model without generating the source code in any target programming language[Méry 2010b]. Real-time animation helps to design a critical system, which helps a specifier to gain confidence that the model is being specified, refined and implemented, does meet the domain expert's requirements. Main objective of this proposed real-time animation framework bridges the gap among different domain experts. For example, in the development of a medical system [Keatley 1999], a formal model that is designed by a software engineer is not understandable by the medical experts like doctors or medical practitioners due to lack of mathematical knowledge. If a software engineer presents a formal specification into animated graphics, based on actual behavior of the formal specification, then the animated graphics can be simpler and easily understandable by the doctors, physicians and medical practitioners.

5.2.1 Traceability

Gotel et al. [Gotel 1994] have given basic definition of the requirements traceability as follows:

The requirements traceability is the ability to describe and follow the life of a requirement, in both a forward and backward direction, i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases.

Requirements traceability has provided twofold value. First of all, using requirements traceability, changes in the context of an application can easily be analyzed for their impact on the code and test cases, and vice versa, which heavily shortens the time required for software maintenance. On the other hand, increased accountability simplifies the verification of a system to its requirements and allows better monitoring of the process. Requirement traceability is also used as to advocate desirable property of the software development processes [Lindvall 1996]. Lots of problems are identified during a system development process. Traceability is used as an optional activity during system development due to limited available resources related to the traceability [Blaauboer 2007].

A project team always determines the way in which project development process should be performed at the initial phase of the project development. Various kinds of decisions realized for acceptance or rejection about a project plan are made by a project board, where as all other technical details are determined by others [Blaauboer 2007]. In this chapter, we have introduced a new technique to use for traceability, which helps to find bugs at the early stage of the system development through visual animations of a formal specification. All these approaches, methods, techniques and tools proposed for the requirements traceability are useful as long as its adoption decision is present preferably at the early stages of a project, and we need to understand how a decision on requirements traceability is made and which factors influence an adoption of the traceability. In the following, we present the conceptual treatment of these questions, which eventually provide us with a theoretical lens to examine this adoption in a systematic manner.

The traceability needs of different stakeholders according to the different kinds of goals. The requirements traceability presents a connection between requirements and related artifacts, which are created during the system development using requirements. A set of tools [Galvao 2007; Aizenbud-Reshef 2006; guo 2009; Hull 2005] is used for requirement traceability for the different purpose during the software life-cycle. However, we have used real-time animator based on the formal model to trace the hidden requirements of a complex system. In requirements engineering and elicitation phase it is important that the rationales and sources to the requirements are captured in order to understand requirements evolution and verification. During design phase, the requirements traceability allows to keep track of what happens when change request is implemented before a system is redesigned. Traceability can also give information about the justifications, important decisions and assumptions behind requirements [Ramesh 2001]. Most important advantage of the requirements traceability is to support validation of the system functionality according to the stakeholder requirements.

5.3 Related Work

Prototype refers to an incomplete version of the system development, which simulates only few aspects of the final system when requirements are indefinite and system behavior is unclear [Davis 1992]. The prototype is only used to be clarified and validated requirements. The experiences gain from prototypes are helping to produce a quality system and the requirements specification document. The prototypes work very well for only small parts of the complex problems. Various kinds of traditional techniques are used to build a rapid throwaway prototype; these include functional and logic programming languages, simulation techniques, object-oriented languages, and visual programming languages.

A prototype of executable formal specifications is used to bridge the gap between a traditional software prototyping and formal methods. An executable formal specification is considered as an abstract program which enables abstract requirements, designs formulation, explored and validated at an early stage of the system development [Fuchs 1992]. Such kinds of prototyping techniques help to discover behavior of the system interacting with its environment that can be observed before it exists in the actual system. Validation of a system assists to design formal documentation using the specification descriptions of system. In few cases, an executable specification forms only relevant document for all phases of the system development, such as in the use of executable specifications with transformational approaches [Berzins 1993].

Goguen and Meseguer [Goguen 1982] have proposed a novel approach for constructing a prototype using formal specification. They advocated the use of an algebraic specification language named OBJ, which can be executed by interpreting the equations of the OBJ specification as a left-to-right term rewriting system. Since several, attempts have been made to execute formal notations for rapid prototyping. Siddiqi et. all [Siddiqi 1997] have divided these attempts into three categories. First category belongs to the use of functional and logical programming languages for the construction of the prototypes [Turner 1985; Henderson 1986a; Kowalski 1985; Mashkoor 2009]. Second category is distinguished by "specially designed and specific purpose" executable specification languages that are usually embedded in an existing programming language which provides the execution mechanism [Henderson 1986b]. Last category can be characterized as the development of an environment for the automatic prototyping of specifications. Siddiqi et. all [Siddiqi 1997] have proposed distinct approach for supporting environment, which combines the benefits of a formal system specification and its subsequent execution via rapid prototype model.

Animation is a simulation technique which is used to execute a model and shows the animation in the visual form using a formal model of a given specification. Lots of works have been done over the last few decades in this area while the idea was originally proposed by Balzer et all [Balzer 1982]. The contributions mainly differ by (a) how far the model is from the underlying requirements, (b) how far the visualization is from phenomena within the software environment, (c) how the simulation works (through direct execution of the model or preliminary translation to some executable form), and (d) how interactive and controlled the simulation can be [Van 2004]. Bloomfield et al. [Bloomfield 1986] had presented a case study in which a simple prototype for nuclear reactor protection was modelled in VDM [Bjørner 1978], where as a part of the safety assessment process was presented

using prolog [Clocksin 1987] animation. Another interesting study of the industrial use of animation in the analysis of a formal model of information flow in dynamic virtual organisations (VOs) is presented by John et al. [Fitzgerald 2008]. VDM tool is used for developing the formal model of virtual organisation (VO) structure. This development also supports interaction with the model without requiring exposure to the formalism. All kinds of simulation tools for Requirements Engineering (RE) with compressive comparison study have been presented by Schmid et. all [Schmid 2000].

According to our literature survey, none of the existing approaches discuss to construct a formal specification based prototype, which can use real-time data set to test the validation of the formal specifications for developing any critical systems like avionic and medical systems. Most of the existing tools are used *toy-data* set for validating the formal specifications. Limitations of existing tools are that they cannot support real-time environment to capture the data set for testing. We have proposed an architecture for real-time animation using formal specification [Méry 2010b], which can be used for real-time animation for any existing animation tool and formal language. We have given the prototype model of this framework for Event-B specification. This tool is very helpful to show the real-time system behavior from a formal specification and meets stakeholder's requirements. Moreover, it bridges the gap between different kinds of domain experts. It can help a formal model designer to gain confidence that the formal model that is being designed, refined in an incremental way and finally implemented, does meet the domain requirements. The real-time animation tool is very helpful for evidence-based validation as well as in the certification process. This technique uniformly establishes what to check and how to check it and gives certain evidence of correctness.

5.4 Animation

Animation is a well-established technique that is used to check for compilation of the actual requirements of stakeholders with the given software specifications. It shows an actual behavior of the system through execution of a formal model in the form of animation. An executable model is based on the software specifications; the software behavior is simulated by executing that model; the simulation is visualized on a textual or graphical model representation by highlighting the current model element being executed. Animation thus allows a software engineer to discover the presence of problems, not their absence. Several kinds of benefits and limitations of animation [Schmid 2000; Hayes 1989] are given as follows:

5.4.1 Benefits of Animation

- Animation has major benefits of validating a system model through earlier detection and correcting the problems for improving the quality of requirements specification.

- Animation provides behavior of a system model, which can be used to validate the internal mechanism of the system model by inspection, and it helps to clarify requirements using animated interaction with the specification when requirements are

unclear.

- In this prototyping technique, all kinds of tools have their own automatic translation tools, which is used for making formal specifications executable.
- Execution of the formal model in the form of animation helps inspection and formal reasoning as a means of validation for better understanding of the given system. Quite often the stakeholders are not sure about what they exactly want or how to describe their ideas. This technique helps to them to discover real requirements.

5.4.2 Limitations of Animation

- Animation techniques are not for exhaustive testing. In a complex system, there are a large number of states, which is impossible to test due to the problem of "states explosion".
- Animation tools are not always stakeholder-friendly due to specific notations of the supported modeling languages, which are not easily understandable by non-technical stakeholders and might be difficult to read and interpret such animations.
- As long as the requirements engineering process is in progress, the requirements engineers have to handle ambiguous and incomplete requirements. Nevertheless, in that case requirements engineers are obliged to define a semantically correct and formal system model in order to run animation.
- Animation always focuses on the behavioral aspects of a system. However, non-functional requirements, such as reliability, cannot be animated from requirements. The mean life function, such as the mean time to failure (MTTF), is widely used as the measurement of a product's reliability and performance. This value is often calculated by dividing the total operating time of the units tested by the total number of failures encountered. Hence, reliability is usually measured by a rate. For example, the reliability of a system is 95%, then how it is possible to measure that the reliability of that system is 95% through animation.

5.5 Proposed Architecture

Fig. 5.1 depicts a functional architecture that can use the real-time data set to animate a formal model without generating a source code in any target language (C, C++, VHDL, etc.). This architecture has six components: Data acquisition and preprocessing unit; Feature extraction unit; Database; Graphical animations tools; Interfacing plug-in; and formal specification model. All these six components can use any particular tool for building a prototype for realizing the concepts of a real-time animator.

We have used some existing tools to build a prototype model of this proposed architecture. Fig. 5.2 presents prototype implementations in order to understand the different development phases of the real-time animator. This architecture is applicable to building an animation tool for any formal modeling languages like VDM, Z and TLA$^+$ etcetera.

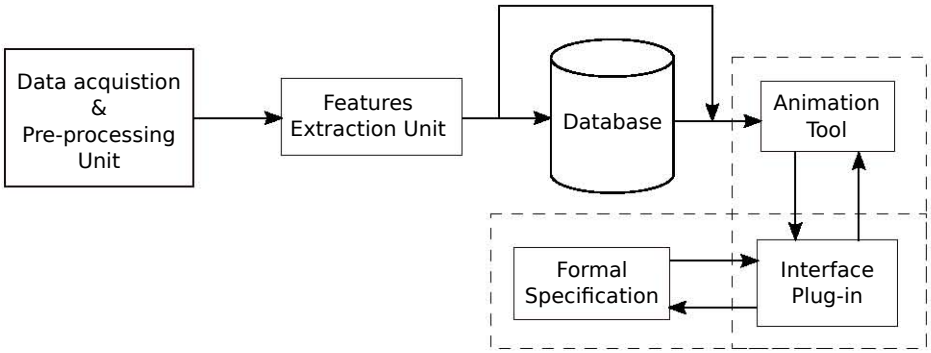


Figure 5.1: A functional architecture to animate a formal specification using real time data set without generating source code

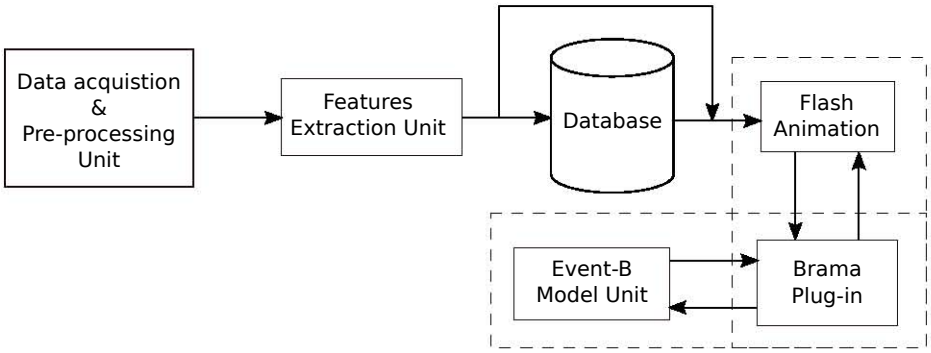


Figure 5.2: A prototype model of the functional architecture to animate a formal specification

Here, we present equivalent architecture of a real-time animator in the context of Event-B formal modeling language. The prototype architecture has six components: Data acquisition and preprocessing unit; Feature extraction unit; Database; Graphical animations dedicated tool: Macromedia Flash; a Formal model animation tool Brama plug-in to interface between Flash animation and Event-B model; and formal specification system Event-B.

5.5.1 Data acquisition & Preprocessing

Data acquisition and preprocessing begin with the physical phenomenon or physical property to be measured. Examples of this include temperature, light intensity, heart activities and blood pressure [Dorticós 2006] and so on. Data acquisition is the process of sampling of real-world physical conditions and conversion of the resulting samples into digital numeric values. The data-acquisition hardware can vary from environment to environment (i.e camera, sensor, etc.). The components of data-acquisition systems include sensors that convert physical properties. A sensor, which is a type of transducer, that measures a physical quantity and converts it into a signal which can be read by an observer or by an instrument.

Data preprocessing is a next step to perform on raw data to prepare it for another pro-

cessing procedure. Data preprocessing transforms the data into a format that will be more easily and effectively processed for the purpose of the user. There are a number of different tools and methods used for preprocessing on different types of raw data, including: sampling, which selects a representative subset from a large population of data; transformation, which manipulates raw data to produce a single input; denoising, which removes noise from data; normalization, which organizes data for more efficient access.

5.5.2 Feature extraction

The features extraction unit is a set of algorithms that is used to extract the parameters or features from the collected data set. A set of algorithms is implemented in any particular language (Matlab, C, C++ etc.). All these algorithms are different for each system. For example, during prototype implementation of this architecture, we have used a set of algorithms for extracting the ECG features or parameters. These parameters or features are numerical values that are used by animated model at the time of animation. The feature extraction relies on a thorough understanding of the entire system mechanics, the failure mechanisms, and their manifestation in the signatures. The accuracy of the system is fully dependent on the feature or parameter values being used. Feature extraction involves simplifying the amount of resources required to describe a large set of data accurately. When performing analysis of a complex data one of the major problems stems from the number of variables involved. Analysis with a large number of variables generally requires a large amount of memory and computation power or a classification algorithm which overfits the training sample and generalizes poorly to new samples. Feature extraction is a general term for methods of constructing combinations of the variables to get around these problems while still describing the data with sufficient accuracy. Collecting measured data and processing these data to accurately determine model parameter values is an essential task for the complete characterization of a formal model.

5.5.3 Database

The database unit is optional. It stores the feature or parameter values in a database file in any specific format. This database file of parameters or features can be used in future to execute the model. Sometimes, feature extraction algorithms take more time to calculate the parameters or the features. In such a situation, modeler can store the parameters or the features in a database file to test the model in the future. A modeler can also use the extracted parameters or features directly in the formal model, without using the database.

5.5.4 Graphical animations tool: Macromedia Flash

The animated graphics are designed in the Macromedia Flash tool [Reinhardt 2007]. Macromedia Flash, a popular authoring software developed by Macromedia, is used to create vector graphics-based animation programs with high graphic illustrations and simple interactivity. Here, we use this tool to create an animated model of the physical environment and to use the Brama plug-in to connect the Flash animation and the Event-B model. This tool also helps to connect the real-time data set to a formal model specification using some

intermediate steps and finally makes the animated model closer to the domain expert expectations.

5.5.5 Animator: Brama plug-in

Brama [Servat 2006] is an animator for Event-B specification, which is designed by Clearsy. Brama is an Eclipse plug-in suit and Macromedia Flash extension that can be used with Windows, Linux and MacOS for Rodin platform [RODIN 2004]. Brama can be used to create animations at different stages of development of a simulated system. To do so, a modeler may need to create an animation using the Macromedia Flash plug-in for Brama. The use of this plug-in is established through a communication between the animation and the simulation.

Brama is a tool allowing to animate Event-B models on the Rodin platform. It allows animating and inspecting a model using Flash animations. Brama has two objectives: to allow the formal model's designer to ensure that his model is executed in accordance with the system it is supposed to represent; to provide this model with a graphic representation and animate this representation in accordance with the state of the formal model. A modeler can represent the system manually within Rodin [RODIN 2004] or represent the system with the Macromedia Flash tool that allows for communication with the Brama animation engine through a communication server. The graphic representation must be in Macromedia Flash format and requires the use of a separate tool for its elaboration (Flash MX, for example). Once the Event-B model is satisfactory (it has been fully proven, and its animation has demonstrated that the model behaves like its related system), you can create a graphic representation of this system and animate it synchronously with the underlying Event-B Rodin model. Brama does not create this animation. It is up to the modeler to create the representation of the model depending on the part of the model he wants to display. However, Brama provides the elements required to connect your Flash animation and Event-B model [Servat 2006].

5.5.6 Formal modeling Language: Event-B

Event-B is a *proof-based* formal methods [Cansell 2007; Abrial 2010] for system-level modeling and analysis of large reactive and distributed systems. In order to model a system, Event-B represents in terms of *contexts* and *machines*. The set theory and first-order logic are used to define contexts and machines of a given system. Contexts [Cansell 2007; Abrial 2010] contain the static parts of a model. Each context may consist of carrier sets and constants as well as axioms, which are used to describe the properties of those sets and constants. Machines [Cansell 2007; Abrial 2010] contain the dynamic parts of an Event-B model. This part is used to provide the behavioral properties of a model. A machine model is a state, which is defined by means of variables, invariants, events and theorems. The use of refinement represents systems at different levels of abstraction and the use of mathematical proof verifies consistency between refinement levels. Event-B is provided with tool support in the form of an open and extensible Eclipse-based IDE called Rodin [RODIN 2004] which is a platform for Event-B specification and verification.

5.6 Applications and Case Studies

We have applied our proposed approach of real-time animator [Méry 2010b] in the development of the cardiac pacemaker [Méry 2009] (see Part-II). The pacemaker specification [Scientific 2007; Goldman 1974; NITRD 2009] has been proposed by the software quality research laboratory at McMaster University as a pilot project for the Verified Software Initiative [Woodcock 2006; Macedo 2008]. The challenge is characterized by system aspects, including hardware requirements and safety issues. A cardiac pacemaker is a high confidence medical device [NITRD 2009; Hoare 2009; Woodcock 2007] that is implemented to provide proper heart rhythm when the body's natural pacemaker does not function properly. In the cardiac pacemaker, the electrodes are attached into the right atrium or the right ventricle, or in both chambers. Cardiac pacemaker has several operational modes that regulate the heart functioning. The specification document [Scientific 2007] describes all possible operating modes that are controlled by the different programmable parameters of the pacemaker. All the programmable parameters are related to real-time and action-reaction constraints, that are used to regulate the heart rate.

In the development of the cardiac pacemaker, real-time animator of formal specification helps to animate the formal model with *real-time data* set instead of *toy-data*, and offers a simple way for specifiers to build a domain-specific visualization that can be used by domain experts to check whether a formal specification corresponds to their expectations. The pacemaker models must be validated to make sure that they meet requirements of the pacemaker. Hence, validation must be carried out by both formal modeling and domain experts.

5.7 Limitations

The proposed architecture of the real-time animator is sufficient to validate refinement-based formal specifications using real-time data set instead of *toy-data* set. The major limitation of this tool is real-time data collection and features extraction for testing the validation of formal specifications of a system, where feature extraction algorithms are not able to calculate required features under real-time (i.e. ECG features extraction). Due to limitation of this algorithm, we proposed off-line validation technique using database. Database is used to store the extracted features in a specific file format for validating the formal models.

Another limitation we have discovered through our experiments that every refinement step is not animatable, specially early development of the formal model (or abstract model), where concrete functional behaviors of the system are not specified yet. Incremental refinement approach builds the system gradually and provides the concrete system behavior, which may be animatable. Refinement based modeling helps for obtaining stepwise validation for modeling a system [Mashkoor 2009]. This is consistent with using animation as a kind of quality-assurance activity during development. Early stages of system models are too abstract and not able to present parametric based desired behavior of the system. So that, this real-time animation is useful to validate later refinement stages of a system

or concrete models, when some parametric behaviors are introduced in the system. These parameters are used as features in the formal specifications for real-time validation. We believe that one animation per abstraction level is sufficient. In fact, the first refinement of a level may often have a non-determinism too wide to allow for meaningful animation (concept introduction), but subsequent refinements get the definitions of the new concept precise enough to allow animation.

5.8 Conclusion

The objective of this proposed architecture is to validate the formal model with real-time data set in the early stage of development without generating the source code in any target language. Here, we focused the attention on the techniques introduced in the architecture for using the real-time data set to achieve the adaptability and confidence on a formal model. Moreover, this architecture may provide validation for the formal model with respect to the high level specifications according to the domain experts (i.e medical experts). At last, this proposed architecture should be adaptable to various target platforms and formal models techniques (Event-B, Z, Alloy, TLA$^+$ etc.).

Our approach has involved for designing and using of the real-time animator for executing a formal specification to validate actual requirements. The main objectives of our work are to promote the use of such kind of real-time animator [Méry 2010b] to bridge the gap between software engineers and stakeholders to build a quality system, and to discover all ambiguous informations from the requirements. Moreover, this tool helps to verify the correctness of the behavior of the system according to the stakeholders requirements. The formal verification and evidence based testing using an animation offer to obtain that challenge of complying with FDA's QSR, ISO/IEC and IEEE standards quality system directives [Keatley 1999] and help to get certification for highly complex critical systems.

A key feature of this validation as it is full automation and animation of specification in the early stage of formal development. The case study (see Part-II) has shown that requirements specifications could be used directly in the real-time environment without modifications for automatic test result evaluation using our approach. Moreover, there are scientific and legal applications as well, where the formal model based animation can be used to simulate (or emulate) certain scenarios to glean more information or better understandings of the system and assist to improve the final given system. Main contributions of proposing this real-time animator tool are,

- to reduce the gap between software engineers and stakeholders requirements using real-time animator and easy to explain model behavior to the domain experts as well stakeholders;
- a real-time animation of a specification supplements inspection and reasoning as means for validation. This is especially important for the validation of non-functional behaviour;
- a real-time animation technique is available in early phase of the system development life-cycle, which can be used to correct validation errors immediately, without

incurring costly redevelopment;

- ambiguous and incomplete requirements can be clarified and completed by hands-on experience with the specifications using our approach;
- the goal-oriented animation for evidence based expection to verify particular portions of a behavior model;
- animation can be helpful for incremental model building and analysis of a complex system;
- helps to domain experts to analyse work process guidelines;
- animator assists to regulatory agencies and helps to meet ISO/IEC and IEEE standards;
- ability to monitor a real-time environment using animator at animation time and analyse the requirements, violations of goals, expectations on the environments, and domain properties;

This work has been influenced by guiding principles and technical benefits of the formal system engineering, which offers participation of both software engineers and user stakeholders that helps to move closer towards a quality requirements specification and to develop an error-free system.

CHAPTER 6

Refinement Chart

"Simplicity is prerequisite for reliability."
(Edsger W. Dijkstra)

Refinement techniques [Abrial 2010; Abrial 1996a; Back 1979] serve as a key role for modeling a complex system in an incremental way. This chapter presents another required tool namely refinement chart for handling the complexity of a system. Refinement chart is a graphical representation of a complex system using layering approach, where functional blocks are divided into multiple simpler blocks in a new refinement level, without changing the original behavior of the system. The main objective of this refinement chart is to model the whole system using graphical notations and to obtain a concrete specification. The refinement chart offers a clear view of assistance in "system" integration. This approach also gives a clear view about the system assembling based on operating modes and different kinds of features. To show the effectiveness of this approach, we have used for graphical modeling to simplifying the complexity of the system in the development of our selected case studies.

6.1 Introduction

High-confidence medical devices (ICD, pacemaker, infusion pump, etc.), automotive and avionic systems are too much error prone in operating due to the complexity of the systems [Butler 1996b; Miller 1998; Abrial 1996c]. New methodologies are needed to make critical viable in the future marketplace by simplifying the various design stages. This chapter proposes a refinement-based graphical technique for designing the complex critical systems. The refinement chart provides an easily manageable representation for different refinement subsystems and offers a clear view of assistance in system integration. This methodology simplifies specification, synthesis, and validation of the systems and enables an efficient creation/customization of the critical systems at low-cost and development time.

Despite all the efforts of the community, critical system designers still need a new way for modeling the systems and analyse the complexity of the systems. This is mainly because of a set of requirements that is mandatory for an efficient modeling solution, but is still not provided by a single existing environment:

- Infrastructure for critical system integration and inter-operation.
- Introspection features for easier debugging and analysis of complex specifications.
- Model-based development and component-based design frameworks.
- System integration of critical infrastructure.
- Possibility of annotating models for different purposes, (e.g., directing the synthesis or hooking to verification tools).
- Decomposition of the complex system into different independent subsystems.

The contribution of this chapter is to propose a new graphical notation based refinement chart for a complex critical system design. This technique provides the solutions for all the requirements enumerated above. The refinement chart is proposed in this methodology for designing a critical system like medical device, automotive and avionic systems.

This chapter is organized as follows. Section 2 presents related work. Section 3 depicts a refinement chart and describes basic rules for presenting any system using the refinement chart. Section 4 presents assessment of the refinement chart using applications and case studies and finally, in Section 5 provides some concluding remarks.

6.2 Related Work

A *modal system* is a system characterized by *operation modes*, which coordinates system operations. Many systems are *modal systems*, for instance, space and avionic systems [Butler 1996b; Miller 1998], steam boiler control [Abrial 1996c], transportation and space system and so on. Operation modes explore the actual system behavior through observation of a system functioning under multiple situations. In this approach, a system is considered as a set of operating modes, where each operating mode is categorised according to the system functionality over different operating conditions.

Modecharts [Jahanian 1994] is a graphical technique, which is used to handle mode and mode switching of a system. The authors have given the detailed information about the state space partition, various working conditions of the system and to define the control information in the large state machines. However, modecharts lack adequate support to specifying and reasoning about functional properties. Some papers [Real 2004; Fohler 1992] have also addressed the problem of mode changing in a real-time system. Dotti et al. [Dotti 2009] have proposed both formalization and a refinement notion for a *modal system*, using existing support for the construction of *modal system*.

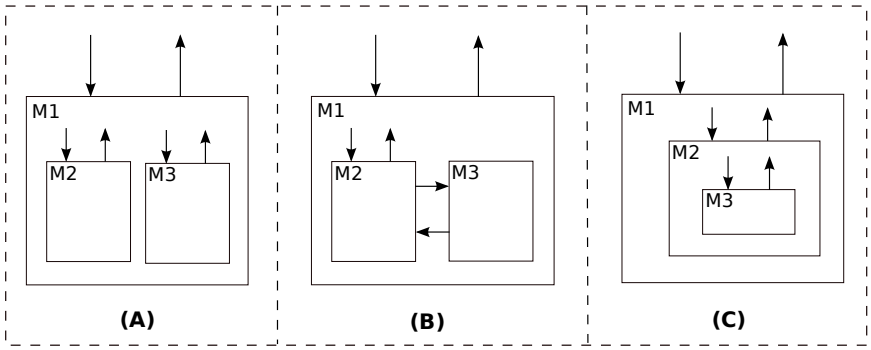
According to our literature survey, none of the existing approaches discuss a refinement-based technique for handling the complexity of a system. We have given a technique of the refinement chart for presenting different operating modes under various subsystems. Each subsystem represents an independent function according to the operating modes. This refinement chart technique helps to design complex system structure and relationship between two subsystems using operating modes, which helps in system integration using code structure of the different subsystems.

6.3 Refinement Chart

The purpose of this refinement chart is to specify modal system requirements in a form that is easily and effectively implementable. During the modeling of modal system, several styles of specification are usually adopted for handling the complex operating modes. Functional blocks are divided into multiple simpler blocks in a new refinement level, without changing the original behavior of a system. The final goal is to obtain a specification that is detailed enough to be effectively implemented, but also to correctly describe the requirements of a system.

The development of embedded software for the critical system requires significant lower level manual interaction for organizing and assembling a whole system. This is inherently error-prone, time-consuming and platform-dependent. To detect the failure cases in a software is not an easy task. Manually reviewing the source code is the only way to trace the cause of a failure. Due to the technological advancement and modern complexity of the critical system software, this is an impossible task for any third party investigator without prior knowledge of the software. Consequently, we have proposed the synthesis of a system using incremental refinements, to synchronize and integrate the different subsystems of a system. This approach also helps in code integration and to test the different subsystems of a system independently.

As the nature of critical systems is often characterizable as *modal systems*, we follow a state-based approach to propose suitable abstractions. We consider that the state of a model is detailed enough to allow one to distinguish its different operating conditions and also to characterize required mode functionality and possible mode switching in terms of state transitions.



$(A)\ M1 \sqsubseteq (M2 \parallel M3)$
$(B)\ M1 \sqsubseteq (M2 \succ M3)$
$(C)\ (M1 \sqsubseteq M2,\ M2 \sqsubseteq M3)$

Figure 6.1: Refinement charts

Each subsystem that forms the specification is represented into a block diagram as a refinement chart. Fig. 6.1 presents the diagrams of the most abstract modal system. The

diagrams use a visual notation loosely based on Statechart [Harel 1987b]. A mode is represented by a box with a mode name; a mode transition is an arrow connecting two modes. The direction of an arrow indicates the previous and next modes in a transition. Refinement is expressed by nesting boxes. A refined diagram with an outgoing arrow from an abstract mode is equivalent to outgoing arrows from each of the concrete modes. It is also similar to ingoing arrow. In a refinement, nesting box can be arranged hierarchically and can be represented by basic rules of our refinement chart (see Fig. 6.2). Basic rules of refinements are a parallel refinement [$M1 \sqsubseteq (M2 \parallel M3 \parallel ..... \parallel M_{n-1} \parallel M_n)$], sequential refinement [$M1 \sqsubseteq (M2 \succ M3 \succ ..... \succ M_{n-1} \succ M_n)$] and nested refinement [$(M1 \sqsubseteq M2,\ M2 \sqsubseteq M3, ... ,\ M_{n-1} \sqsubseteq M_n)$]. Furthermore, refinement charts, which appears in the hierarchical form can be represented in the sequential or parallel or nesting, or in all sequential, parallel and nesting ways. A complete system can be represented by using mixing of all refinement chart notations, means each subsystem can be refined by any rule that is given in Fig. 6.2.

Fig. 6.1 presents for only three modes (M1, M2 and M3) with different kinds of refinements. The parallel relationship among several refinement boxes states that a system operates simultaneously in all the subsystems. For instance, Fig. 6.1(A) represents the abstract mode $M1$ and two parallel refinements are represented by nesting mode boxes $M2$ and $M3$. Transition between these two refinements $M2$ and $M3$ are not allowed. Entry into a parallel refined subsystem requires entry into all of its immediate child refinement. A transition out of one refinement requires an exit out of all the refined subsystems in parallel to it. The sequential relationship among several refinement boxes states that the system operates in at most one of these subsystems at any time. For example, Fig. 6.1(B) represents an abstract mode $M1$ and two sequential refinements are presented by the nesting mode boxes $M2$ and $M3$ in two levels of hierarchy, where $M2$ and $M3$ are embedded in $M1$. The transitions between $M2$ and $M3$ allows the system to go from one refinement to another refinement according to the operating modes. The nesting relationship among several refinement boxes states that the system operates in any subsystems. For example, Fig. 6.1(C) represents an abstract mode $M1$ and the subsystems refinement by a nesting box $M2$ and the subsystem $M2$ is refined by a nesting box $M3$ in three levels of hierarchy, where $M2$ is embedded in $M1$ and $M3$ is embedded in $M2$. A transition is allowed to next level of refined subsystem. A transition out of one refinement requires an exit out of all the refined sub level of refined subsystems.

$$M1 \sqsubseteq (M2 \parallel M3 \parallel ..... \parallel M_{n-1} \parallel M_n)$$
$$M1 \sqsubseteq (M2 \succ M3 \succ ..... \succ M_{n-1} \succ M_n)$$
$$(M1 \sqsubseteq M2,\ M2 \sqsubseteq M3, ... ,\ M_{n-1} \sqsubseteq M_n)$$

Figure 6.2: Basic rules of refinement chart

As an example, Fig. 6.3 presents the diagrams of the most abstract modal system for the one electrode pacemaker system (A) and the resulting models of three successive refinement steps (B to D). The diagrams use a visual notation to represent the bradycardia

operating modes of the pacemaker under functional and parametric requirements. An operating mode is represented by a box with a mode name; an operating mode transition is an arrow connecting two operating modes. The direction of an arrow indicates the previous and next operating modes in a transition. Refinement is expressed by nesting boxes. A detailed description about these refinement blocks related to the one-electrode cardiac pacemaker is given in case studies.

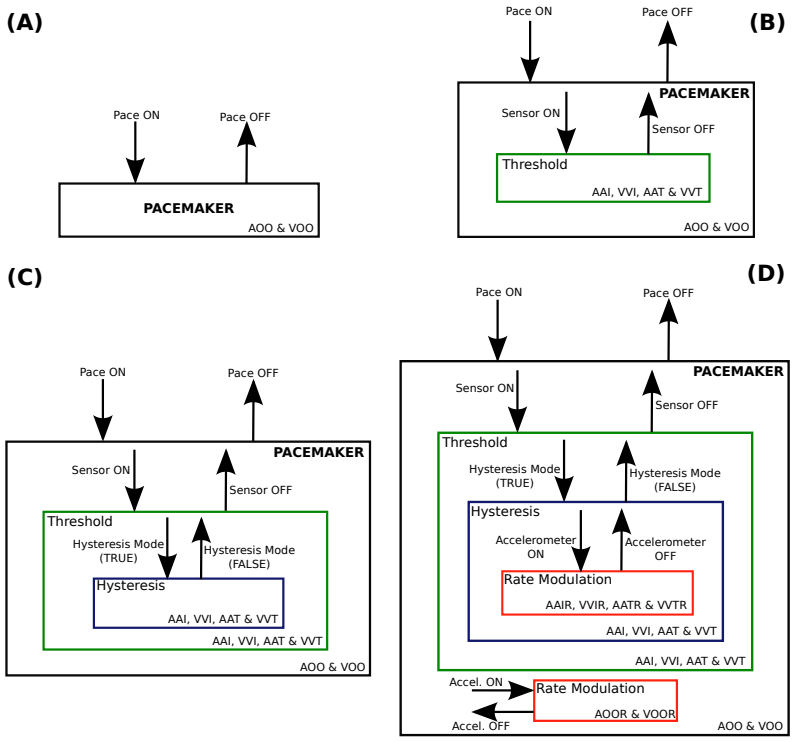


Figure 6.3: Refinements of one-electrode pacemaker using the refinement chart

Refinement based representation is used during the decomposition and synthesis phases of a system. The purpose of the refinement chart is to provide an easily manageable representation for different refinements of a system. The refinement chart offers a clear view of assistance in system integration. This is an important issue not only for being able to derive system-level performance and correctness guarantees, but also for being able to assemble components in a cost-effective manner.

Refinement is a modeling technique that is used to introduce more concrete behavior of a system in the next level of refinement, where it preserves the safety properties and system behavior between two refinement levels. This preservation property allows us to model the whole system from initial specification to a concrete level in a form of executable specification using incremental development. The concrete model is considered as that it is preserving system behavior, thereby establishing that the generated code satisfies the initial specification [Walters 1990]. Proof of consistency between source and target of a refinement is an intrinsic capability of a refinement process, which can be composed in a similar fashion [Smith 2008].

A refinement-based system development has a different cost structure than the traditional development life cycle. The cost of building models and related other required design knowledge may be higher for producing the first system. However, these costs are amortized when reuse these models and designs for developing the other future systems. Thus, the cost of producing first program may be higher, but the cost of development for reproducing advanced version of the products and reuse same codes in other products should be less than conventional programming [Smith 2008; Walters 1990]. The cost of handling of proof obligations of specifications and refinements should be less than the cost of analyzing the final product. It can also help to a code designer to improve the code structures, code optimization, and code generation techniques. Every incremental refinement represents additional functionalities. This refinement-based structure may greatly improve the safety, hardware integration and guidelines to develop the critical systems. We, therefore, propose a simple methodology of system integration using the refinement chart, that seeks to minimize the effort and overhead.

6.4 Applications and Case Studies

We have applied the refinement chart [Méry 2011e] in the system development for handling the complexity of different type of case studies (see Part-II). Mainly, we have used for developing a cardiac pacemaker model and adaptive cruise control (ACC) model. Refinement chart helps to model the system integration, which also complies with refinement based formal development. The block diagrams of the refinement chart help to build the complete system and used to handle the complexity of the whole system through decomposing in multiple independent parts. Here, refinement chart models different kinds of operating modes, and decompose the whole system based on operational modes. Decomposing using the refinement chart helps to analyse individual component and interaction or switching from one operating mode to other operating modes. Formal verification and validation are carried out by both formal modeling experts and domain experts (medical experts and control engineers), while refinement chart based system integration approach and system development are carried out by industrial people.

6.5 Conclusion

Today, in order to respect the certifiable assurance and safety, time to market and strict cost constraints, critical system designers need some new modeling and simulation solutions. The solutions must also permit software component modeling, component integration in a distributed environment, easier debugging of complex specifications, and mitigated connection with other, existing or new systems [Méry 2010d].

In this chapter, we would like to stress the original contribution of our work. At each refinement step, some functional blocks are divided into simpler blocks, without changing the behavior of a system is represented by the refinement chart. We have proposed a technique to synthesis, integrate, and synchronize the subsystems of a system using incremental refinements. This approach helps in code integration and to test the different subsystems

independently. The purpose of the refinement chart is to provide an easily manageable representation for different refinement subsystems. The refinement chart offers a clear view of assistance in the system integration. This is an important issue not only for being able to derive system-level performance and correctness guarantees, but also for being able to assemble components in a cost-effective manner. Moreover, the refinement chart represents a block diagram for each subsystems and provides a structure in various refinements to build the complete system. Concrete refinement charts provide system integration information in the form of compose and decompose of software codes according to the blocks diagrams. Composition and decomposition help to improve the code structure and code optimization. To find a minimum set of events for each independent subsystem is known as code optimization, and synthesizing and synchronizing of a set of events are known as code structuring. The refinement chart specially covers component-based design frameworks and decomposition, integration of critical infrastructure and device integration. The complexity of design is reduced by structuring systems using modes and by detailing this design using refinement.

CHAPTER 7

EB2ALL : An Automatic Code Generator Tool

"Beware of bugs in the above code;I have only proved it correct, not tried it."

(Donald Knuth)

The most important step in the software-development life cycle is the code implementation. This chapter presents a design architecture of an automatic code generation tool, which can generate code into several programming languages (C, C++, Java and C#) [Méry 2010a; EB2ALL 2011; Méry 2011c; Méry 2011b]. This tool is a collection of plug-ins, which are used for translating the Event-B formal specifications into different kinds of programming languages. The translation tool is rigorously developed with safety properties preservation. This is an essential tool, which supports code implementation phase of our proposed development life-cycle methodology for developing the critical systems.

7.1 Introduction

Formal methods provide a sound mathematical basis for system requirements descriptions and aim to produce zero-defect software, by controlling the whole software-development process, from specification to implementation. The capability of formal and automated verification of safety properties in formal models, before transformation into code, has added real value to industrial systems, including hardware systems and software systems. Several constraining requirements are existing particularly in the embedded domain due to limited size of memory for translating from formal specifications to a given target programming language (C [Kernighan 1988; Pearce 2007], C++ [Stroustrup 1994], Java [Arnold 2005; Gosling 2005] and C# [Richter 2006]). To overcome such kinds of problems, first, a compromise must be found between the expressiveness of the formal implementation language and the simplicity of the translation process. Another compromise is also necessary between formal models, which generally favor the readability and the simplicity of the verification process, over the code efficiency.

The code generation process consists in several stages: formal implementations are translated into programs in a given programming language using a *tool chain* of a translator, and then these programs are compiled. This approach offers several advantages: the

translation process is as simple as possible, and it can be validated in an easy way; secondly having a formal specification of a system suggests as a next step to use it during the testing phase. Software testing tries to check the correctness of a system with respect to its specification in program states that are chosen for the test. The simplicity of the translation ensures the traceability between formal specification and executed code.

This chapter describes a tool translating Event-B specification into any given target programming language. The structure of Event-B and the nature of tool has been developed to support for direct-translation from Event-B formal specification into any target programming language. We provide a rigorous translation tool EB2ALL [EB2ALL 2011; Méry 2011b] for Event-B specification to target programming language that can easily be adapted in any domain and gives freedom for developers to adjust at best their integer representation for overcoming memory-related problems.

The EB2ALL code generator supports automatic generation of C, C++, Java and C# code from Event-B [Abrial 2010] formal specifications. The tool EB2ALL is a collection of plugins, which are named as EB2C, EB2C++, EB2J and EB2C# [EB2ALL 2011; Méry 2011b]. All these tools are used as plug-in features for the Rodin development tool [RODIN 2004]. Rodin development tool is an open and extensible Eclipse-based IDE, which is a platform for Event-B specification and verification.

We present a multi-phased translation process from Event-B [Abrial 2010] models. An Event-B model supports *set-theoretical* notations that are impossible to directly translate into any target programming language. The translator automatically rewrites partially formal notations of Event-B [Abrial 2010], that can be easily translatable for the final target programming languages. Any target programming language source code is then automatically generated from the model via using an appropriate translation phase of the tool. The final translated code is applicable to compile into an executable code using the conventional compilation tools.

A developer can also use translated code to extend the functionality of a system by inserting extra code or some new functionalities that are not included in the Event-B formal development. Some parts of the implementation code are not supported by the code generator, and a user wants to implement some existing components more efficiently are main reasons for inserting extra code into the automatically generated code. Moreover, it offers a flexible way for Event-B designer to generate C, C++, Java and C# codes. Due to manual intervention in the generated source code, we propose a code verification technique using the meta-proof and software model checking tools like BLAST [Beyer 2007] for verifying desired behaviors of the developed system. This tool is freely available for download[1]. The use of this tool is exemplified through the generation of C, C++, Java and C# codes from specification of a cardiac pacemaker (see Part-II).

A code translation from Event-B was relatively easy, but its subsequent use presented more problems. The most important challenging task in the code generation is the code verification. The reason is that the preservation at the code level of the properties proved at the architectural level is guaranteed only if - the underlying platform is correct and - correctness of the final system when filling in the stubs for internal actions into the automatic

[1] Download: http://eb2all.loria.fr/

generated code. Another important challenge is to support all formal notations of Event-B. Few formal notations are used at the abstract level for a system development, those symbols are not directly translatable. We have also faced a specific challenge related to the indeterministic behavior of a system. Most of the formal specifications are indeterministic, which are not safe for an automatic code generation. To make a formal specification deterministic before code generation and to verify that the system behaviors are correct according to the developer, and also comply with indeterministic system specifications, are challenging tasks in this code generation process. Invariants are used for defining type definition and safety properties of a system. How to use invariants corresponding to the safety properties for verifying generated codes is also one kind of challenge.

This chapter is organized as follows. Section 2 presents related work and Section 3 depicts an architecture of the translator in a form of a *tool chain* and describes various parts of the translation process. Section 4 presents use of code generator plugins. Section 5 discusses limitations of the tool and finally, in Section 6 provides some concluding remarks.

7.2 Related Work

Automatic code generation is a standard technique in the area of Software Engineering. Several tools are developed by research community for generating source code from graphical modeling tool like UML [Rumbaugh 1999; Smith 2001; Georg 2001] to any target programming language like C++ or Java. But automatic source code generation from formal specification to a high-level programming language is supported by few formal techniques like Classic B [Abrial 2010] and Vienna Development Method (VDM) [Bjørner 1978; Overture]. The VDM [Bjørner 1978; Overture] is a set of techniques and tools based on formal specification language - the VDM Specification Language (VDM-SL). Extended version of VDM tool (VDM++) supports modeling of object-oriented and concurrent systems. VDM tools attract both industrial as well as academic peoples in the area of formal based development. VDM tools provide features for analyzing models, testing and proving properties of models, and generating program codes from validated VDM models.

A tool vMAGIC [Pohl 2009] is based on Java library that is used for automatic code generation for VHDL. According to the paper, this tool is very usable and reliable, but a lot of useful features are not implemented yet. This tool is continued under development for adding new features that will be able to do semantic operations as well. In the area of model-driven software engineering, a tool PADL2Java [Bonta 2009] has been developed that translates PADL models into Java code. PADL is a process algebraic architectural description language equipped with a rigorous semantics and transformation rules into multi-threaded object-oriented software, which is employed in the verification tool TwoTowers [Bernardo 1998]. This tool provides the code generation approach and code synthesizing techniques.

SPARK [Barnes 2003] is a formally-defined programming language based on a restricted subset of the Ada language [Ada 1981], intended to be secure and to support the development of high integrity software related to the critical systems. It describes desired behavior of the system components and to verify the expected runtime requirements. Main

features of this language are to support strong static typing, static and run-time checking, object-oriented programming, exception handling, parallel tasks, etc. Static verification tools allow to check the absence of general run-time errors like numerical overflow or division by zero and that the user-specified properties hold. The proofs will either be generated automatically or developed with the programmer's assistance for the more complex cases.

From Classic-B [Abrial 1996a] notation to 'C', C++ and ADA language translation tool has been developed by D. Bert, et al. [Bert 2003]. This paper presents a methodology for translating a formal specification based-on Classic-B modeling language. Before generating the 'C' code, the specification model must be restated into an intermediate language 'B0'. The intermediate language 'B0' is restricted set of Classic-B formal notations. This tool is particularly designed for generating a 'C' code for an embedded system. This tool is not able to handle any complex expressions in the specification, so, this tool has very limited use. Event-B [Abrial 2010] modeling language is based on Classic-B modeling language. All the formal notations in Event-B is borrowed from Classic-B formal notations. In the area of code generation from Event-B model to 'C' code is proposed by Stephen Wright [Wright 2009]. But this tool is particularly designed for MIDAS [Wright 2009] project. This tool also supports a subset of Event-B formal notations with a very simple expression form. This tool is no more usable for any Event-B formal specification. Edmunds et al. [Edmunds 2010; Edmunds 2011] have presented a way for generating code, for concurrent programs, from Event-B specifications. Authors aim to integrate their code generation approach with existing Event-B methodology and tools.

As for as we know that there is no any mature translation tool existing, which can translate directly from Event-B formal specification into any target language (C, C++, Java and C#). We have developed a tool that supports automatic translation into any target languages (C, C++, Java and C#) from Event-B formal specifications. This tool also supports the only subset of Event-B [Abrial 2010] formal notations, but it is much richer than previously developed tools [Bert 2003; Wright 2009]. This chapter discusses the code generation approach underlying EB2C, EB2C++, EB2J and EB2C# tools [EB2ALL 2011; Méry 2011b]. This is our first step toward in the direction of code generation from Event-B formal specification, and we are continued working on the translation tool, which can support all kinds of formal notations of Event-B.

7.3 A Basic Framework of Translator

A translator tool is developed as a set of plugins for the Rodin Tool [RODIN 2004], which can generate the source codes from a formal specification into many programming languages (C , C++, Java and C#). The translation tool is named as EB2ALL, which is a group of four kinds of different plugins, called EB2C, EB2C++, EB2J and EB2C#. All these tools have common architecture and a set of protocols for generating a source code. The translation process consists in transforming the concrete part of an Event-B project into a semantically equivalent text written in any target programming language. This section proposes an architecture for the Event-B translator; different parts of the translator have been shown in Fig. 7.1 and this architecture supports translation for several target

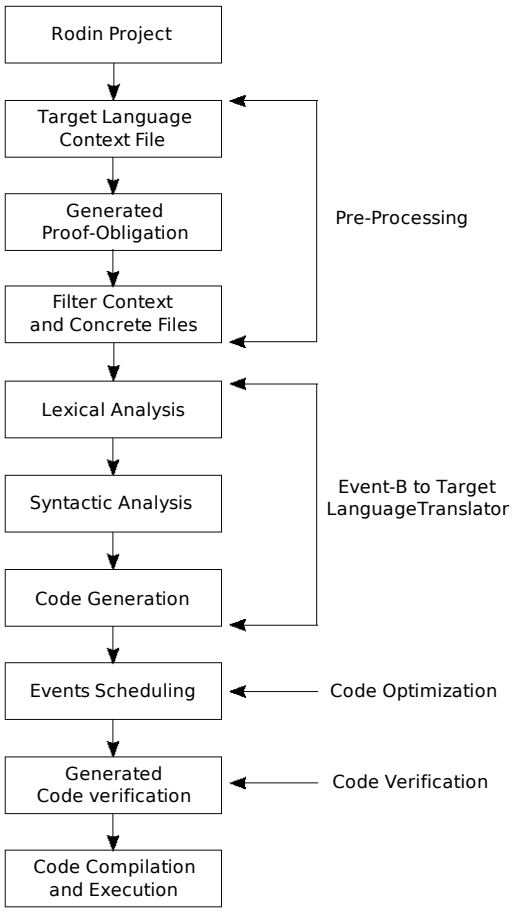


Figure 7.1: A General Architecture of a Translation tool

programming languages like C, C++, Java and C#. The translator tool is customized for each new target programming language to generate an efficient code, which can support various kinds of execution platforms. The proposed architecture is able to generate a verified code, which also comply with the behavior of formal specifications. The translation tool is implemented in the Eclipse framework as a set of plug-ins for Event-B. Event-B translation process as a chain of tool description is given as follows:

7.3.1 Selection of a Rodin Project

Formal development of a system in Event-B modeling language is provided by an open and extensible Eclipse-based IDE called Rodin [RODIN 2004], which supports system specifications and verification. A visual interface and a set of plugins help to specify and to prove a system under logico theory. A proof manager is integrated in Rodin tool. Event-B models and all proof-related informations of a system project are always stored in the Rodin database. The translator tool is implemented as a set of plugins under the Eclipse development framework, using the recommended interfaces to traverse the statically checked internal database, thus decoupling the tool from the syntax of the Event-B notation by

accessing its underlying meaning. The syntax of the mathematical notation, that is, expressions, predicates, and assignments, are maintained in the form of an abstract syntax tree. This is the first phase of the translation process, which presents an explicit selection of a particular Rodin [RODIN 2004] project from all the loaded projects into the workspace. A particular selected project is passed to the next phase of the translation process for generating the source code.

7.3.2 Introduction of a Context File

7.3.2.1 Motivation

Failure of a critical system can be a cause of loss of life, financial loss, environmental damage and injury, where the cost of failure is not tolerable or affordable. For a critical system *run-time* error may be just as hazardous as any other logical error. For instance, overflow and underflow of bounded integer are a type of a run-time error that should be addressed in the proof process. For developing a critical system, an automated technique based on formal methods have been matured to provide a high degree of confidence. Introduction of a context file is a very important phase of the code generation process, which provides one more level of refinement to make a system specification deterministic. The deterministic model is known as concrete well formed specification, which shows correct system behavior before generating a source code in a deterministic way. New context file consists of all *primary data types* corresponding to the programming languages (C, C++, Java and C#).

The papers [Sites 1974; Coleman 1979; Cousot 2007] address a solution for proof of clean termination that provides the facts that a program is totally correct. Clean termination means that a program terminates normally without any execution-time semantic errors like integer overflow, use of undefined variables, subscript out of range, etcetera [Sites 1974; Cousot 2007]. We propose the Pre-Processing stages for obtaining a deterministic model using one more level of refinement introduction through a new context, which are similar to the clean termination approach [Sites 1974; Coleman 1979]. This refinement phase provides the deterministic definitions of constants and variables using one more level of refinement. This new refinement generates a lot of proof obligations. Types of generated proof obligations and proof details are similar to the papers [Sites 1974; Coleman 1979; Cousot 2007]. This level of refinement complies system specification abstractly. The generated proof obligations are discharged by automatic as well as manual, and all proofs that are necessary to verify the specification in order to guarantee the consistency and correctness of the system. Therefore, this phase is very important to maintain all safety properties into automatic code generation from the Event-B model.

7.3.2.2 Selection of a Context File

This phase provides many context files, to add into a current project according to a target programming language choice (C, C++, Java and C#). Table 7.1 shows bounded integer data types for all target programming languages. In the Event-B language, there are two kinds of constants and variables (*data*): abstract data and concrete data. Abstract data

consists in all the elements and sets that can be used in set theory, as defined in Event-B (integers, enumerated sets, cartesian products, power sets, relations, and so on). They are mainly used at the higher levels of specification (machines and in top level refinements) [Abrial 2010].

Concrete data are those which may be used in the final translation process, each time the data thus introduced will not be further refined in the development. It is the case for constants and variables at the implementation level but also for parameters and results of operations, which cannot be refined in Event-B [Abrial 2010]. Concrete data must match ordinary data types in a usual programming language because they should be "implemented" directly in the target programming language. So, the correspondence between concrete data and data types must be obvious. In standard Event-B, they are the following ones:

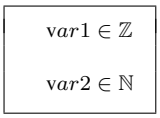
- enumerated types (including the boolean type)
- bounded integer type (from MININT to MAXINT)
- arrays on finite index intervals where the type of elements is a concrete type (in set theory, they are similar for total functions)

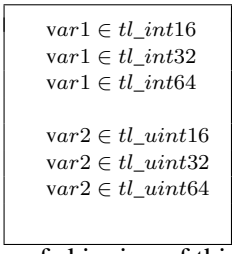
Event-B type	Formal Range	C & C++ type	Java type	C# type
tl_int16	$-2^{15}..2^{15}-1$	int	short	short
tl_uint16	$0..2^{16}-1$	unsigned int	-	ushort
tl_int32	$-2^{31}..2^{31}-1$	long int	int	int
tl_uint32	$0..2^{32}-1$	unsigned long int	-	uint
tl_int64	$-2^{63}..2^{63}-1$	-	long	long
tl_uint64	$0..2^{64}-1$	-	-	ulong

Table 7.1: Integer bounded data type declaration in different context files

7.3.2.3 Refinement using a new Context File

A new context file consists of a new data type definition equivalent to the primary data type (see Table 7.1). This new context file helps to model a system more deterministic through converting indeterministic data type into a deterministic data type. Introduction of new deterministic data types are defined through a new refinement. In this new refinement process, a developer can replace all Event-B data types of all the context and last concrete machine models, corresponding to the limitations of the selected target programming language data types. The following figure presents an example for converting indeterministic data type into a deterministic data type.





A developer can skip this refinement level. In case of skipping of this refinement level in the translation process, the translator generates default maximum bounded integer primitive data type for all the variables and constants. New redefined deterministic ranges using refinement generates a lot of proof obligations. New refined model is passed to the next phase of the translation process.

7.3.3 Generated Proof Obligations

Introduction of a new context file is used as a data refinement of the system through removing all non-deterministic range of data types into a selected target programming language. The refinement is supported by the Rodin [RODIN 2004; Back 1979] platform guarantees the preservation of safety properties. Thus, the behavior of the final system is preserved by an abstract model as well as in the correctly refined models. A lot of proof obligations are generated, which can prove automatically as well as through manual intervention using interactive proof procedures [Abrial 2010; Back 1979]. A model developer can discharge all generated proof obligations with the help of the Rodin proof tool [RODIN 2004]. For example, when indeterministic constants and variables change into to the deterministic data type ranges in new refinement level, then some proof obligations will be generated due to restrictive set of data ranges according to the predefined system behaviors. All these new generated proof obligations are required to discharge before continue the process. There is no guarantee to preserve all safety properties (related to overflow or underflow) of the proved system unless all proof obligations are discharged. A proved Rodin project can pass to the next phase of the translation process.

Refinement using a context file provides proof of absence of run time errors into the generated code. Such kind of approach is also known as formal code verification [Burns 2004]. Formal code verification techniques are used to demonstrate that at every point in the code where a run time error may occur like numeric overflow or underflow. A set of required conditions as in the form of predicates guarantees that the run time error will not occur. One more level of refinement approach is a very valuable step in the translation process to save from run time error, which demonstrates that especially in systems where the occurrence of an undesired run time error is unrecoverable.

7.3.4 Filter Context and Concrete machine Modules

Refinement is a key feature of a formal development, which supports incremental development for specifying a complex system. Event-B modeling language supports refinement based incremental development of a system, where a formal model is a series of development starting with a very abstract model of the system under development. Details are

gradually added to this first model by building a sequence of more concrete events and ending with the implementation of machine as a final concrete machine. The relationship between two successive models in this sequence is *refinement* [Abrial 2010; Back 1979]. Relations between modules are only relations *sees* and *refines*.

This stage of the automatic translation process is used to filter all context and concrete machine files from the selected project. Context files consist of static properties as *sets*, *constants*, *functions* and *enumerated sets* of a system, while machine files contain dynamic properties as *variables*, *functions* and *events* of a system. A set of filtered context and concrete files contain concrete *sets*, concrete *constants*, concrete *variables*, concrete *functions* and concrete *events*, which are used for implementation of a system. A selected project always contains some concrete models, which are refinements of the abstract models. The translation tool automatically filters a set of context and concrete files. If the translation tool is not able to filter all context and concrete files from the selected project, then immediately the translation process is terminated with an error message, else all filtered context and concrete files are passed to the next translation phase for continuing translation process.

7.3.5 Basic Principles of Code Generation

This phase is the heart of the translation tool. Before this level, all phases are used as preprocessing steps for obtaining the run-time errors free codes through data refinement of the formal specification. Main objective of this phase is to translate statically checked and proved formal specifications of Event-B model into *observationally equivalent* standard programming languages (C, C++, Java and C#). A set of rules has been generated for producing a source code, which is applicable to all context and concrete machine modules (i.e. those found to have no further refinement).

The translation tool generates source files as equivalent to the number of concrete machine files [Abrial 2010]. Source code generated file has a similar name corresponding to the concrete machine file, and a file extension generates according to the choice of a programming language. Source code generated files are saved in the folder of a particular language (C, C++, Java and C#) in the workspace of the selected Rodin project. Source code generated file begins with insertion of header comments containing a timestamp and name of the selected Rodin project. Some required header informations are also inserted in the header of the source file according to the target programming language requirements. For instance, in C++ and C# source files contain all required header files related to the standard template library (STL) and Collection class of .NET Framework [Blanc 2007; Stroustrup 1994], respectively. A Java source file contains *sets* definition of set operations for handling the *sets* based notations of Event-B using standard class of Java utilities.

Main cause of failure of this translation tool is unable to parse any predicate. For example, the current tool is not able to handle relational operator ($\leftrightarrow$) and quantifiers ($\exists$ and $\forall$). If any predicate expression contains any quantifiers or relational operators, then the translation process is unable to translate it into a selected programming language, and the translation process fails. In case of translation failure, the tool immediately proceeds with translation of the next module. For avoiding the errors in the translation process, we have

considered mainly two approaches:

- To model a system specification using a set of symbols (see Table 7.2 and Table 7.3), which is supported by the translation tool.
- Transformation of final concrete models into a supported symbol list (see Table 7.2 and Table 7.3) through the refinement process.

Before generating a source code from a model, a user is required to refine the system using a subset of Event-B symbols (see in Table-(7.2, 7.3 and 7.4) , which can restate the model in a more translatable form. Supported symbols are available in Table-(7.2, 7.3 and 7.4). These tables show a set of Event-B syntax to the equivalent C, C++, Java and C# programming languages. All constants defined in a model's context must be replaced with their literal values. This translation tool supports *Sets* theory notations (not in 'C'), conditional, arithmetical and logical expressions of a formal specification. These formal notations and expressions are translated into equivalent programming language code. A detailed translation process of the context and concrete machine files are given in the following sections.

7.3.5.1 Process Context and Machine Files using Lexical and Syntax Analysis

Context and machine files consist of static and dynamic properties of a system in the form of modular architecture, which represents a system behavior using formal notations. To generate a source code into any programming language, it is required to process the context and machine files separately using lexical and syntactic analysis. Usually, the parsing of a formal model is divided into two stages: lexical analysis and syntactic analysis. In real-world problem like code generation, lexical analysis and syntactic analysis stages may be intertwined with each other [Sebesta 2009]. Rodin is an open source extensible IDE [RODIN 2004], which is developed in the Eclipse. We are very thankful to the Rodin development team for providing source code of Rodin tool. Source code of the Rodin tool is well-written and developed as in the form of a set of plugins to design a complete tool. The Rodin tool has a set of library files of an Abstract Syntax Tree (AST) which is mainly used for lexical and syntactic analysis for Event-B notations at the time of modeling. We have used same library for lexical and syntactic analysis of Event-B model for generating a source code into any programming language (C, C++, Java and C#). To generate a source code into any target programming language, input source (Event-B formal model) is always same. Therefore, we have similar kinds of procedures to process context and machine files using existing AST library of the Rodin tool.

7.3.5.2 Process Context Files

The context of Event-B model consists of *sets*, *enumerated sets*, *constants*, *arrays* and *constant functions*, associated with their respective type. The translation tool supports all kinds of context components to generate a constant type with respect to a programming language. An instance of the context consists in associating to each name a value consistent

Event-B	'C' & 'C++' Language	Comment
n..m	int	Interger type
x ∈ Y	Y x;	Scaler declaration
x ∈ tl_int16	int x;	'C' & 'C++' Contexts
x ∈ n..m → Y	Y x [m+1];	Array declaration
x :∈ Y	/* No Action */	Indeterminate Init.
x : | Y	/* No Action */	Indeterminate Init.
x = y	if(x==y) {	Conditional
x ≠ y	if(x!=y) {	Conditional
x < y	if(x<y) {	Conditional
x ≤ y	if(x<=y) {	Conditional
x > y	if(x>y) {	Conditional
x ≥ y	if(x>=y) {	Conditional
(x>y) ∧ (x≥z)	if ((x>y) && (x>=z) {	Conditional
(x>y) ∨ (x≥z)	if ((x>y) || (x>=z) {	Conditional
x := y + z	x = y + z;	Arithmetic assignment
x := y - z	x = y - z;	Arithmetic assignment
x := y * z	x = y * z;	Arithmetic assignment
x := y ÷ z	x = y / z;	Arithmetic assignment
x := F(y)	x = F(y);	Function assignment
a := F(x↦y)	a = F(x, y);	Function assignment
x := a(y)	x = a[y];	Array assignment
x := y	x = y;	Scalar action
a := a ⩤ {x↦y}	a[x] = y;	Array action
a := a ⩤ {x↦y} ⩤ {i↦j}	a[x]=y; a[i]=j;	Array action
X⇒Y	if(!X || Y){	Logical Implication
X⇔Y	if((!X || Y) && (!Y || X)){	Logical Equivalence
¬x<y	if(!(x<y)){	Logical not
x ∈ ℕ	unsigned long int x	Natural numbers
x ∈ ℤ	signed long int x	Integer numbers
∀	/* No Action */	Quantifier
∃	/* No Action */	Quantifier
fun ∈ ℕ × ℕ → ℕ	unsigned long int fun(unsigned long int $arg1$, unsigned long int $arg2$) { //TODO: Add your Code return; }	Function Definition

Table 7.2: Event-B to C & C++ translation syntax

Event-B	'Java' & 'C#'	Comment
n..m	short	Interger type
x ∈ Y	Y x;	Scaler declaration
x ∈ tl_int16	short x; (Java) & ushort x; (C#)	'Java' & 'C#' Contexts
x ∈ n..m → Y	Y [ ]x = new Y[m+1];	Array declaration
x :∈ Y	/* No Action */	Indeterminate Init.
x : | Y	/* No Action */	Indeterminate Init.
x = y	if(x==y) {	Conditional
x ≠ y	if(x!=y) {	Conditional
x < y	if(x<y) {	Conditional
x ≤ y	if(x<=y) {	Conditional
x > y	if(x>y) {	Conditional
x ≥ y	if(x>=y) {	Conditional
(x>y) ∧ (x≥z)	if ((x>y) && (x>=z) {	Conditional
(x>y) ∨ (x≥z)	if ((x>y) || (x>=z) {	Conditional
x := y + z	x = y + z;	Arithmetic assignment
x := y - z	x = y - z;	Arithmetic assignment
x := y * z	x = y * z;	Arithmetic assignment
x := y ÷ z	x = y / z;	Arithmetic assignment
x := F(y)	x = F(y);	Function assignment
a := F(x↦y)	a = F(x, y);	Function assignment
x := a(y)	x = a[y];	Array assignment
x := y	x = y;	Scalar action
a := a ⩤ {x↦y}	a[x] = y;	Array action
a := a ⩤ {x↦y} ⩤ {i↦j}	a[x]=y; a[i]=j;	Array action
X⇒Y	if(!X || Y){	Logical Implication
X⇔Y	if((!X || Y) && (!Y || X)){	Logical Equivalence
¬x<y	if(!(x<y)){	Logical not
x ∈ ℕ	unsigned long int x	Natural numbers
x ∈ ℤ	signed long int x	Integer numbers
∀	/* No Action */	Quantifier
∃	/* No Action */	Quantifier
fun ∈ ℕ × ℕ → ℕ	public long fun(long $arg1$, long $arg2$) { //TODO: Add your Code return; }	Function Definition

Table 7.3: Event-B to Java & C# translation syntax

Event-B	**'C++'**	**Comment**
set_var	set <*data type*> set_var	STL library
$\cup$	set_union(...)	STL library
$\cap$	set_intersection(...)	STL library
$\setminus$	set_difference(...)	STL library
$\subseteq$	includes(...)	STL library
$\subset$	includes(...) && !(equal(...))	STL library
$\not\subseteq$	!(includes(...))	STL library
$\not\subset$	!(includes(...)) && !(equal(...))	STL library
Event-B	**'Java'**	**Comment**
set_var	Set <*data type*> set_var = new HashSet<*data type*>()	Java Utilities
$\cup$	unionSet(...)	Java Utilities
$\cap$	intersectionSet(...)	Java Utilities
$\setminus$	differenceSet(...)	Java Utilities
$\subseteq$	isSubset(...)	Java Utilities
$\subset$	isSubset(...) && !(isEqualSet(...)) {	Java Utilities
$\not\subseteq$	!(isSubset(...))	Java Utilities
$\not\subset$	!(isSubset(...)) && !(isEqualSet(...))	Java Utilities
Event-B	**'C#'**	**Comment**
set_var	HashSet <*data type*> set_var = new HashSet<*data type*>()	.NET Framework 4
$\cup$	UnionWith(...)	.NET Framework 4
$\cap$	IntersectWith(...)	.NET Framework 4
$\setminus$	ExceptWith(...)	.NET Framework 4
$\subseteq$	IsSubsetOf(...)	.NET Framework 4
$\subset$	IsProperSubsetOf(...)	.NET Framework 4
$\not\subseteq$	!(IsSubsetOf(...))	.NET Framework 4
$\not\subset$	!(IsProperSubsetOf(...))	.NET Framework 4

Table 7.4: Event-B to Java & C# Sets translation syntax

with its declared type. The observational equivalence is based on equivalence between Event-B values and target programming language values. This equivalence on values is naturally extended on instances of context. The observational equivalence between Event-B sets and target programming language types is given in Table-7.5.

Event-B types	**Target Language types**
Enumerated sets	Enumerated types
Basic integer sets	Predefined integer types
Event-B array types	Target programming language array type
Function	Target programming language function structure
Sets theory	Set theory implementation using advanced library function in target language (not in 'C')

Table 7.5: Equivalence between Event-B and programming language

7.3.5.3 Mapping Event-B Constant Types to Programming Language

- **Enumerated Sets**

  An Event-B enumerated sets is semantically equivalent to a target programming language enumerated type. It is very easy to translate into a target programming language equivalent form due to equivalent semantical structure.

Event-B
$partition(ESet, \{On\}, \{Off\})$

C and C++
$enum\ ESet\{On, Off\};$

Java and C#
$public\ enum\ ESet\{On, Off\};$

- **The Numeric Types**

  The links between the Event-B and target programming languages for integer values have been considered as crucial for the efficiency of a generated code and for the correctness of the translation. So, the solution is provided in the second phase by introducing target programming language context, and it is able to interface very tightly between Event-B types and a target programming language type. The Event-B numeric types ($\mathbb{N}$, $\mathbb{N}_1$ and $\mathbb{Z}$) are either all mapped to the pre-defined context files (see 2nd phase of the translation process) or defined the maximum integer range according to a programming language. For translating constant data type in C, C++ and C# programming languages use $const$ keyword, while Java uses $static\ final$ keyword for defining a constant.

Event-B
$Lnum \in \mathbb{N}$

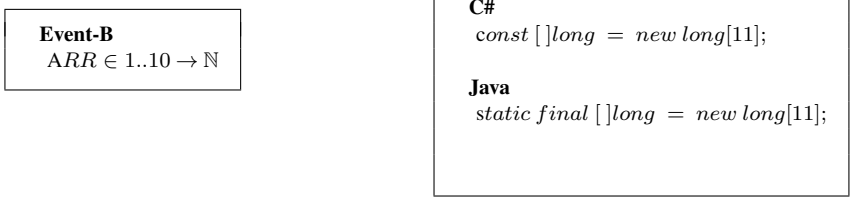
C, and C++
$const\ long\ int\ Lnum;$

C#
$const\ long\ Lnum;$

Java
$static\ final\ long\ Lnum;$

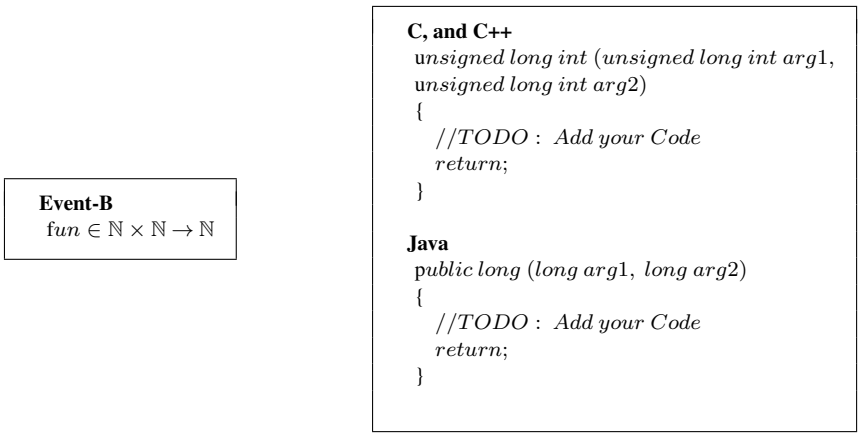
- **The Array Type**

  The links between Event-B arrays and target programming language arrays are not straightforward. In Event-B, an array corresponds to a total function whereas in the target programming language, an array corresponds to a contiguous zone of memory (coded as the beginning address of the array and its size). However, it is easy to do a semantical correspondence between an array element *arr(i)* in the Event-B and the value at the location *arr[i]* in target programming languages (see Table-(7.2,7.3)).



- **The Function Type**

  The link between Event-B function and target programming language function is also very ambiguous. The Event-B functions are generated explicitly into a target language code, and function definitions are placed in a corresponding source file. The translation tool only supports total function of Event-B into equivalent corresponding target programming language function. However, it is an easy way to do a semantical correspondence between function passing parameters in a target programming language is equivalent to the elements of left side of the total functions symbol ($\rightarrow$) and output of a target programming language function corresponds to the right-hand side of the total functions symbol ($\rightarrow$) in the Event-B. So, this step of function translation generates a function structure into a target programming language (see Table-(7.2,7.3)).



- **The Set Type**

  The translation tool is a set of plugins, in which C++, Java and C# languages plugins can support $Sets$ formal notation for translation. The Event-B sets type is translated

into a programming language using the standard template library (STL) in C++, advanced Java class utilities in Java and Generic Collection of .NET Framework in C#. We have developed some functions with the help of existing library functions in C++, Java and C#, which are equivalent to the Event-B sets operations (see Table-(7.2, 7.3 and 7.4). We have given the following snap shot of sets operations, which are by default declared in every source code of Java file.

```
public static <T> Set<T> unionSet(Set<T> setA, Set<T> setB)
{
Set<T> tmp = new HashSet<T>(setA);
tmp.addAll(setB);
return tmp;
}

public static <T> Set<T> intersectionSet(Set<T> setA, Set<T> setB)
{
Set<T> tmp = new HashSet<T>(setA);
tmp.retainAll(setB);
return tmp;
}

public static <T> Set<T> differenceSet(Set<T> setA, Set<T> setB)
{
Set<T> tmp = new HashSet<T>(setA);
tmp.removeAll(setB);
return tmp;
}

public static <T> boolean isSubset(Set<T> setA, Set<T> setB)
{
return setB.containsAll(setA);
}

public static <T> boolean isEqualSet(Set<T> setA, Set<T> setB)
{
return (setB.containsAll(setA)&& setA.containsAll(setB));
}
...
```

- **Set Definition:** To define a set type into C++, Java and C#, the translation tool uses some fixed code structures to define a $sets$ type with the help of STL library, Java utilities and .NET Framework (when set type is unknown). The following structures are excerpted from a translated code to understand the set definition of a context model:

  In the C++ language...

```
class DATASet{
private:
        string element;
public:
    DATASet() {element="";}
    DATASet(string elem) {element = elem; }
```

```
    string outElement() const { return element; }
    bool operator<(DATASet temp) const {
     return (element<temp.outElement());
     }
    bool operator==(DATASet temp) const {
     return (element==temp.outElement());
     }
};
set <DATASet> DATA; /* Sets definition */
```

In the Java language...

```
public static class DATASet{
    private String element;
    public DATASet() {element="";}
    public DATASet(String elem) {element = elem; }
    public String outElement() { return element; }
};
Set <DATASet> DATA = new HashSet<DATASet>(); /* Sets definition */
```

In the C# language...

```
public static class DATASet{
    private string element;
    public DATASet() {element="";}
    public DATASet(string elem) {element = elem; }
    public string outElement() { return element; }
};
HashSet <DATASet> DATA = new HashSet<DATASet>(); /* Sets definition */
```

To define a set of numbers are very simple, which can be defined as follows: In C++ language...

```
set <int> A; /* Sets definition */
```

In the Java language...

```
Set <Long> A= new HashSet<Long>(); /* Sets definition */
```

In the Java language...

```
HashSet <int> A= new HashSet<int>(); /* Sets definition */
```

A set of context files is used as an input to declare all kinds of data-types in form of global constants on top of a source code file. All elements of the context files are declared as global. Context element type information is derived from the type-defining $AXIOM$ statement within the context, which may express as integer ranges, specially supported bit-map types or arrays of these defined by mapping functions.

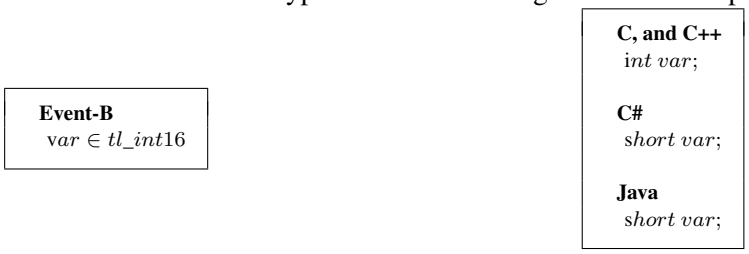
7.3.5.4 Process Machine Files

Machine file contains dynamic behavior of a system, which is denoted by generally *variables*, *arrays*, *functions* and *events*. All these components of a machine file model a system. In the following section, we present automatic transformation of a machine model into any programming language (C, C++, Java and C#).

7.3.5.5 Mapping Event-B Variable Types to Programming Language

A machine file contains *functions*, *arrays* and *variables* for representing a system state. All these elements are declared as global. Global element's type information is derived from the type-defining $INVARIANT$ statements within the machine, which may be expressed as integer ranges, function structure, and arrays. The Event-B to target language code generator generates target language function definitions corresponding to the invariants. The Event-B functions are generated explicitly into the target language code and function definitions are placed in the corresponding source file. The translation tool translates all those into target programming language code according to the same rules, which are defined in the last section for defining a constant type. Static and dynamic type definitions have only difference between context and machine modules data types. For instance, constants and variables have the same definition using primitive data types but C, C++ and C# language constants use *const* keyword and Java uses $static\ final$ keyword with a constant data type declaration, and variables are defined in all languages without using *const* and *static final* keywords.

More than one invariants or axioms may be defined in a single invariant or axiom using and ($\wedge$) logical operator. This translation tool automatically parses an invariant or axioms at the time of constant or variable data type declaration during the translation process.



7.3.5.6 Mapping Event-B Events to Programming Language

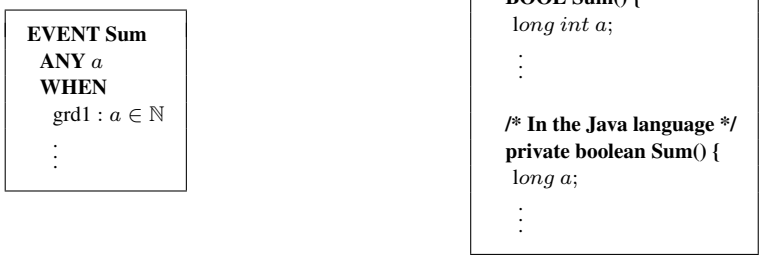
The translation tool provides a recursive process to generate a source code for each event of the Event-B specification into a target programming language. The translation tool always checks for *null* event (i.e. guard of a false condition), never generates a source code for that event, and inserts suitable comment into the source code for the traceability purpose. This automatic reduction is performed to avoid generation of an unreachable *run-time* code.

$$evt\_name \triangleq \textbf{ANY m, n WHERE } grd(v, m, n) \textbf{ THEN } act(v, m, n) \textbf{ END}$$

- **To Process Event's Variable**

  In Event-B specification, there are two kinds of variables: global variables and local variables. Global variables are derived directly from *VARIABLES* statements of a concrete machine, and all these variables have global scope. Local variables are derived from the *ANY* statement of the particular event, and these are entirely local to the corresponding event function. The type of this local variable is declared into the event's guards. Once the guards of an event have been classified, and that conferring local variable type information are used for variable declaration in a function. Remaining guards are used to generate local assignment and conditional statements

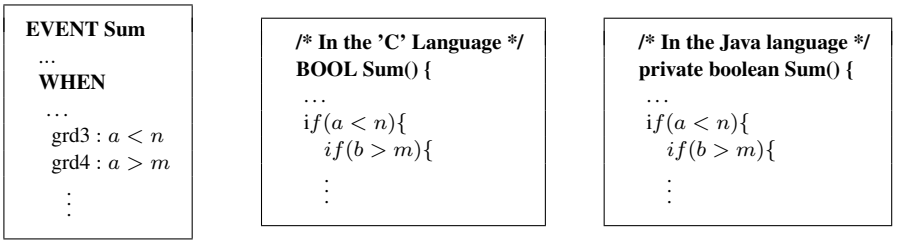
in the guard section. Local variable type information is derived in a similar fashion as the global variables from the guard predicates instead of using INVARIANTS. A recursive process is used to find a type of Event-B local variable corresponding to the programming language. Each event of Event-B is generated in equivalent to the programming language function. After generation of a function header, all local variables, array declarations are inserted at the beginning of the function, giving them scope across the whole function body.



- **To Process Event's Guard**

  In the Event-B, guard handling is very ambiguous due to contain several kinds of modeling notations, such that local variable type definition, conditions and use of different kinds of logical operators ($\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$). Therefore, for handling so many complex situations, we have designed a recursive algorithm for parsing a complex guard and separate different kinds of predicates in the form of formal notations for generating a programming language code using guards. Thus, each guard must be automatically analyzed to resolve this ambiguity from the context information. In a formal model, the guards are known as pre-conditions and action predicates are known as post-conditions. In an event, all pre-conditions must be true for executing the post-conditions or action predicates. For translating an event of a formal model into a programming language, we translate it into corresponding target language function (C, C++, Java and C#).

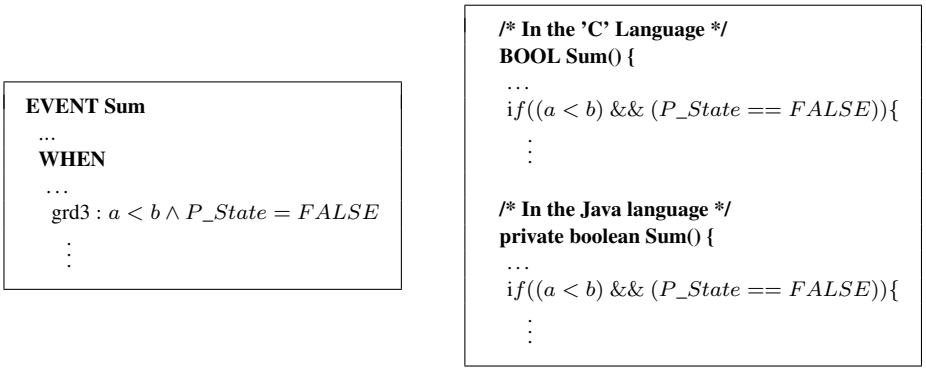
  Pre-conditions of a guard are translated into equivalent if condition statement. A group of guards is translated in the form of *nested-if* conditions, and are placed into an event function as a set of nested conditional statements, using directly translated conditional and local variables declared within nested scope ranges. A suitable comment is also inserted with each guard condition to understand the different elements of the guards like local variables and pre conditions.



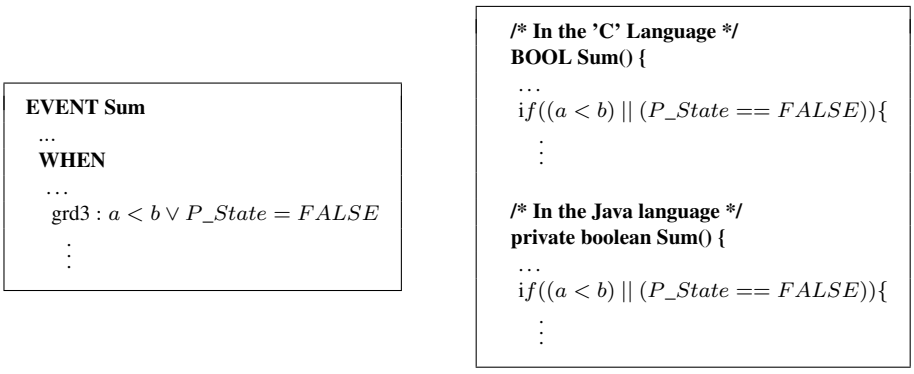
A set of guards can be represented through logical operators, which are simply in the form of propositions calculus. In the account of code generation, we want to explore

the possibility of logical operators in the translation process. An important goal is to handle the various kinds of logical operators in terms of providing a large class of a set of symbols supported for the code generation and to support various kinds of modeling structures. A key part of a structuralist approach is to define the various logical operators as special functions defined on implication structures. It will be helpful in what follows to use Conjunction, Negation, and Disjunction as examples of the way these characterizations work. Universal and existential quantifications are also part of the guards but these are not defined or discussed here due to restricted in the current version of translation tools.

Conjunction ($\wedge$): The conjunction operator ($\wedge$) in a guard predicate is a function of two arguments, such that for any two predicates connected with a conjunction operator ($\wedge$) is translated as follows:



Disjunction ($\vee$): The disjunction operator ($\vee$) in a guard predicate is a function of two arguments. Any two predicates connected with a disjunction operator ($\vee$) is translated as follows:



Negation ($\neg$): The negation operator ($\neg$) in a guard predicate is a function of single argument, such that for any predicate with a negation operator ($\neg$) in a guard is translated as follows:

EVENT Sum
...
WHEN
...
grd3 : $\neg a < b$
$\vdots$

/* In the 'C' Language */
BOOL Sum() {
...
$if(!\,(a < b))\{$
$\vdots$

/* In the Java language */
private boolean Sum() {
...
$if(!\,(a < b))\{$
$\vdots$

Some more logical operators are like implication ($\Rightarrow$) and equivalence ($\Leftrightarrow$), which can be easily rewritten using logical conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$) operators. For example, the implication ($\Rightarrow$) and equivalence ($\Leftrightarrow$) operators, the translator tool automatically rewrite a predicate in an equivalent form using conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$) operators, an equal relation may signify an assignment or equality comparison, and the precise meaning (and hence the resulting translation) deduced from the type and scope of its operands.

Implication ($\Rightarrow$): The implication operator ($\Rightarrow$) in a guard predicate is a function of two arguments, such that for any predicate connected with an implication operator ($\Rightarrow$) is translated as follows:

EVENT Sum
...
WHEN
...
grd3 : $a < b \Rightarrow P\_State = FALSE$
$\vdots$

/* In the 'C' Language */
BOOL Sum() {
...
$if(!\,(a < b)\ ||\ (P\_State == FALSE))\{$
$\vdots$

/* In the Java language */
private boolean Sum() {
...
$if(!\,(a < b)\ ||\ (P\_State == FALSE))\{$
$\vdots$

Equivalence ($\Leftrightarrow$): The equivalence operator ($\Leftrightarrow$) in a guard predicate is a function of two arguments, such that for any predicate connected with an equivalence operator ($\Leftrightarrow$) is translated as follows:

EVENT Sum
...
WHEN
...
grd3 : $a < b \Leftrightarrow P\_State = FALSE$
$\vdots$

/* In the 'C' Language */
BOOL Sum() {
...
$if((!\,(a < b)\ ||\ (P\_State == FALSE))\ ||$
$(!\,(P\_State == FALSE)\ ||\ (a < b)))\{$
$\vdots$

/* In the Java language */
private boolean Sum() {
...
$if((!\,(a < b)\ ||\ (P\_State == FALSE))\ ||$
$(!\,(P\_State == FALSE)\ ||\ (a < b)))\{$
$\vdots$

Arithmetical expressions, calling functions and set operations (see Table-(7.2,7.3)) are also supported by the Event-B formal notations in the guards, which are all translatable into any target programming language. Event-B expressions and statements are code generated, such that the generated code behaves like it is expected from the specification. A set of translation rules with a basic syntactic architecture is defined in Table-7.2,7.3. Translation tool follows the similar set of rules for generating a source code. A special kind of ambiguity of a functional-image relation is resolved during the translation process, which may be used to model a data array or an external function. The meaning of functional-image statements within a model is automatically resolved to an array if the mapping is a global variable, otherwise to call an uninterpreted function. A complete set of guards of an event is translated into equivalent programming language code in the form of pre conditions in an event. During the code translation process, a run-time exception function is generated if an undefined expression or an error statement is occurred. This call of exception function terminates the code translation and reports that an undefined expression into a code generation log file.

The translation tool can support a very complex predicate, where more than one predicate are defined in a single guard. The translation tool is able to automatically parse the whole predicate into a set of predicates, separately for translation purpose.

Set operations: The set operators ($\cup$, $\cap$, $\setminus$) in a guard predicate is a function of two arguments, which uses some intermediate steps according to the C++, Java and C# programming languages. The translation of set based expression is translated as follows:

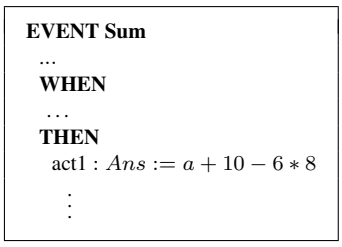
EVENT SetFun
...
WHEN
...
grd3 : $A \cup B \subseteq C$
⋮

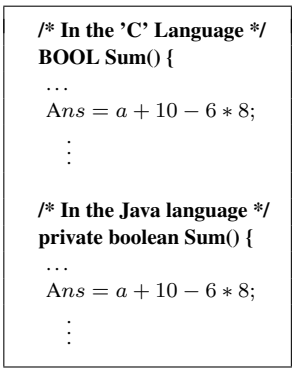
```
/* In the 'C++' Language */
BOOL SetFun() {
 ...
 set < int > tset2;
 set_union(A.begin(), A.end(), B.begin(), B.end(),
 inserter(tset2, tset2.begin()));

 if((includes(C.begin(), C.end(), tset2.begin(), tset2.end()))){
   ⋮

/* In the Java language */
private boolean SetFun() {
 ...
 if(isSubset(unionSet(A, B), C)){
   ⋮

/* In the C# language */
private boolean SetFun() {
 ...
 A.UnionWith(B);
 if(A.IsSubsetOf(C)){
   ⋮
```

- **To Process Event's Action**

  The next sub-stage of an event translation presents the action translation. In the

EVENT-B, all action predicates of an event are considered as in the form of concurrent execution. The set of action predicates are post-conditions in the Event-B events, which state that all action predicates only valid when all pre-conditions or guards are satisfied [Abrial 2010; Abrial 1996a]. Event-B modeling approach supports that any state variable is not allowed to be modified by different action expressions, means Event-B ensures that any state variable used as an action assignee is not modified by any prior post conditions or action predicate. In the code translation process, all action predicates are generated into equivalent programming expression. In a programming language, all action expressions are executed in a sequential order of what they have defined in a formal specification. But all action expressions are executed only when all if conditions become $true$. Event-B supports three kinds of assignment operators *becomes equal to* (:=), *becomes in*(:∈) and *becomes such that* (: |), where *becomes in*(:∈) and *become such that* (: |) are used basically in an abstract model, and through the refinement process, it is represented in the more concrete form as *becomes equal to* (:=). The translation tool only supports *becomes equal to*(:=). If a concrete model uses any *becomes such that* (: |) and *becomes in* (:∈) assignment operators, then the translation tool does not generate action predicates into programming expressions and move to the next action predicate to continue processing. A similar way of parsing is applied on Event-B action statement like a guard statement. The translation tool translates all Event-B actions into an equivalent target programming language source code.





An action translation supports assignments to scalar variables, override statements acting on array-type variables, arithmetic complex expressions and set operations. The Event-B supports a special form of the action predicates, which shows that a state variable can be used in the right side of the assignment operator (:=). To handle such kinds of action predicates, the translation tool automatically modifies the action predicates through a re-write phase and store the value in an intermediate local variable, and finally translate into programming language expression with an assignee in the action expression.

EVENT Sum
...
WHEN
...
THEN
act1 : $OvrVar := OvrVar \lhd\!\!+ \{3 \mapsto 67, 4 \mapsto 88, t \mapsto 56\}$
act1 : $Arr(i) := 5$
⋮

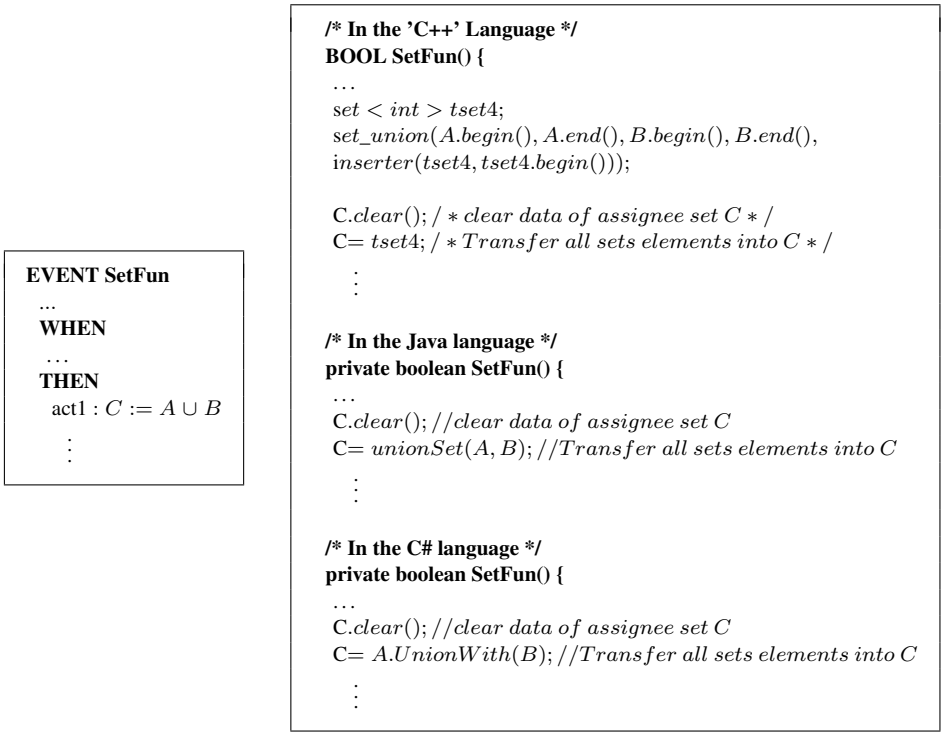
```
/* In the 'C' Language */
BOOL Sum() {
 ...
 OvrVar[3] = 67;
 OvrVar[4] = 88;
 OvrVar[t] = 56;
 Arr[i] = 5;
   ⋮

/* In the Java language */
private boolean Sum() {
 ...
 OvrVar[3] = 67;
 OvrVar[4] = 88;
 OvrVar[t] = 56;
 Arr[i] = 5;
   ⋮
```

After insertion of all kinds of action predicates through the translation process into a generated code event function, adds an extra statement returning a boolean $true$, which express run-time traceability and states that an event function is triggered successfully. After insertion of returning boolean statement, inserts all curly braces (}) according to the total number of guards except those guards, who represents local variable data types. Finally, a returning boolean $false$ statement again inserts before closing the final braces of an event function. The main objective of this $false$ boolean returning statement at a time of execution is that, when this event function executes and any guard of an event function will be $false$, then this function returns $false$ to indicate that this event function is not executed.

```
/* In the 'C' Language */
BOOL EventFun() {
 if(Cond1){
   if(Cond2){
   ...
   Assignment Expr.
   ...
     return TRUE;
   }
 }
     return FALSE;
 }
```

```
/* In the Java language */
private boolean EventFun() {
 if(Cond1){
   if(Cond2){
   ...
   Assignment Expr.
   ...
     return true;
   }
 }
     return false;
 }
```

Set Expression: The set operators ($\cup$, $\cap$, $\setminus$) in the action predicate is also function of two arguments similar to the guards predicates, which use some intermediate steps according to the C++, Java and C# programming languages. The translation of set based expression is translated as follows in the action's part:



The translation tool uses a similar kind of fashion to translate each event of the Event-B model. Event-B is a very rich modeling language for representing formal notation of a specification, but translation tool supports only subset of formal notations of the Event-B. Event-B expressions and statements are code generated, such that the generated code behaves like it is expected from the specification. A set of translation rules and basic syntactic architecture is defined in the Table-(7.2, 7.3 and 7.4). During the code translation process, a run-time exception function is generated if an undefined expression or an error statement is occurred. A generated error invokes an exception function, which terminates the code translation process and reports that an undefined expression into a code generation log file. The next level of the translation tool presents the scheduling techniques to produce an executable code.

7.3.6 Events Scheduling

This phase is not producing any translation part of the translation tool. This section introduces to generate a function for organizing all event functions. There are two ways to organize all the event functions:

1. **Optimize:** An optimized code is used to make a group of calling event functions into a new function. An incremental refinement-based structure of events within an Event-B model provides grouping information about the events. A recursive algorithm is used in the translation tool to discover structuring information from current Rodin project, and could exploit it to recursively generate nesting calling a set of functions corresponding to the abstract events. Merging of common event guards is currently avoided in order to preserve direct mapping between Event-B statements

and translated code, at the cost of possible performance optimizations. However, if translatable guards are already placed in an abstract level, then guards are forming a group of concrete events. An event group is inserted for execution in place of multiple events, improving the run-time performance.

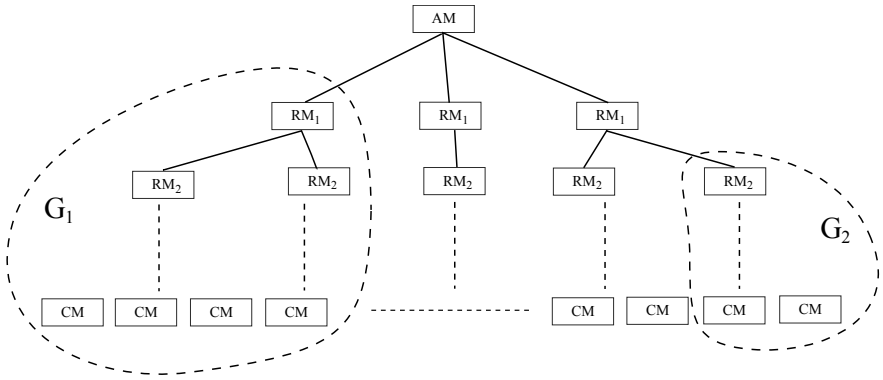


Figure 7.2: Event Scheduling

A Fig. 7.2 shows a basic architecture of event scheduling using optimization approach. This figure represents a tree structure of refinement development, where an abstract model is represented as AM and refinement models are represented through RM_1,RM_2..., and finally concrete models are represented by CM. All refinement models (RM_i) are parts of the single abstract refinement (AM). This figure has two groups G_1 and G_2. G_1 makes a group of a set of events at first refinement level and G_2 makes a group of events at the second refinement level. A user can select the refinement level for making a group of events before code generation. Each group has a set of events, which are only executable when abstract guards are $true$. If a user select higher number of refinement level for optimization then the number of groups may be increased. For instance, in Fig. 7.2 group G_1 has refinement level 1, therefore there are three possible number of groups, while in the group G_2 has refinement level 2 and possible number of maximum groups are five corresponding to the number of blocks in each refinement level.

2. **Sequential:** A sequential organization of the event functions is used to call event functions into a new function, in the same order, defined by their position in the Event-B model. It is not providing any kind of optimization in the calling function.

Event scheduling phase is used to synthesize all target language functions. All these target language functions are equivalent to the Event-B events. We propose two techniques to trigger all translated events. First is calling a function *"Iterate"* that implements a continuous iteration of translated target programming language functions of the Event-B model, in the same order, defined by their position in the Event-B model. Second technique is to optimize the calling order of the events. Optimization approach schedules calling target language functions using a refinement approach. An incremental refinement-based structure of events within an Event-B model exploits to recursively generate nesting calling

functions corresponding to the abstract events. Abstract level guards are forming a group of concrete events. Each group of events are triggered by the main *"Iterate"* function. This technique is used to improve run-time performance wherever at the concrete level has several events. In the sequential order of calling event functions in the *"Iterate"* function invokes every calling function in a sequential order. Whenever guards are satisfied by invoked event function, then the function executes and when guards are not $true$, then the next calling function invokes in a sequential way. Main disadvantage of this technique is that, when a translated code has a lot of calling functions (> 50), then the every function consumes some memory as well as time for invoking a function. While in an optimized way, to make a group of calling functions, which automatically reduces time and memory consumption during invocation of the *"Iterate"* function. A scheduling structure (see Fig. 7.3) shows a calling order of event functions in the *"Iterate"* function.

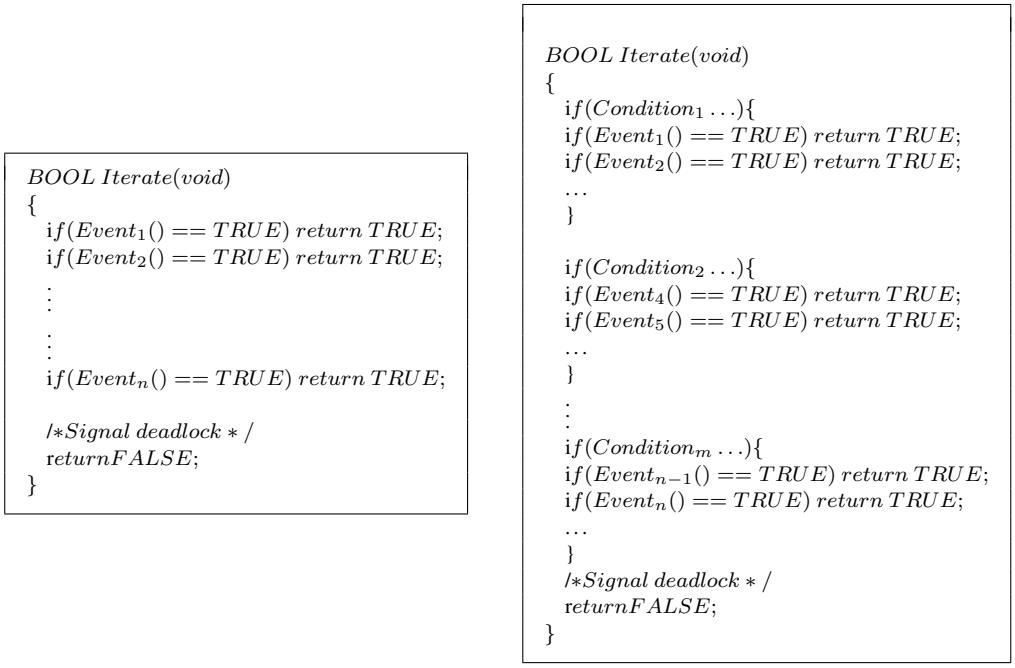


Figure 7.3: Scheduling Architecture

Finally, top-level *main function* of a target programming language is generated to call the generated functions "INITIALISATION" and "Iterate". The only procedural requirement is the calling of "INITIALISATION" prior to "Iterate". All other behavior regarding iteration control may be selected. The INITIALISATION function is exposed to allow later calls to it by the execution environment, providing a mechanism for run-time reset of the Event-B machine if required. In the particular example, the machine is invoked only once and, after initialization, is iterated continuously without any scheduling constraints until either implicit or explicit deadlock (i.e. an event having no actions) is detected. Implicit deadlock is flagged as an error condition, explicit deadlock is treated as normal execution.

In the C and C++ Languages...

```
void main(void)
 {
        if( INITIALISATION()==TRUE ){
        do{
        Iterate();
        }while(!kbhit());
        }
}
```

In the C# language...

```
public static void Main()
 {
M1_VOOR objM1_VOOR = new M1_VOOR();
        if( objM1_VOOR.INITIALISATION()==true ){
        for(int n=0;n<=1000;n++){
        objM1_VOOR.Iterate();
        }
        }
}
```

In the Java language...

```
public static void main(String[] args)
 {
M1_VOOR objM1_VOOR = new M1_VOOR();
        if( objM1_VOOR.INITIALISATION()==true ){
        try{
        for(int n=0;n<=1000;n++){
        objM1_VOOR.Iterate();
        }
        }catch(Exception e){
        e.printStackTrace();
        }
        }
}
```

7.3.7 External Code Injection and Code Verification

This is also an important phase of the code generating process, where the source code files have been generated from Event-B specification. This phase provides a way to introduce some handwritten code or introduction of implicit functions in the form of an interface in order to compile and run the application in target languages (C, C++, Java, C#). For example, when a function is defined abstractly and function returns output value using a set of calculation or algorithms. In that case, a user requires to write a function body, which is generated by the translator. Thus, the user has to write a target language function definition for the operation and add it to the function body of the generated file. To provide some code interface through user intervention, we have introduced code verification step. The code verification is a very important step to verify the correctness of an automatic generated code. This step is required due to manual insertion of an external code. We have considered following two main objectives for adding external codes in the generated codes:

- If some part of the implementation code is not supported by the code generator;
- The user wants to implement some existing components more efficiently.

Due to the complex architecture of the software-development process, it is impossible that any modeling tool can generate directly an executable application. In large s/w, different kinds of languages are used for designing final s/w. So, we have provided a facility of code injection. For instance, addition of hardware specific code (ie. ADA code) into Event-B generated code. For example, in pacemaker case study, Sensor and Actuator are specific hardware units, which control by specific hardware codes. The hardware code is provided by the third-party and pacemaker manufacturing company assumes that this hardware dependent codes are already corrected and verified. So, simple they inject this code directly into the generated code.

Due to injection of the external code and code addition into a complex function body, we have proposed a generated code verification technique, which provides certification of the generated code using this tool. Fig. 7.4 shows an approach to verify a source code of the final developed system. Our idea is to verify the correctness of generated code with respect to Event-B formal specification to give a *meta-proof* that for all Event-B models, and target language translation, the execution of the target language program satisfies the execution (abstractly) of the Event-B model.

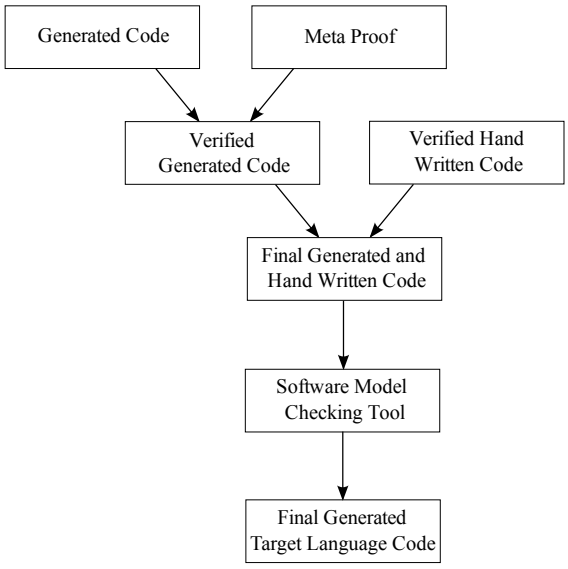


Figure 7.4: Generated Code Verification

In fact, as can be expected, the translation of Event-B descriptions into a target language software cannot be complete, and hence will require an intervention of a software developer in specific positions of the generated code, e.g. for inserting the target language statements corresponding to the internal actions. Hence, next level of code verification injects hand-written code corresponding to the internal actions. Now, we have used software model checking tools according to the different target languages. The software model checking

tools are complemented of the analysis conducted on Event-B specification by making it possible the verification of property preservation at the code level. In fact, although property preservation is guaranteed under certain constraints [Bernardo 2004; Bernardo 2005; Beyer 2007], an inappropriate intervention of software developer on the generated code may lead to the violation of properties proved at an architectural level.

7.3.8 Compiling and Running the Code

Once automatic translation and verification of the Event-B model is completed, all additional requirements are added, an execution environment must be provided and compiled by a suitable target programming language compiler. This final step of the translation tool is to produce generated files for compiling on-target platform (Windows, Linux and Mac).

7.4 How to use Code Generator plugins

To get started using the code generator you should write a Event-B specification. The code generator requires that all files of the Event-B specifications are syntax checked in order to generate correct code. Before generating a source code, it has to ensure that specification is proved and one more level of refinement has been done to make a deterministic model. The code generator tool box will automatically type checked for data types and generates a code according to the programming language.

Fig. 7.5 represents a screen shot of translator tools (EB2C, EB2C++, EB2J and EB2C#)[2] under the Rodin environment [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b]. All these tools are developed as a set of plug-ins under the Eclipse framework. After installation of the EB2C, EB2C++, EB2J and EB2C# plug-ins, menus *Translator/EB2C, /EB2C++, /EB2J, /EB2C#* and tool buttons on the toolbar, will appear. To generate a source code in any target language of any formal model, a user can click on any menu (EB2C, EB2C++, EB2J and EB2C#) or a tool button, then a dialog box will appear (see Fig. 7.5). This dialog box presents a list of active projects. A user can select any project for generating a source code. These tools generate a target language code for all concrete models of the selected project and also generate a log file for the code generating process.

7.4.1 Assessment of the Translation tool

Assessment of this translation tool is given through the code generation of verified formal specifications of the cardiac pacemaker and adaptive cruise control (ACC) (see Part-II). We have illustrated the use of EB2C, EB2C++, EB2J and EB2C# tools [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b] by means of the automatic generation of C, C++, Java and C# codes for the cardiac pacemaker and ACC systems. These codes are automatically generated in C, C++, Java and C# codes from the verified specification in less than five seconds. To find a detail process of code generation from formal specifications in [Méry 2011d].

[2] Download: http://eb2all.loria.fr/

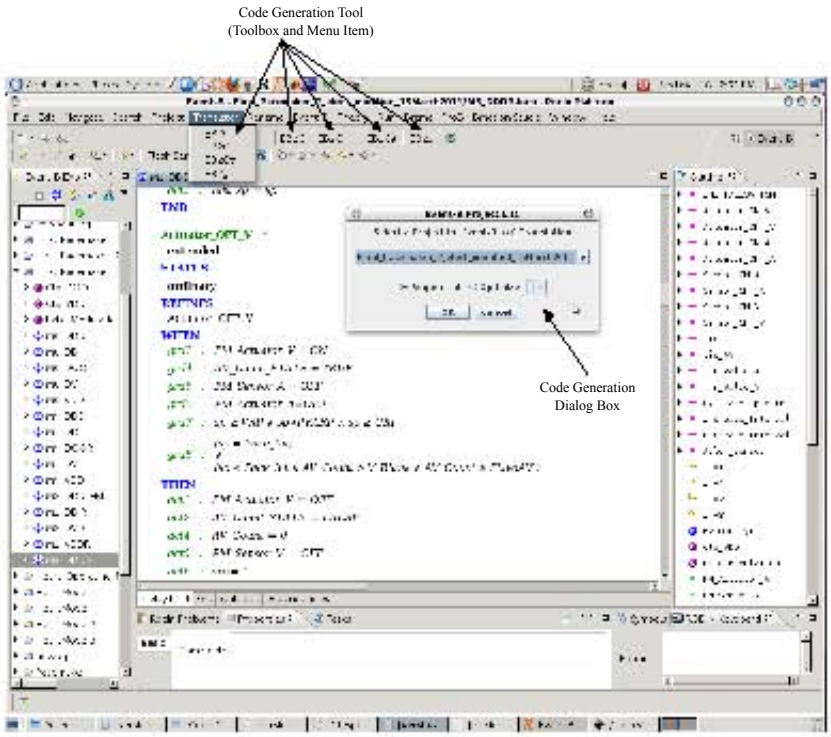


Figure 7.5: Screen shots of Code Generation Tool

7.5 Limitations

This is our primary version of the code generator from Event-B formal specifications to any programming language (C, C++, Java and C#). In this version of the code generator can support only that symbols, which are given in Table-(7.2, 7.3 and 7.4). Event-B language is very rich in the area of modeling, and it supports different kinds of formal notations [Abrial 2010]. All formal notations are not translatable directly into a programming language. A lot of formal notations are applicable at the abstract level of modeling, which may be transformed into another formal notation at the concrete level and that can be easily translatable into any programming language. For instance, relational predicates are usually used for abstract representation, which can be transformed into array or total function representation, and array and total function both are translatable into current version of this tool. Present time the translation tool only supports a subset of formal notations, on behalf of modeling expertise, we believe that these set of Event-B notations are sufficient for modeling any kind of problems and using these symbols any formal specification can be easily translatable into any programming language. In this version of the code generator, the Event-B constructs are not supported except Table-(7.2, 7.3 and 7.4) formal notations. This is our first step toward in the direction of code translation, and it is our ongoing research work. So in future we will provide more and more Event-B symbols, that will be supported by this translation tool.

7.6 Conclusions

This chapter has introduced the main principles, rules and implementation solutions for the translation tools and code verification techniques for generating the target programming languages (C, C++, Java and C#) code from the Event-B specifications [EB2ALL 2011; Méry 2010a; Méry 2011d; Méry 2011c; Méry 2011b]. The syntax adopted is restrictive, but it already covers most numeric applications, supports powerful static-analysis methods and generates fast and safe source code in a target programming language. This chapter is to demonstrate an architecture of the translators and their formal verification. Many algorithms (e.g. embedded system, distributed system) are subject to further refinement. The translator provides useful assistance to human programmers by automatically adding comments, generating code for each process, optimizing expressions and partitioning event as well as data structures. The translator generates a separate code for all events of concrete modules. Systematic studies on partitioning methods using a refinement structure in target programming languages (C, C++, Java and C#) style is an interesting area of future research. This approach may be applicable to massively parallel processing.

The benefits of developing and enhancing the translation tool presented stem primarily from their increased support for automated translation between the two components of a formal model and a target programming language [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b]. It has been shown that the Event-B models have been transformed into a deterministic model [Sites 1974] for automatically translated to the source code using one more level of data refinement. The final concrete model provides sufficient refinement for introducing full determinism and use an easily translatable subset of the notations [Bert 2003]. The Rodin tool supports the development of the translation tool under the Eclipse framework using all required model information via supported interfaces. The Rodin tool uses an internal database to handle model information, which allows model generation is based on underlying meaning of a model and reduces the syntax dependency.

CHAPTER 8

Formal Logic Based Heart-Model

"Innovation is not the product of logical thought,
although the result is tied to logical structure."
(Albert Einstein)

A closed-loop model of a system is considered as a *de facto* standard in the area of system engineering for validating a system model. The cardiac pacemaker and implantable cardioverter-defibrillators (ICDs) are main critical medical devices, which require closed-loop modeling (integration of system and environment modeling) for verification purpose to obtain a certificate from the certification bodies. This chapter presents basic concepts for modeling the heart system, which provides a biological environment for building a closed-loop system for the cardiac pacemaker. The heart model is mainly based on electrocardiography analysis, which models the heart system at the cellular level. This heart model will be used for modeling the closed-loop system of the cardiac pacemaker in Part-II.

8.1 Introduction

The human heart is well known as a mechanical device of amazing efficiency that pumps blood via the circulatory system continuously throughout the person's lifetime. It is one of the most complex and important biological systems, providing oxygen and nutrients to the body to sustain life [Malmivuo 1995]. The regular impulses generated by the heart result in rhythmic contractions through a sequence of muscles in the heart, beginning at the natural pacemaker known as the sinoatrial (SA) node, which produces an action potential that travels across the atrioventricular (AV) node, the bundle of His and the Purkinje fibres distributed throughout the ventricles. The pattern and the timing of these impulses determine the heart rhythm. Variable time intervals and conduction speeds during the heartbeat generate abnormal heart rhythms, which are also known as heart rhythm impairments. Heart rhythm impairment is the principal source of several diseases [Khan 2008]. Electrocardiography analysis is frequently used to diagnose various types of heart disease [2] by presenting the timing properties of the electrical system of the heart. These are the most fundamental properties of the heart.

Cardiac pacemakers and ICDs are the two main types among the remarkable range of medical and technological devices recommended by doctors in cases of abnormal heart rhythm. These devices are used to maintain the heart rhythm, and are life-saving in many

instances. In the last few years, the use of cardiac pacemakers and cardioverter-defibrillators has increased. However, these devices may sometimes malfunction. Device-related problems have been responsible for a large number of serious injuries. Many deaths and injuries caused by device failure have been reported by the FDA [Maisel 2001], which advocates safety and security guidelines for using these devices. FDA officials have found that many deaths and injuries related to the devices are caused by product design and engineering flaws, which can be considered as firmware problems [CDRH 2006; NITRD 2009].

Providing assurance guarantees for medical devices makes formal approaches appealing. Formal model-based methods have been successful in targeted applications[Bowen 1993; Jetley 2004; Jetley 2006; Méry 2010b; Méry 2010d; Lee 2006a] of medical devices. Over the past decade, there has been considerable progress in the development of formal methods [Abrial 2010; Fitzgerald 2007; Fitzgerald 2010] to improve confidence in complex software-based systems. Although formal methods are part of the standard recommendations for developing and certifying medical systems, the integration of formal methods into the certification process is, in large part, unclear. In particular, it is a very challenging task to ensure that the end product of the software-development system behaves securely.

8.1.1 Motivation

The most challenging problem is environment modelling. That is, to validate and to verify the correct behaviour of a system model requires an interactive formal model of the environment. For example, a formal model of a cardiac pacemaker or ICD requires a heart model to verify the correctness of the developed system (see Fig. 8.1). No tools and techniques are available to provide environment modelling that would enable verification of the developed system model. Medical devices are tightly coupled with their biological environment (i.e., the heart) and use actuators and sensors to interact with the biological environment. Because of this strong relationship between the medical device (e.g., a pacemaker) and the related biological environment (i.e., the heart), it is necessary to model the functioning of the medical device within the biological environment. The environment model will be independent of the device model, which is helpful in creating an environment for medical devices that simulates the actual behaviour of the system. The medical device model will be dependent on the biological environment. Whenever an undesired state occurs in the biological environment, the device model must act according to the requirements. The main objective is to use a formal approach to modelling the medical device and the biological environment to verify the correctness of the medical system.

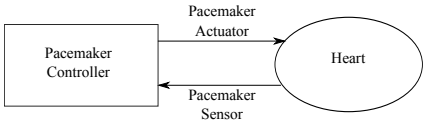


Figure 8.1: Cardiac pacemaker and Heart interaction

To model the biological environment (the heart) for a cardiac pacemaker or ICD, we propose a method for modelling the heart using logico-mathematical theory [Méry 2011f; Méry 2011i]. The heart model is based on electrocardiography analysis [Harrild 2000;

Khan 2008; Bayes 2006], which models the heart system at the cellular level [Neumann 1966]. In this investigation, we present a methodology for modelling a heart that involves extracting a set of biological nodes (SA node, AV node, etc.), impulse propagation speeds between nodes, impulse propagation times between nodes and cellular automata (CA) for propagating impulses at the cellular level. This model is developed through incremental refinement, which introduces several properties in an incremental way and verifies the correctness of the heart model. A key feature of this heart model is the representation of all possible morphological states of the (ECG) [Bayes 2006; Artigou 2007]. These morphological states represent both the normal and the abnormal states of the ECG. The morphological representation can generate any kind of heart model (a patient's model or a normal heart model) using the ECG. This model can observe both the failure of impulse generation and the failure of impulse propagation. The mathematical heart model, based on logico-mathematical theory, is verified using the Rodin [RODIN 2004] proof tool and the model checker ProB [ProB]. The model is also verified by electro-physiology and cardiac experts. The main objective of this heart model is to provide a biological environment (the heart) for formalizing a closed-loop system (a combined model of a cardiac pacemaker and the heart).

The outline of the remaining chapter is as follows. Section 2 presents the related work. A brief outline of the heart system is introduced in Section 3. Section 4 gives an idea of proposed approach. Section 5 gives an outline of the formal development of the heart model. Section 6 discusses the chapter with some lessons learned from this experience, and Section 7 concludes the chapter.

8.2 Related Work

Heart modeling is a challenging problem in the area of real-time simulation for the clinical purpose. Heart modeling problem is handled by the research community using a variety of different methods. Electrocardiogram (ECG) is an important diagnostic method to measure the heart's electrical activities, which was invented by Willem Einthoven in 1903 [Plonsey 1987]. Electrocardiogram is used for modeling the heart [Plonsey 1987]. At present time, a technological advancement technique is capable of produce a high quality cellular model of an entire heart model.

K.R. Jun et al. [Kye-Rok Jun 1994] have modeled a cellular automata model of an activation process in ventricular muscle. They have presented the 2-dimensional cellular automata model, which accounts for the local orientation of the myocardial fibers and their distributed velocity, and refractory period. A three dimensional finite volume-based computer mesh model of human atrial activation and current flow is represented by Harrild et. al [Harrild 2000]. The cellular level based this model includes both the left and right atria and the major muscle bundles of the atria. The results of this model demonstrate a normal sinus rhythm and extract the patterns of septum's activation. Due to memory and time complexity in computation of the three-dimensional model, an empirical approach is used to model the whole heart. The empirical approach means, it is a simpler representation of the complex process at the cellular level. In this new approach, researchers have adopted

some approximation to model the whole heart without compromising in the actual behavior of the heart. Berenfeld et al. [Berenfeld 1996] have developed a model that can give an insight into the local and global complex dynamics of the heart in the transition from normal to abnormal myocardial activity and help to estimate myocardial properties. Adam [Adam 1991] has analyzed the wave activities during depolarization in his cardiac model, which is represented by simplifying the heart tissue.

Recently, a real time Virtual Heart Model (VHM) has been developed by Jiang et al. [Jiang 2010] to model the electro-physiological operation of the functioning and malfunctioning. They have used time automata model to define the timing properties of the heart. Simulink Design Verifier[1] is used as a main tool for designing the model of the Virtual Heart Model (VHM). A heart model based on Uppaal Model checker [Bengtsson 1996] is developed by Eunkyoung et al. [Jee 2010] for developing the cardiac pacemaker model. This is a very simple heart model, which provides an environment to simulate and verify the pacemaker software in modeling phase. Our approach is purely based on formal techniques for modeling the heart model using electrocardiography analysis. To model the heart for a cardiac pacemaker or cardioverter-defibrillators (ICDs), we propose a method for modeling a mathematical heart model based on logico-mathematical theory, which can be implemented in any formal methods based tools (Z, TLA$^+$, VDM, etc.). Here, in this chapter, the model is developed using refinement approach at maximum at the cellular level. The incremental refinement approach helps to introduce several properties in an incremental way and to verify the correctness of the heart model [Méry 2011f; Méry 2011i]. Main key feature of this heart model is representation of all the possible morphological states of the electrocardiogram (ECG), which is used to represent the normal and abnormal states through observation of failure of the impulse generation and failure of the impulse propagation in the heart [Bayes 2006; Khan 2008; Malmivuo 1995; Artigou 2007].

8.3 Background

8.3.1 The Heart System

he human heart is wondrous in its ability to pump blood to the circulatory system continuously throughout a lifetime. The heart consists of four chambers: right atria, right ventricle, left atria and left ventricle, which contract and relax periodically. Atria forms one unit, and ventricles form another. The heart's mechanical system (the pump) requires at the very least impulses from the electrical system. An electrical stimulus is generated by the sinus node (see Fig. 8.2), which is a small mass of specialized tissue located in the right atrium of the heart. The electrical stimulus travels down through the conduction pathways and causes the heart's lower chambers to contract and pump out blood. The right and left atria are stimulated first and contract for a short period of time before the right and left ventricles. Each contraction of the ventricles represents one heartbeat. The atria contracts for a fraction of a second before the ventricles, so their blood empties into the ventricles before the ventricles contract.

[1] http://www.mathworks.com/products/sldesignverifier/

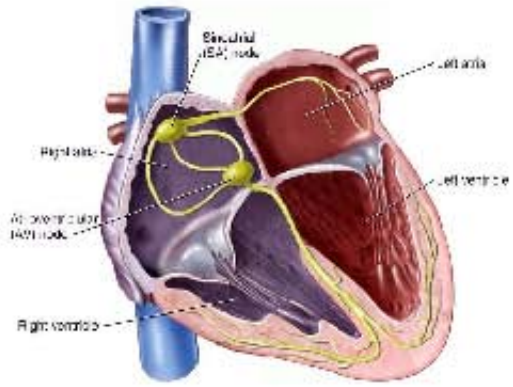


Figure 8.2: Heart or Natural Pacemaker

Arrhythmias are due to cardiac problems producing abnormal heart rhythms. In general, arrhythmias reduce haemodynamic performance, including situations where the heart develops an abnormal rate or rhythm or when normal conduction pathways are interrupted, and a different part of the heart takes over control of the rhythm. An arrhythmia can involve an abnormal rhythm increase (tachycardia; $>$ 100 bpm) or decrease (bradycardia; $<$ 60 bpm), or may be characterized by an irregular cardiac rhythm, e.g. due to asynchrony of the cardiac chambers. The irregularity of the heartbeat, called bradycardia and tachycardia. The bradycardia indicates that the heart rate falls below the expected level while in tachycardia indicates that the heart rate goes above the expected level of the heart rate. An artificial pacemaker can restore synchrony between the atria and ventricles [Barold 2004; Ellenbogen 2005; Hesselson 2003; Lee 2006b; Love 2006; Malmivuo 1995]. Beats per minute (bpm) is a basic unit to measure the rate of heart activity.

8.3.2 Basic overview of Electrocardiogram (ECG)

The electrocardiogram (ECG or EKG) [Khan 2008; Hesselson 2003] is a diagnostic tool that measures and records an electrical activity of the heart precisely in the form of signals. Clinicians can evaluate the conditions of a patient's heart from the ECG and perform further diagnosis. Analysis of these signals can be used for interpreting diagnosis of a wide range of heart conditions and predict related diseases. ECG records are obtained by sampling the bioelectric currents sensed by several electrodes, known as leads. A typical one-cycle ECG tracing is shown in Fig.-8.3. Electrocardiogram term is introduced by Willem Einthoven in 1893 at a meeting of the Dutch Medical Society. In 1924, Einthoven received the Nobel Prize for his life's work in developing the ECG [Khan 2008; Barold 2004; Love 2006; Malmivuo 1995].

The normal electrocardiogram (ECG or EKG) is depicted in Fig.-8.3. All kinds of segments and intervals are represented in this ECG diagram. Depolarization and repolarization of ventricular and atrial chambers are presented by deflection of the ECG signal. All these deflections are denoted by alphabetic order (P-QRS-T). Letter P indicates atrial depolarization, and the ventricular depolarization is represented by the QRS complex. The ventricular repolarization is represented by T-wave. Atrial repolarization appears during

the QRS complex and generates a very low amplitude signal which cannot be uncovered from the normal ECG signal.

8.3.3 ECG Morphology

Sequential activation, depolarization, and repolarization are deflected distinctly in the ECG due to the anatomical differences of the atria and the ventricles. Even all sequences are easily distinguishable when they are not in correct sequence: P-QRS-T. Each beat of the heart can be observed as a series of deflections, which reflect the time evolution of electrical activity in the heart [Bayes 2006; Khan 2008; Artigou 2007]. A single cycle of the ECG is considered as one heart beat. The ECG may be divided into the following sections:

- **P-wave:** It is a small low-voltage deflection caused by the depolarisation of the atria prior to atrial contraction as the activation (depolarisation) wave-front propagates from the SA node through the atria.
- **PQ-interval:** the time between the beginning of atrial depolarisation and the beginning of ventricular depolarisation.
- **QRS-complex:** QRS-complex are easily identifiable between P- and T-wave because it has the characteristic waveform and dominating amplitude. The dominating amplitude is caused by currents generated when the ventricles depolarise prior to their contraction. Although atrial repolarisation occurs before ventricular depolarisation, the latter waveform (i.e. the QRS-complex) is of much greater amplitude, and atrial repolarisation is therefore, not seen on the ECG.
- **QT-interval:** the time between the onset of ventricular depolarisation and the end of ventricular repolarisation. Clinical studies have demonstrated that the QT-interval increases linearly as the RR-interval increases [4]. Prolonged QT-interval may be associated with delayed ventricular repolarisation which may cause ventricular tachyarrhythmias leading to sudden cardiac death [9].
- **ST-interval:** the time between the end of S-wave and the beginning of T-wave. Significantly elevated or depressed amplitudes away from the baseline are often associated with cardiac illness.
- **T-wave:** ventricular repolarisation, whereby the cardiac muscle is prepared for the next cycle of the ECG.

8.4 Proposed Idea

Our proposed method exploits the heart model based on logico-mathematics to help the formal community to verify the correctness of the developed model of any medical device like a cardiac pacemaker. The heart model is mainly based on impulse propagation time and conduction speed at the cellular level [Méry 2011f; Méry 2011i]. This method uses

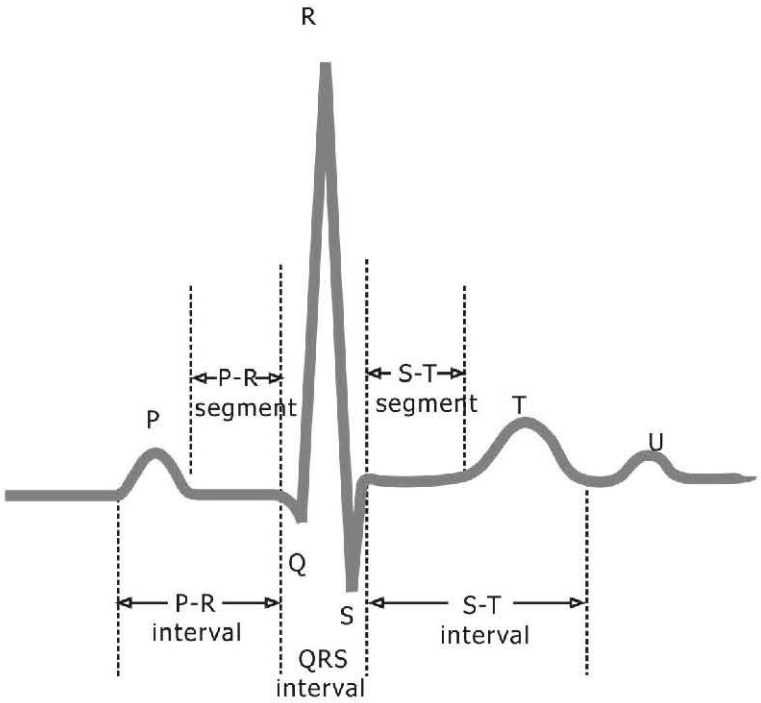


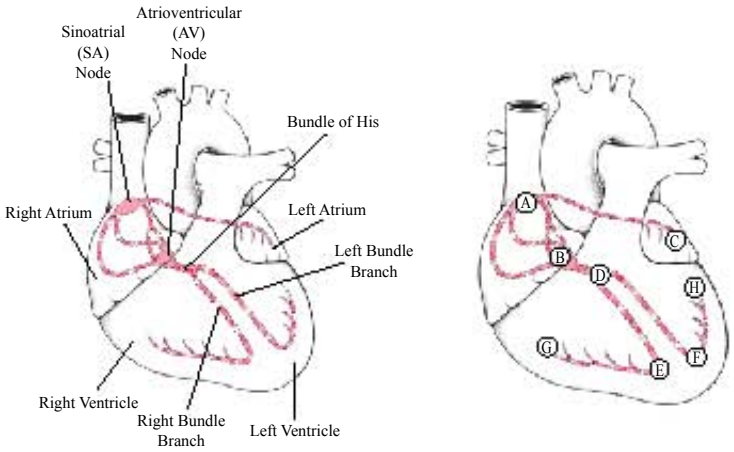
Figure 8.3: A typical one-cycle ECG tracing [Malmivuo 1995]

advance capabilities of the combined approach of formal verification and model validation using a model-checker in order to achieve considerable advantages for the heart system modeling. Fig. 8.4(a) shows the main important components and impulse conduction path in the entire heart system. The heart is a muscle with a special electrical conduction system. The system is made of two nodes (special conduction cells) and a series of conduction fibers or bundles (pathways). For modeling the heart system, we have assumed eight landmark nodes (A,B,C,D,E,F,G,H) in a whole conduction network as shown in Fig. 8.4(b), which can control the whole heart system. We have discovered all these landmarks through literature survey [Malmivuo 1995; Khan 2008] and a long discussion with cardiologist and physiologist. Centimeters per second (cm/sec.) is a basic unit to measure the conduction speed and milliseconds (ms.) is a basic unit to measure the conduction time.

Below we introduce the necessary elements to formally define our heart systems.

Definition 1 (The Heart System). *Given a set of nodes N, a transition (conduction) t is a pair (i, j), with i, j* $\in N$ *. A transition is denoted by i* $\rightsquigarrow$ *j. The heart system is a tuple HSys = (N, T,* N_0*,* TW_{time}*,* CW_{speed} *) where:*

• *N = { A, B, C, D, E, F, G, H } is a finite set of landmark nodes in the conduction pathways of the heart system;*
• $T \subseteq N \times N$ *= {A* $\mapsto$ *B, A* $\mapsto$ *C, B* $\mapsto$ *D, D* $\mapsto$ *E, D* $\mapsto$ *F, E* $\mapsto$ *G, F* $\mapsto$ *H} is a set of transitions to represent electrical impulse propagation between two landmark nodes;*
• N_0 *= A is the initial landmark node (SA node);*



(a) Basic Electrical Conduction system (b) Landmarks in network

Figure 8.4: The Electrical Conduction and Landmarks of the Heart System

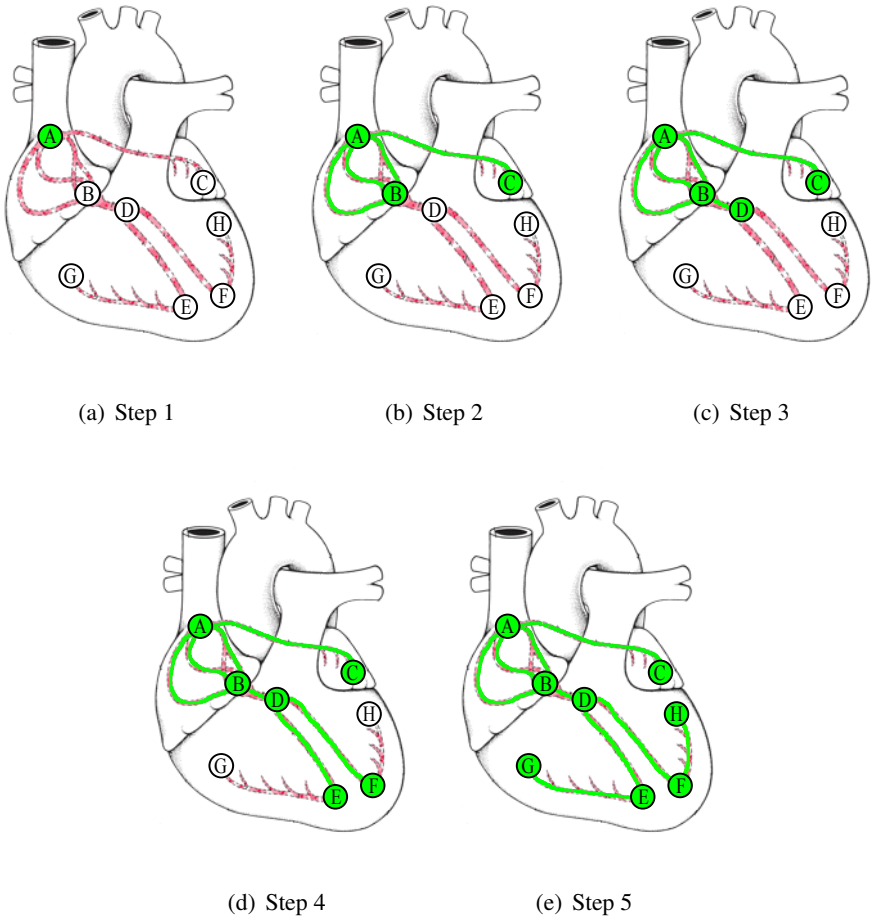
• $TW_{time} \in N \rightarrow TIME$ *is a weight function as time delay of each node, where* $TIME$ *is time delay in range;*
• $CW_{speed} \in T \rightarrow SPEED$ *is a weight function as impulse propagation speed of each transition, where* $SPEED$ *is propagation speed in range.*

Property 1 (Impulse Propagation Time). *In the heart system, electrical impulse originates from SA node (node A) and then travels through the entire conduction network and terminates to the atrial muscle fibers (node C) and at the end of Purkinje fibers into both sides of the ventricular chambers (node G and node H). Impulse propagation times delay to differ for each landmark node (N). The Impulse propagation time is represented as total function* $TW_{time} \in N \rightarrow \mathbb{P}(0..230)$. *The Impulse propagation time delay for each node (N) is represented as:* $TW_{time}(A) = 0..10$, $TW_{time}(B) = 50..70$, $TW_{time}(C) = 70..90$, $TW_{time}(D) = 125..160$, $TW_{time}(E) = 145..180$, $TW_{time}(F) = 145..180$, $TW_{time}(G) = 150..210$ *and* $TW_{time}(h) = 150..230$.

Property 2 (Impulse Propagation Speed). *Similar to the impulse propagation time, the impulse propagation speed also differs for each transition (*$i \rightsquigarrow j$*, where i, j ∈ N). The Impulse propagation speed is represented as total function* $CW_{speed} \in T \rightarrow \mathbb{P}(5..400)$. *The Impulse propagation speed for each transition is represented as:* $CW_{speed}(A \mapsto B) = 30..50$, $CW_{speed}(A \mapsto C) = 30..50$, $CW_{speed}(B \mapsto D) = 100..200$, $CW_{speed}(D \mapsto E) = 100..200$, $CW_{speed}(E \mapsto G) = 300..400$ *and* $CW_{speed}(F \mapsto H) = 300..400$.

An electrical activity is spontaneously generated by the sinoatrial (SA) node, located high in the right atrium, which is represented by the node A in Fig. 8.5(a). The sinoatrial (SA) node is the physiological pacemaker of a normal heart, responsible for setting the rate and rhythm. The electrical impulse spreads through the walls of the atria, causing them to contract. The conduction of the electrical impulse throughout the left and right atria is seen on the ECG as the P wave (see Fig. 8.3). From the sinus node, electrical impulse propagates throughout the atria and reach to the nodes B and C, but cannot propagate directly across

the boundary between atria and ventricles. The electrical impulse travels outward into atrial muscle fibers and reached at the end of muscle fibers, which is represented by the node C in the conduction network (see Fig. 8.5(b)).



(a) Step 1 (b) Step 2 (c) Step 3

(d) Step 4 (e) Step 5

Figure 8.5: Impulse Propagation through Landmarks of the Heart System

Normally, only pathway available for an electrical impulse to enter the ventricles is through a specialized region of cells called atrioventricular (AV) node. The atrioventricular node (AV node) is located at the boundary between the atria and ventricles, which is represented by the node B (see Fig. 8.4(b)). The AV node provides only conducting path from the atria to the ventricles. The AV node functions as a critical delay in the conduction system. Without this delay, the atria and ventricles would contract at the same time, and blood wouldn't flow effectively from the atria to ventricles. The delay in the AV node forms much of the PR segment on the ECG. Part of atrial repolarization can be represented by the PR segment (see Fig. 8.3).

Propagation from the atrioventricular (AV) node (A) to the ventricles is provided by a specialized conduction system. The distal portion of AV node is composed of a common bundle, called the bundle of His is denoted as a landmark node D (see Fig. 8.4(b)). The

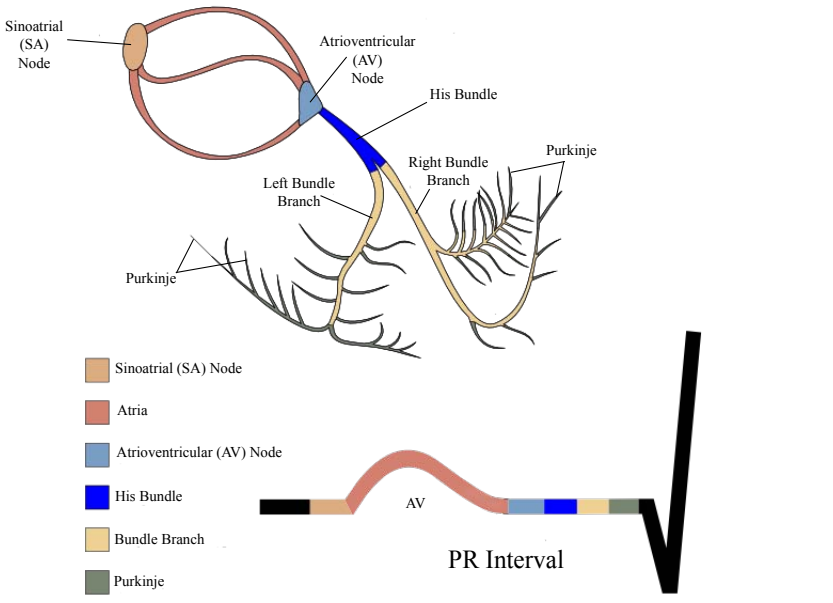


Figure 8.6: Time Intervals and Impulse Propagation in the ECG signal [Malmivuo 1995]

bundle of His splits into two branches in the inter-ventricular septum, the left bundle branch and the right bundle branch. The electrical impulses then enter the base of the ventricle at the Bundle of His (node D) and then follow the left and right bundle branches along the inter-ventricular septum (see Fig. 8.5(c)).

Two separate bundle branches propagating along each side of the septum, constituting the right and left bundle branches. We have assumed two landmark nodes E and F (see Fig. 8.4(b)) at the downside of the heart into both left and right bundle branches. These specialized fibers conduct the impulses at a very rapid velocity (see Table 8.1). The left bundle branch activates the left ventricle, while the right bundle branch activates the right ventricle (see Fig. 8.5(d)).

The bundle branches then divide into an extensive system of Purkinje fibers that conduct the impulses at high velocity (see Table 8.1) throughout the ventricles. The Purkinje fibers, stimulate individual groups of myocardial cells to contract. We have assumed finally two landmark nodes G and H (see Fig. 8.4(b)) at the end of the Purkinje fibers into both sides of the ventricles. These two nodes represent the end of the conduction network in the heart system. The bundles ramify into Purkinje fibers that diverge to the inner sides of the ventricular walls (see Fig. 8.5(e)). Upon reaching at the end of the Purkinje fibers, the electrical impulse is transmitted through the ventricular muscle mass by the ventricular muscle fibers themselves. Propagation along the conduction system takes place at a relatively high speed once it is within the ventricular region, but prior to this (through the AV node) the velocity is extremely slow [Malmivuo 1995; Khan 2008].

The electrical system provides a synchronised system between atria and ventricle, which helps in contraction of the heart muscle and optimizes haemodynamic. Changing

time intervals or conducting speeds among landmarks (see Fig. 8.4(b) and Fig. 8.6) is a major cause of generating abnormalities in the heart system. Abnormalities due to electrical signal into the heart can generate different kinds of arrhythmias. Slow conduction speed generates bradycardia, and fast conduction speed generates tachycardia. In this model, we have taken all possible sets of range values of conduction speed and conduction time for each landmark node and conduction path. This model represents the morphological structure of the ECG signal through the conduction network (see Fig. 8.6).

8.4.1 Heart Block

In this section, we have considered to explain the basic heart blocks into the heart conduction system. We have formalised the basic heart blocks in the proposed methodology. The heart block is a term given to a disorder of conduction of an impulse which stimulates heart muscle contraction. The normal cardiac impulse arises in the sinoatrial (SA) node (A) situated in the right atrium and spreads to the atrioventricular (AV) node (B), whence it is conducted by specialised tissue known as the bundle of His (D) which divides into left and right bundle branches into ventricles (see Fig. 8.4(a)). Disturbance into conduction may demonstrate as slow conduction, intermittent condition failure, or complete conduction failure. All these kinds of conduction failures are also known as 1st, 2nd and 3rd degree blocks. We have shown different kinds of heart blocks throughout the conduction network using a set of landmark nodes (see Fig. 8.7).

8.4.1.1 SA block:

This block occurs within the sinoatrial (SA) node (A). It is described as the sinoatrial (SA) nodal blocks, which is also known as a sick sinus syndrome. Sinoatrial (SA) node is failed for impulse originating at SA nodes and heart misses one or two beats at regular or irregular intervals (see Fig. 8.7(a)).

8.4.1.2 AV block:

In the situation of atrioventricular (AV) block the sinus rhythm is normal, but there is a conduction defect between the atria and the ventricles. Main reason of this block may be in the AV node (B) or bundle of His (D), or both (B, D) (see Fig. 8.7(b)).

8.4.1.3 Infra-Hisian block:

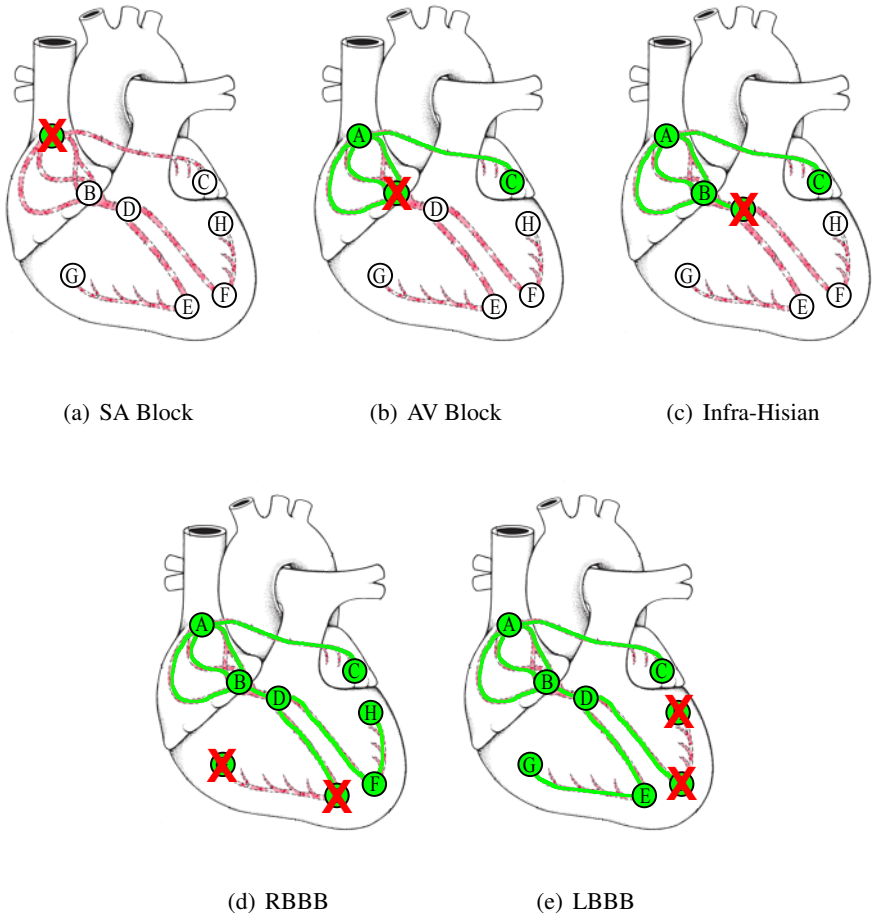
Blocks that occur below the atrioventricular (AV) node (B) are known as Infra-Hisian blocks (see Fig. 8.7(c)).

8.4.1.4 Left bundle branch block:

In the normal heart, activation of both ventricles takes place simultaneously. Left bundle branch block occurs when conduction is interrupted into the left branch bundle of His. Blocks that occur within the fascicles of the left bundle branch are known as hemiblocks (see Fig. 8.7(d)).

8.4.1.5 Right bundle branch block:

Right bundle branch block occurs when conduction is interrupted into the right branch bundle of His (see Fig. 8.7(e)).



(a) SA Block (b) AV Block (c) Infra-Hisian

(d) RBBB (e) LBBB

Figure 8.7: Impairments in Impulse Propagation due to the Heart Blocks

8.4.2 Cellular Automata Model

A set of spatially distributed cells form a cellular automata (CA) model, which contains a uniform connection pattern among neighbouring cells and local computation laws. Cellular automata (CA) is originally proposed by Ulam and von Neumann [Neumann 1966] in the 1940s to provide a formal framework for investigating the behaviour of complex, spatially distributed systems. Cellular Automata is a discrete dynamic system corresponding to the space and time. The cellular autamata modeling is the uniform property in state transitions and the interconnection patterns. The model component is specified by a single property due to same patterns instead of specifying each component separately. Cellular autamata model helps to visualize the system's dynamics [Makowiec 2008; Vangheluwe 2000;

Harrild 2000; Malmivuo 1995].

A cellular automata model can have an infinite number of cells in any dimension. Here, we consider a finite number of cells in two dimensions as shown in Fig. 8.8. A two-dimensional cellular automata model is defined as:

Definition 2 (The Cellular Automata Model).
Cellular Automata $(CA) = < S, N, T >$ *: Discrete Time System*
S : the set of states
N: the neighbouring patterns at (0,0),
T: the transition function
In the usual case of Cellular Automata (CA) realized on a D-dimensional grid, N consists of D-tuples of indices from a coordinate set:
I: $N \subseteq I^D$,
Hence the cellular model for 2D becomes,
$N \subseteq I^2$.
$T : S^{|N|} \rightarrow S$
To consider an automaton specified by the cellular automata (CA), let λ *and* α *be a global state and the global transition function of the cellular autamata (CA), respectively. Then* $\lambda = \{\tau | \tau : I^2 \rightarrow S\}$ *and* $\alpha(\lambda(i,j)) = T(\tau | N + (i,j))$ *for all* τ *in* λ *and (i,j) in* I^2.

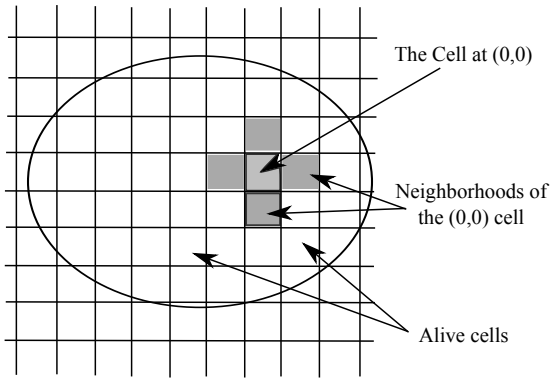


Figure 8.8: A Two-Dimensional Cellular Automata Model

Definition 3 (State Transition of a Cell). *The heart muscle system is composed of heterogeneous cells, the cellular automata model of the muscle system,* CAM_{CA}*, is characterized with no dependencies on the type of cells.* CAM_{CA} *is defined as follows:*
$CAM_{CA} = < S, N, T >$
$S = \{Active, Passive, Refractory\}$
$N = \{(0,0), (1,0), (-1,0), (0,1), (0,-1)\}$
$s'_{m,n} = s_{m,n}(t+1)$
$s'_{m,n} = T(s_{m,n}, s_{m+1,n}, s_{m-1,n}, s_{m,n+1}, s_{m,n-1})$
where, $s_{m,n}$ *denotes the state of the cell located at (m,n) and T is a transition function of cellular automata (*CAM_{CA}*), which is a function for the next state to be defined in Fig. 8.9.*

Each cell in the heart muscle should have one of the states: $Active$, $Passive$ or $Refractory$. Initially, all cells have $Passive$ state. In this state, a cell is discharged

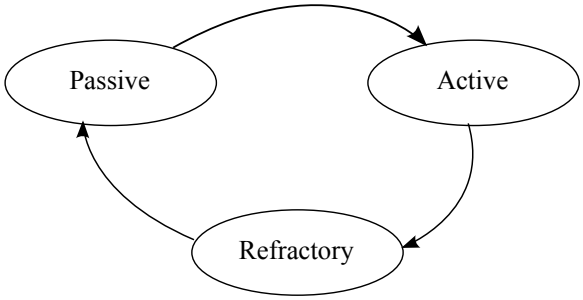


Figure 8.9: State Transition of a Cell

Location in the Heart	Cardiac Activation Time (ms.)
SA Node (A)	0..10
Left atria muscle fibers (C)	70..90
AV Node (B)	50..70
Bundle of His (D)	125..160
Right Bundle Branch (E)	145..180
Left Bundle Branch (F)	145..180
Right Purkinje fibers (G)	150..210
Left Purkinje fibers (H)	150..230

Table 8.1: Cardiac Activation Time in the Heart [Malmivuo 1995]

electrically and has no influences on its neighbouring cells. When an electrical impulse propagates, then the cell would be charged and eventually activated ($Active$ state). Now, the cell transmits the electrical impulse to its neighbour cells. The electrical impulse is propagated to all cells in the heart muscle. After an activation of the cell would be discharged and enter the $Refractory$ state in which the cell cannot be reactivated. After a moment, the cell changes its state to the $Passive$ state, in which the cell awaits next impulse.

Location in the Heart	Conduction Velocity (cm/sec.)
A $\mapsto$ B	30..50
A $\mapsto$ C	30..50
B $\mapsto$ D	100..200
D $\mapsto$ E	100..200
D $\mapsto$ F	100..200
E $\mapsto$ G	300..400
F $\mapsto$ H	300..400

Table 8.2: Cardiac Activation Velocity in the Heart [Malmivuo 1995]

8.5 Functional Formal Modeling of the Heart

To formalize the heart model, we have used Event-B modeling language, while the proposed idea can be formalized with any kind of formal methods tools like Z, ASM, TLA$^+$ and VDM etcetera. Here we have used Event-B [RODIN 2004; Abrial 2010] modeling language, which supports refinement approach [Back 1979] that helps to verify the correctness of the system in an incremental way.

Heart model development is expressed in an abstract and general way. We describe the incremental development of the heart model in several phases using step-wise refinements. The initial model formalizes the system requirements and environmental assumptions, whereas the subsequent models introduce design decisions for the resulting system. Following summary informations present global view of the heart system development, which help to understand the whole modeling approach.

Initial Model : This is an observation model, which specifies a heart state in the form of $true$ and $false$, where $true$ represents a normal rhythm and $false$ represents an abnormal rhythm of the heart.

Refinement 1 : This is a conduction model of the heart, which specifies beginning of the impulse propagation at SA node and ending of the impulse propagation at Purkinje fibers in both left and right ventricles.

Refinement 2 : This model specifies impulse propagation between landmark nodes with global clock counter to model a real-time system to satisfy the temporal properties of impulse propagation.

Refinement 3 : This is a perturbation model of the heart, which specifies perturbation in the heart conduction system and helps to discover exact block into the heart conduction system.

Refinement 4 : This is a simulation model of the heart, which introduces impulse propagation at the cellular level using cellular automata.

8.5.1 The Context and Initial Model

Event-B models are described in two major components: *context* and *machine*. Context contains the static part of a model whereas machine contains the dynamic part. The context uses sets and constants to define axioms and theorems. Axioms and theorems represent the logical theory of the elements of a system. The logical theory lists static properties of constants related to a system and provides an axiomatization of system environment. Context can be extended by other contexts and referenced by a set of machines, while a machine can be refined by machines.

We need to choose the electrical features for modeling the heart system. To model the heart system, we identify a set of electrical impulse propagation nodes *ConductionNode* of the heart conduction network (see Fig. 8.4(a)). These nodes are basic landmarks, which enable expression of the normal and abnormal behavior of the heart system. These landmarks were identified through a literature survey [Malmivuo 1995; Khan 2008] and fruitful discussions with a cardiologist and a physiologist. Three constants define impulse propagation time *ConductionTime*, impulse propagation path *ConductionPath*, and impulse

propagation velocity *ConductionSpeed*. Static properties are defined in the context model for specifying an electrical impulse propagation network of the heart system, impulse propagation time for each landmark node, and impulse propagation speed for every path. A path is represented by pair of landmark nodes (see Definition 1, Properties 1 and 2, Table 8.1).

$$
\begin{array}{l}
axm1 : partition(ConductionNode, \{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{F\}, \{G\}, \{H\}) \\
axm2 : ConductionTime \in ConductionNode \rightarrow \mathbb{P}(0 \mathinner{..} 230) \\
axm3 : ConductionPath \subseteq ConductionNode \times ConductionNode \\
axm4 : ConductionSpeed \in ConductionPath \rightarrow \mathbb{P}(5 \mathinner{..} 400)
\end{array}
$$

As you see axioms are extracted from the definitions and are validated by cardiologist and physiologist.

8.5.2 Abstract Model

We define an abstract model for indicating the heart state according to the observation impulse propagation on the conduction nodes. The machine model represents a dynamic behavior of the heart system through the step-wise impulse propagation into the atria and ventricular chambers. To define the dynamic properties, we have introduced four variables *ConductionNodeState*, *CConductionTime*, *CConductionSpeed* and *HeartState* in invariants. The variable *ConductionNodeState* is defined as a function, which shows boolean states of the landmark nodes. When the electrical impulse passes through the landmark nodes (see Fig. 8.4(b)), then the visited nodes become *TRUE* and the unvisited landmark nodes are represented by *FALSE*. The variables *CConductionTime* and *CConductionSpeed* represent current impulse propagation time and velocity in the conduction network. The last variable *HeartState* represents a boolean state *TRUE* or *FALSE*. *TRUE* represents the normal condition of the heart while *FALSE* represents an abnormal condition of the heart.

$$
\begin{array}{l}
inv1 : ConductionNodeState \in ConductionNode \rightarrow BOOL \\
inv2 : CConductionTime \in ConductionNode \rightarrow 0 \mathinner{..} 300 \\
inv3 : CConductionSpeed \in ConductionPath \rightarrow 0 \mathinner{..} 500 \\
inv4 : HeartState \in BOOL
\end{array}
$$

In the abstract specification of the heart model, there are three events, namely *HeartOK* to represent a normal state of the heart, *HeartKO* to express abnormal state of the heart and *HeartConduction* to update the value of each landmark node of the conduction network in terms of visited landmark nodes (*ConductionNodeState*), impulse propagation intervals (*CConductionTime*) and impulse propagation velocities (*CConductionSpeed*).

The event *HeartOK* specifies a set of required conditions for a normal state of the heart system. The first guard *grd1* states that all landmark nodes should be visited in a single cycle of impulse propagation. The second guard states that the current impulse propagation time of each landmark node should lie within the pre-specified range of the impulse propagation times. The final guard states that the current impulse propagation velocity of each path should lie between pre-defined impulse propagation velocities. If all guards are satisfied then the heart state indicates the normal condition as being *TRUE*.

EVENT HeartOK
WHEN
grd1 : $\forall i \cdot i \in ConductionNode \Rightarrow ConductionNodeState(i) = TRUE$
grd2 : $\forall i \cdot i \in ConductionNode \Rightarrow CConductionTime(i) \in ConductionTime(i)$
grd3 : $\forall i, j \cdot i \mapsto j \in ConductionPath \Rightarrow$
$CConductionSpeed(i \mapsto j) \in ConductionSpeed(i \mapsto j)$
THEN
act1 : $HeartState := TRUE$
END

The event *HeartKO* specifies as an opposite set of guards to those for the normal state of the heart system to specify abnormal conditions of the heart. These guards state that if any landmark node is not visited in a single cycle of the impulse propagation, or if any the current impulse propagation time of any landmark node does not lie within the pre-specified range of the impulse propagation times, or if the current impulse propagation velocity of any path does not lie within the pre-defined range of impulse propagation velocities, then the heart system is in an abnormal state represents by its normal condition being *FALSE*. Different kinds of heart diseases affect the electrical impulse propagation time and velocity in the heart system [Khan 2008]. These changes affect the actual heart rhythm and help to identify the possible abnormal behaviours of the heart.

EVENT HeartKO
WHEN
grd1 : $\exists i \cdot i \in ConductionNode \wedge ConductionNodeState(i) = FALSE)$
$\vee$
$(\exists j \cdot j \in ConductionNode \wedge CConductionTime(j) \notin ConductionTime(j))$
$\vee$
$(\exists m, n \cdot m \mapsto n \in ConductionPath \wedge CConductionSpeed(m \mapsto n)$
$\notin ConductionSpeed(m \mapsto n))$
THEN
act1 : $HeartState := FALSE$
END

The event *HeartConduction* formalizes the heart behaviour in an abstract manner by updating the values for impulse propagation time, impulse propagation velocity and visited state of the landmark nodes non-deterministically. This event is used to model more concrete behaviour of the heart system at the next level of refinement.

EVENT HeartConduction
BEGIN
act1 : $ConductionNodeState :\in ConductionNode \rightarrow BOOL$
act2 : $CConductionTime :\in ConductionNode \rightarrow 0 \mathinner{..} 300$
act3 : $CConductionSpeed :\in ConductionPath \rightarrow 0 \mathinner{..} 500$
act4 : $HeartState :\in BOOL$
END

8.5.3 Refinement 1: Introducing Steps in the Propagation

In the abstract model, we have presented that the impulse propagation time, velocity and visited landmark nodes have been updated in an atomic step when electrical impulse fire from the sinus (SA) node and moves towards the Purkinje fibers into ventricles (G, H nodes) and in the left atria muscle fibers (C node). Our main objective is to model step by step impulse propagation through all landmark nodes, where the electrical impulse must pass through a number of intermediate landmark nodes before reaching to the terminal nodes (C, G, H). This refinement is a very simple refinement, where we introduce two extra events *SinusNodeFire* and *HeartConductionEnd* as the refinement of the event *HeartConduction*. The event *SinusNodeFire* models the behavior of a sinoatrial (SA) node, which originates electrical impulse for traversing throughout the heart system using the conduction network (see Fig. 8.4). Guards of this event state that if all landmark nodes are unvisited (means FALSE state) and current impulse propagation time of each node is 0, and impulse propagation velocity of each path is 0, then the conduction node state *ConductionNodeState* of a landmark node A (SA node) sets TRUE and current impulse propagation time of SA node (A) sets to 0.

EVENT SinusNodeFire Refines HeartConduction
WHEN
grd1 : $\forall n \cdot n \in ConductionNode \Rightarrow ConductionNodeState(n) = FALSE$
grd2 : $\forall n \cdot n \in ConductionNode \Rightarrow CConductionTime(n) = 0$
grd3 : $\forall n, m \cdot n \in ConductionNode \wedge m \in ConductionNode \wedge$
$n \mapsto m \in ConductionPath \Rightarrow CConductionSpeed(n \mapsto m) = 0$
THEN
act1 : $ConductionNodeState(A) := TRUE$
act2 : $CConductionTime(A) := 0$
END

The next event *HeartConductionEnd* represents end state of the impulse propagation into Purkinje fibers of ventricles (G, H nodes) and left atria muscle (node C). This event resets all variables for generating next impulse at the SA node. The actions of this event reset all conduction node state as *FALSE*, current impulse propagation time of all landmark nodes reset to 0, current impulse propagation velocity of all landmark nodes reset to 0, and the heart state sets as *FALSE*. All these actions are required before originating the next electrical impulse from the SA node (A).

EVENT HeartConductionEnd Refines HeartConduction
BEGIN
act1 : $ConductionNodeState := \{A \mapsto FALSE, B \mapsto FALSE, C \mapsto FALSE,$
$D \mapsto FALSE, E \mapsto FALSE, F \mapsto FALSE, G \mapsto FALSE, H \mapsto FALSE\}$
act2 : $CConductionTime := \{A \mapsto 0, B \mapsto 0, C \mapsto 0, D \mapsto 0,$
$E \mapsto 0, F \mapsto 0, G \mapsto 0, H \mapsto 0\}$
act3 : $CConductionSpeed := \{A \mapsto B \mapsto 0, A \mapsto C \mapsto 0, B \mapsto D \mapsto 0,$
$D \mapsto E \mapsto 0, D \mapsto F \mapsto 0, E \mapsto G \mapsto 0, F \mapsto H \mapsto 0\}$
act4 : $HeartState := FALSE$
END

8.5.4 Refinement 2: Impulse Propagation

In the second refinement, we introduce several events as a refinement of the event *Heart-Conduction* to model the impulse propagation into the heart conduction network. New events are formalizing impulse flow between two landmark nodes separately; for instance, electrical impulse moves from SA node (A) to AV node (B). This level of refinement introduces seven events for modeling the whole conduction path from originating nodes (A) to the ending nodes (C, G, H). A variable *CCSpeed_CCTime_Flag* is introduced as a boolean type to capture the value of current impulse propagation time and current impulse propagation velocity. A new variable *Cycle_Length* declares a time interval for the single heart beat, which may change in every cycle of an electrocardiogram (ECG). This refinement also introduces a logical clock to synchronise all states of the heart system and checks the heart states under a required time length in the conduction network. A new variable *tic* is defined as current *clock counter*. Invariants (*inv4-inv10*) are introduced as safety properties, which define that if the heart state is *TRUE* then the impulse propagation time and the impulse propagation velocity are always lied within the standard range of time and velocity during the impulse conduction throughout the conduction network (see Fig. 8.4(b)).

$$
\begin{aligned}
&inv1 : CCSpeed\_CCTime\_Flag \in BOOL \\
&inv2 : Cycle\_Length \in 500..2000 \\
&inv3 : tic \in \mathbb{N} \\
\\
&inv4 : HeartState = TRUE \Rightarrow CConductionTime(B) \in ConductionTime(B) \land \\
&\qquad CConductionSpeed(A \mapsto B) \in ConductionSpeed(A \mapsto B) \\
&inv5 : HeartState = TRUE \Rightarrow CConductionTime(C) \in ConductionTime(C) \land \\
&\qquad CConductionSpeed(A \mapsto C) \in ConductionSpeed(A \mapsto C) \\
&inv6 : HeartState = TRUE \Rightarrow CConductionTime(D) \in ConductionTime(D) \land \\
&\qquad CConductionSpeed(B \mapsto D) \in ConductionSpeed(B \mapsto D) \\
&inv7 : HeartState = TRUE \Rightarrow CConductionTime(E) \in ConductionTime(E) \land \\
&\qquad CConductionSpeed(D \mapsto E) \in ConductionSpeed(D \mapsto E) \\
&inv8 : HeartState = TRUE \Rightarrow CConductionTime(F) \in ConductionTime(F) \land \\
&\qquad CConductionSpeed(D \mapsto F) \in ConductionSpeed(D \mapsto F) \\
&inv9 : HeartState = TRUE \Rightarrow CConductionTime(G) \in ConductionTime(G) \land \\
&\qquad CConductionSpeed(E \mapsto G) \in ConductionSpeed(E \mapsto G) \\
&inv10 : HeartState = TRUE \Rightarrow CConductionTime(H) \in ConductionTime(H) \land \\
&\qquad CConductionSpeed(F \mapsto H) \in ConductionSpeed(F \mapsto H)
\end{aligned}
$$

Events are introduced in this refinement to model the impulse propagation from SA node towards the Purkinje fibers landmark nodes (G, H) and atria fibers nodes (C). Each event is synchronised through progressive electrical impulse propagation in the conduction network. We have given formalization of only one event *HeartConduction_A_B* to understand the basic formalization steps of all other events. All other events of impulse propagation in the conduction network among landmark nodes have been modeled in a similar fashion.

EVENT HeartConduction_A_B Refines HeartConduction
WHEN
grd1 : $ConductionNodeState(A) = TRUE$
grd2 : $ConductionNodeState(B) = FALSE$
grd3 : $CConductionTime(B) \in ConductionTime(B)$
grd4 : $CConductionSpeed(A \mapsto B) \in ConductionSpeed(A \mapsto B)$
grd5 : $CCSpeed\_CCTime\_Flag = FALSE$
THEN
act1 : $ConductionNodeState(B) := TRUE$
act2 : $CCSpeed\_CCTime\_Flag := TRUE$
END

A new event *Update_CCSpeed_CCtime* is a refinement of the event *HeartConduction*. This event is used to capture the current electrical impulse propagation time *CConduction-Time* and the current electrical impulse propagation speed *CCconductionSpeed* during a progressive conduction flow into the heart system in the conduction network.

EVENT Update_CCSpeed_CCtime Refines HeartConduction
ANY $i, j, CSpeed, CTime$
WHERE
grd1 : $i \in ConductionNode$
grd2 : $j \in ConductionNode$
grd3 : $i \mapsto j \in ConductionPath$
grd4 : $CSpeed \in 0 \,..\, 500$
grd5 : $CTime \in 0 \,..\, 300$
grd6 : $CCSpeed\_CCTime\_Flag = TRUE$
grd7 : $HeartState = FALSE$
grd8 : $tic = CTime$
THEN
act1 : $CConductionTime(j) := CTime$
act2 : $CConductionSpeed(i \mapsto j) := CSpeed$
act3 : $CCSpeed\_CCTime\_Flag := FALSE$
END

The electrical impulse propagates at every millisecond. But the impulse propagation time and velocity are different for each landmark node. The progressive increment of the independent logical clock is modeled through event *tic*, that increments time in 1 ms. The event *Clock_Counter* progressively increases the current clock counter tic under pre-defined cycle length $Cycle\_Length$. The predicate in guard $(grd1)$ of event *Clock_-Counter* represents an upper bound time limit. The current clock counter tic is reset to 0 by the event *HeartConductionEnd*. An extra guard is added in the event *HeartConductionEnd* as $tic = Cycle\_Length$ to reset all the parametric values of the heart system for starting a fresh new impulse propagation cycle.

EVENT Clock_Counter
WHEN
grd1 : $tic < Cycle\_Length$
THEN
act1 : $tic := tic + 1$
END

We have defined the event *Clock_Counter* as a type of $Convergent$ and the system variant is defined as $Cycle\_length - tic$, which generates the convergence proof obligations to verify that the time is progressing with the electrical impulse propagation. It means that the electrical impulse is propagating in the conduction network corresponding to the clock counter.

8.5.5 Refinement 3: Perturbation the Conduction

It introduces a set of possible blocks in the heart conducting system. These blocks can occur into the conduction network and give trouble into electrical impulse propagation. A set of landmark nodes partition the different regions for all possible heart blocks. For introducing the heart blocks, we introduce an enumerated set *HeartBlockSets* in a new context model as a static property of the heart system.

$$axm1 : partition(HeartBlockSets, \{SA\_nodal\_blocks\}, \{AV\_nodal\_blocks\}, \{Infra\_Hisian\_blocks\}, \{LBBB\_blocks\}, \{RBBB\_blocks\}, \{None\})$$

To model the heart block system, we define a variable *HeartBlocks* as $HeartBlocks \in HeartBlockSets$. New events are introduced to show different kinds of heart blocks during impulse propagation into the conduction network. Events are *HeartConduction_Block_A_B_C* to formalise the sinoatrial (SA) nodal block, *HeartConduction_Block_B* to represent atrioventricular (AV) nodal block, *HeartConduction_Block_B_D* to specify Infra-Hisian block, *HeartConduction_Block_D_E_G* to present Left bundle branch block, and *HeartConduction_Block_D_F_H* to specify the Right bundle branch block.

Conduction disturbance in the heart during which an impulse formed within the sinus node (A) is blocked or delayed from depolarizing the atria. There are different kinds of SA blocks [Malmivuo 1995; Khan 2008]. To model SA block, we introduce an event *HeartConduction_Block_A_B_C*, which formalises the SA block. In this event, guard ($grd1$) represents that the landmark nodes (A or C) are not visited means FALSE state, or the current impulse propagation time of B and C nodes are not lied within the standard range, or the current impulse propagation velocity of the pairs $A \mapsto B$ and $A \mapsto C$ are not lied within the standard range. When a guard is triggered, then actions of this event state that the heart state is FALSE, and the heart block is a sinoatrial (SA) nodal block.

EVENT HeartConduction_Block_A_B_C Refines HeartKO
WHEN
grd1 : $(ConductionNodeState(A) = FALSE) \vee$
$(ConductionNodeState(C) = FALSE) \vee$
$(CConductionTime(B) \notin ConductionTime(B)) \vee$
$(CConductionTime(C) \notin ConductionTime(C)) \vee$
$(CConductionSpeed(A \mapsto B) \notin ConductionSpeed(A \mapsto B)) \vee$
$(CConductionSpeed(A \mapsto C) \notin ConductionSpeed(A \mapsto C))$
THEN
act1 : $HeartState := FALSE$
act2 : $HeartBlocks := SA\_nodal\_blocks$
END

Any interruption in the conduction of electrical impulses from the atria to the ventricles; it can occur at the level of atria, atrioventricular node, bundle of His, or Purkinje system. It is a type of heart block in which a blocking is at the atrioventricular (AV) junction. It is known as first degree when atrioventricular (AV) conduction time is prolonged; it is called second degree or partial when some but not all atrial impulses reach the ventricle; and it is called third degree or complete when no atrial impulses at all reach the ventricle, so that the atria and ventricles act independently of each other. There are different kinds of AV blocks [Malmivuo 1995; Khan 2008]. To model the AV block, we introduce an event *HeartConduction_Block_B*, which formalises the AV block. The conduction node state *ConductionNodeState* of a landmark node (B) is *FALSE*, which represents a condition for the AV block using guard ($grd1$) and actions state that the heart state is *FALSE* and such kind of heart block is known as the atrioventricular (AV) nodal block.

EVENT HeartConduction_Block_B Refines HeartKO
WHEN
grd1 : $(ConductionNodeState(B) = FALSE)$
THEN
act1 : $HeartState := FALSE$
act2 : $HeartBlocks := AV\_nodal\_blocks$
END

Infra-Hisian block describes a block of the distal conduction system (node D). There are different kinds of Infra-Hisian blocks [Malmivuo 1995; Khan 2008]. To model Infra-Hisian block, an event *HeartConduction_Block_B_D* is used to formalise the desired conditions for a such kind of blocks through landmark nodes (B, D). Guard ($grd1$) represents that the landmark node (D) is *FALSE*, means it is not visited, or the current impulse propagation time of a node D is not lied within the standard range, or the current propagation velocity of a pair $B \mapsto D$ is not lied within the standard range. Actions of this event state that the heart state is FALSE, and the heart block is the Infra-Hisian block.

EVENT HeartConduction_Block_B_D Refines HeartKO
WHEN
grd1 : $(ConductionNodeState(D) = FALSE) \vee$
$(CConductionTime(D) \notin ConductionTime(D)) \vee$
$(CConductionSpeed(B \mapsto D) \notin ConductionSpeed(B \mapsto D))$
THEN
act1 : $HeartState := FALSE$
act2 : $HeartBlocks := Infra\_Hisian\_blocks$
END

The bundle of His divides into a right bundle branch and a left bundle branch, which lead to the heart's lower chambers (the ventricles). For the left and right ventricles to contract at the same time, an electrical impulse must travel down the right and left bundle branches at the same speed. If there is a block in one of these branches, the electrical impulse must travel to the ventricle by a different route. When this happens, the rate and rhythm of your heartbeat are not affected, but the impulse is slowed. Even ventricle will still contract, but it will take longer because of the slowed impulse. This slowed impulse causes one ventricle to contract a fraction of a second slower than the other [Malmivuo 1995; Khan 2008]. The medical terms for bundle branch block are derived from which branch is affected. If the block is located in the right bundle branch, it is called Right bundle branch block. If the block is located in the left bundle branch, it is called Left bundle branch block.

To model the Right bundle branch block, we introduce an event in a similar fashion like past events. A new event *HeartConduction_Block_D_E_G* formalises the Right bundle branch; a guard of this event states that the landmark nodes (E or G) are not visited means FALSE state, or the current impulse propagation time of E and G nodes are not lied within the standard ranges, or the current impulse propagation velocity of the pairs $D \mapsto E$ and $E \mapsto G$ are not lied within the standard range; then the actions of this event state that the heart state is FALSE and the heart block is the Right bundle branch block.

EVENT HeartConduction_Block_D_E_G Refines HeartKO
WHEN
grd1 : $(ConductionNodeState(E) = FALSE) \vee$
$(ConductionNodeState(G) = FALSE) \vee$
$(CConductionTime(E) \notin ConductionTime(E)) \vee$
$(CConductionTime(C) \notin ConductionTime(C)) \vee$
$(CConductionSpeed(D \mapsto E) \notin ConductionSpeed(D \mapsto E)) \vee$
$(CConductionSpeed(E \mapsto G) \notin ConductionSpeed(E \mapsto G))$
THEN
act1 : $HeartState := FALSE$
act2 : $HeartBlocks := RBBB\_blocks$
END

To model the Left bundle branch block, we introduce an event like Right bundle branch event. This new event *HeartConduction_Block_D_F_H* formalises the Left bundle branch. A guard of this event states that the landmark nodes (F or H) are not visited means FALSE state, or the current impulse propagation time of F and H nodes are not lied within the

standard range, or the current impulse propagation velocity of the pairs $D \mapsto F$ and $F \mapsto H$ are not lied within the standard range. Then the actions of this event state that the heart state is FALSE, and the heart block is the Left bundle branch block.

EVENT HeartConduction_Block_D_F_H Refines HeartKO
WHEN
grd1 : $(ConductionNodeState(F) = FALSE) \lor$
$(ConductionNodeState(H) = FALSE) \lor$
$(CConductionTime(F) \notin ConductionTime(F)) \lor$
$(CConductionTime(H) \notin ConductionTime(H)) \lor$
$(CConductionSpeed(D \mapsto F) \notin ConductionSpeed(D \mapsto F)) \lor$
$(CConductionSpeed(F \mapsto H) \notin ConductionSpeed(F \mapsto H))$
THEN
act1 : $HeartState := FALSE$
act2 : $HeartBlocks := LBBB\_blocks$
END

8.5.6 Refinement 4: Getting a Cellular Model

This last refinement introduces cellular level modeling into the heart model. The cellular level modeling is used to model the electrical impulse propagation at the cell level. The formalisation uses cellular automata theory to model the micro-structure based cell model. To formalise the cellular automata, we introduce mathematical properties (see Definition 2 and 3) in a context model. In a biological system, each cell has one of the following states: *Active*, *Passive* or *Refractory*. To define cell states, we declare an enumerated set *CellStates*. We have assumed grid of cells in a square format. Due to square geometry of the cells, we define a constant *NeighbouringCells* to represent a set of coordinated positions of the neighbouring cells. A new function *NEXT* is used to define neighbouring cell's state. This function maps from the power-set of *NeighbouringCells* to a cell's state *CellStates*. A new function *CellS* is defined as to map from *NeighbouringCells* to *CellStates*. This function maps various state like *Active*, *Passive* and *Refractory* to the neighbouring cells.

$axm1 : partition(CellStates, \{PASSIVE\}, \{ACTIVE\}, \{REFRACTORY\})$
$axm2 : x \in \mathbb{Z}$
$axm3 : y \in \mathbb{Z}$
$axm4 : NeighbouringCells =$
$\{\{x, y\}, \{x + 1, y\}, \{x - 1, y\}, \{x, y + 1\}, \{x, y - 1\}\}$
$axm5 : NEXT \in \mathbb{P}(NeighbouringCells) \rightarrow CellStates$
$axm6 : CellS \in NeighbouringCells \rightarrow CellStates$

A set of properties *(axm7-axm10)* is introduced to specify the desired behavior of the biological cell automata in two-dimensions. All these properties implement the state transition of a cell and formalise the transitions automaton (see Fig. 8.9). The first property ($axm1$) states that if the neighbouring cells are in $Active$ state, then the NEXT state of the cell must be *Refractory*. The second property ($axm8$) represents that if the neighbouring cells are in *Refractory* state, then the NEXT state of the cell must be *Passive*. Third

property $(axm9)$ states that if a cell at (x, y) is *Passive*, then if all the neighbouring cells in 2D is *Active*, then a set of neighbouring cells must be in *Active*. Similarly, last property $(axm10)$ presents that if a cell at (x, y) is *Passive*, then and if all the neighbouring cells in 2D is *Active*, then a set of neighbouring cells must be in *Active*.

$axm7 : \forall\, param \cdot param \in \mathbb{P}(NeighbouringCells) \land CellS(\{x, y\}) = ACTIVE \Rightarrow$
$\quad NEXT(param) = REFRACTORY$
$axm8 : \forall\, param \cdot param \in \mathbb{P}(NeighbouringCells) \land CellS(\{x, y\}) =$
$\quad REFRACTORY \Rightarrow NEXT(param) = PASSIVE$
$axm9 : \forall\, param \cdot param \in \mathbb{P}(NeighbouringCells) \land \{x, y\} \in paramCellS(\{x, y\}) =$
$\quad PASSIVE \Rightarrow ((CellS(\{x + 1, y\}) = ACTIVE \lor CellS(\{x - 1, y\}) =$
$\quad ACTIVE \lor CellS(\{x, y + 1\}) = ACTIVE \lor CellS(\{x, y - 1\}) =$
$\quad ACTIVE) \Rightarrow NEXT(param) = ACTIVE)$
$axm10 : \forall\, param \cdot param \in \mathbb{P}(NeighbouringCells) \land \{x, y\} \in param \land CellS(\{x, y\})$
$\quad = PASSIVE \Rightarrow ((CellS(\{x + 1, y\}) \neq ACTIVE \land CellS(\{x - 1, y\}) \neq$
$\quad ACTIVE \land CellS(\{x, y + 1\}) \neq ACTIVE \land CellS(\{x, y - 1\}) \neq$
$\quad ACTIVE \Rightarrow NEXT(param) = PASSIVE)$

Each cell in the heart muscle must have one of the states: *Active*, *Passive* or *Refractory*. Initially, all cells have *Passive* state. In this state, a cell is discharged electrically and has no influences on its neighbouring cells. When electrical impulse propagates, then the cell would be charged and eventually activated (*Active* state). Now, the cell transmits the electrical impulse to its neighbour cells. The electrical impulse is propagated to all cells in the heart muscle. After an activation, the cell would be discharged and enter into the *Refractory* state in which a cell cannot be reactivated after a moment, a cell changes its state to the *Passive* state, in which the cell awaits next impulse (see Fig. 8.9).

To model the dynamic behavior of the cell automata, we declare four variables m, n, *Transition* and *NextCellState*. Two variables m and n represent current position of the active cell during impulse propagation. The variable *Transition* is defined as boolean to set the transition state $TRUE$ or $FALSE$ to model the behavior of a tissue. Last variable *NextCellState* is used to store the values of next neighbouring positions after every transition.

$inv1 : m \in \mathbb{Z}$
$inv2 : n \in \mathbb{Z}$
$inv3 : Transition \in BOOL$
$inv4 : NextCellState \in CellStates$

To implement the dynamic behavior of a cell in two-dimensions, we introduce two events *HeartConduction_Cellular* to make transition TRUE for the electrical conduction at the cell level and *HeartConduction_Next_UpdateCell* to calculate status of the neighbouring cells and update the current position (m, n) of the cell. The event *HeartConduction_-Cellular* is used to set the boolean states of the variable *Transition*. First guard of this event states that any path $(p \mapsto q)$ is one of the pair from a set of pairs of the conduction network. Next guard $(grd2)$ states that the current impulse propagation speed and velocity flag *CC-Speed_CCTime_Flag* is TRUE and a set of coordinate positions $(param)$ of neighbouring

cells is represented in third guard. Fourth guard states that the current cell position (m, n) is *Passive* and last guard represents that the cell transition state $Transition$ is *FALSE*. If all guards satisfy, then the transition state of a cell becomes *TRUE*.

EVENT HeartConduction_Cellular
ANY $p, q, param$
WHERE
grd1 : $p \mapsto q \in ConductionPath$
grd2 : $CCSpeed\_CCTime\_Flag = TRUE$
grd3 : $param = \{\{m, n\}, \{m+1, n\}, \{m-1, n\}, \{m, n+1\}, \{m, n-1\}\}$
grd4 : $\{m, n\} \in dom(CellS) \land CellS(\{m, n\}) = PASSIVE$
grd5 : $NextCellState = CellS(\{m, n\})$
grd6 : $Transition = FALSE$
THEN
act1 : $Transition := TRUE$
END

The event *HeartConduction_Next_UpdateCell* is used to calculate the state of neighbouring cells and update the position of a current cell (m, n). First guard of this event represents a set of coordinate positions $(param)$ of neighbouring cells and next guard $(grd2)$ states that selected neighbouring cells are a set of cells $(dom(NEXT))$. The last guard presents a transition state *Transition* is $TRUE$. Action of this event calculates a set of the next neighbouring cells in $act1$. Next action $(act2)$ sets $FALSE$ of a transition state. The last two actions update the value of a current cell (m, n) to continuously impulse propagation in the heart using the conduction network.

EVENT HeartConduction_Next_UpdateCell
ANY $param$
WHERE
grd1 : $param = \{\{m, n\}, \{m+1, n\}, \{m-1, n\}, \{m, n+1\}, \{m, n-1\}\}$
grd2 : $param \in dom(NEXT)$
grd3 : $Transition = TRUE$
THEN
act1 : $NextCellState := NEXT(param)$
act2 : $Transition := FALSE$
act3 : $m :\in \{m-1, m, m+1\}$
act4 : $n :\in \{n-1, n, n+1\}$
END

Finally, we have completed the formal specifications of the heart modeling. In the next section, we present model validation of the heart model using Event-B model checker ProB tool.

8.5.7 Model Validation and Analysis

There are two main validation activities in Event-B, and both are complementary for designing a consistent system in the medical domain:

Model	Total number of POs	Automatic Proof	Interactive Proof
Abstract Model	29	22(76%)	7(24%)
First Refinement	9	6(67%)	3(33%)
Second Refinement	159	155(97%)	4(3%)
Third Refinement	10	1(10%)	9(90%)
Fourth Refinement	11	10(91%)	1(9%)
Total	218	194(89%)	24(11%)

Table 8.3: Proof Statistics

- *consistency checking*, which is used to show that the events of a machine preserve the invariant, and *refinement checking*, which is used to show that one machine is a valid refinement of another. A list of automatically generated proof obligations should be discharged by the proof tool of the Rodin platform.
- *model analysis*, which is done by the ProB tool and consists in exploring traces or scenarios of our consistent Event-B models. For instance, the ProB may discover possible deadlocks or hidden properties that are not expressed by generated proof obligations.

This section validates the model by using ProB tool [ProB ; Leuschel 2003] and proof statistics. "Validation" refers to the activity of gaining confidence that the developed formal models are consistent with the requirements. We have used the ProB tool [ProB] that supports *automated consistency checking* of Event-B machines via model checking [Clarke 1999] and constraint-based checking [Jackson 2002]. This tool assists us to validate the heart model according to the conduction network and a set of landmark nodes. It is the complementary use of both techniques to develop formal models of critical systems, where high safety and security are required. The heart model is carefully verified through animations and under supervision of physiologist and cardiologist. We have validated various scenario cases of normal and abnormal heart conditions, and we have also tested morphological behavior [Bayes 2006; Artigou 2007] of the ECG during impulse propagation from the SA node (A) to the Purkinje fibers (F, H) in the ventricles. The logic-based mathematical model of the heart can generate all possible scenarios of normal and abnormal heart conditions in the ECG caused by changes in time and velocity among landmark nodes. ProB was very useful in animating all models and in verifying the absence of error (no counter-examples exist) and deadlock.

Table 8.3 expresses the proof statistics of the development using the Rodin tool. These statistics measure the size of the model, the proof obligations generated and discharged by the Rodin prover and those are interactively proved. The complete development of the heart model results in 218(100%) proof obligations, within which 194(89%) are proved automatically by the Rodin tool. The remaining 24(11%) proof obligations are proved interactively using Rodin tool. For the heart model, many proof obligations are generated because of the introduction of the new functional behaviors. To guarantee the correctness of these functional behaviors, we have established various invariants in the incremental refinements. Most of the proofs are interactively discharged in the third refinement of

the heart model. These proofs are quite simple, and have been discharged with the help of simplifying predicates. Few proof obligations are proved interactively in other refinements. The incremental refinement of the heart system helps to achieve a high degree of automatic proof.

8.6 Discussion

This chapter presents a methodology for modelling a biological system, such as the heart, by modelling a biological environment. The main objective of this methodology is to model the heart system and integrate it with the model of a medical device such as a cardiac pacemaker, thereby modelling the closed-loop system to enable certification of the medical system via the certification bodies [Keatley 1999; NITRD 2009] for safe operation. To build a closed-loop model using both environment and device modelling is considered a standard approach to validation, given that designing an environment model is a challenging problem in the real world. Industry has long sought such an approach to validating system models in a biological environment. We have discovered much information via a literature survey and long discussions with experts in cardiology and physiology, and have concluded how best to model the heart system as a cellular-level architecture in an efficient and optimum way. Because of the complexity of the cellular-level calculations (see Sec. 2), previous models have failed to model the heart system.

We have proposed modelling the heart in an abstract way to simulate the desired behaviour of the heart system while avoiding the complexity. More importantly, the heart model is based on logico-mathematical theory. Our primary objective was to model the heart system using only simple logico-mathematical methods. The heart model is an environmental model for medical devices that may improve their development in the early phases. As such, it will contribute only one element of the verification process. Other verification steps will also be required. Medical experts have elaborated every minor detail in an effort to understand the complexity of the biological system, particularly because the heart system is the most complex organ in the body. The proposed approach contains only a main part of the specification of the system behaviour, with the remaining information being hidden. We have spent much time identifying an exact abstract model of the heart system that satisfies medical experts. We have used the EVENT B modelling language to model and verify the system. The ProB model checker was used to verify the correctness of the heart model via animation. Any other formal specification language and model checker could be used to model the heart system based on our proposed methodology.

8.7 Conclusion

This chapter has presented a methodology for producing a mathematical model of a heart based on logico-mathematical theory [Méry 2011f; Méry 2011i]. This model is the first computational model that considers the heart as an electrical conduction system. Given that a cardiac pacemaker interacts with the heart exactly at this level (i.e., electrical impulses), this model is a very promising "environmental model" to be used in parallel with

a pacemaker model to form a closed-loop system. This model therefore has an immediate use in "the grand challenges in formal methods" where an industrial pacemaker specification has been elected as a benchmark. To formalize the heart system, we have used the Event-B modelling language [RODIN 2004; Abrial 2010] to develop the proof-based formal model. Our approach involves formalizing and reasoning about impulse propagation in the whole heart system through the conduction network (see Fig. 8.4(a)). More precisely, we would like to stress the original contribution of our work. We have proposed a method for modelling a human heart based on logico-mathematical theory. The main objectives of this proposed idea are as follows:

- To obtain a certification procedure for providing a higher safety integrity level
- To verify the system in a patient model (in a formal representation)
- To analyse the biological environment (the heart) in a mathematical way
- To analyse the interaction between the heart model and a cardiac pacemaker or ICD.

In summary, we have formalized the known characteristics and physiological behaviour of the heart. The formalization highlights various aspects of the problem, making different assumptions about impulse propagation and establishing different properties related to the CA. We have outlined how an incremental refinement approach to the heart system enables a high degree of automatic proof using the Rodin tool. Our various developments reflect not only many facets of the problem, but also the learning process involved in understanding the problem and its ultimate possible solutions.

The consistency of our specification has been checked through reasoning, and validation experiments were performed using the ProB model checker with respect to safety conditions. As part of our reasoning, we have proved that the initialization of the system is valid, and we have calculated the preconditions for operations. These have been executed to guarantee that our intention to have total operations has been fulfilled. At each stage of the refinement, we have introduced a new behaviour for the system and proved its *consistency* and *refinement checking*. We have introduced more general invariants at the refinement level, showing that the initialization of the whole system is valid. Finally, we have validated the heart system using the ProB model checker as a validation tool and have verified the correctness of the exact behaviour of our heart system with the help of physiology and cardiology experts.

Part II

Case Studies

The following two subsequent chapters present the results of our proposed development life-cycle methodology, and related techniques and tools for developing the industrial-scale case studies. We have selected two case studies that cover two different domains related to the medical and automotive areas for showing the effectiveness of the proposed methodology, and a set of associated techniques and tools. The main objective is to select two different kinds of case studies to show generalization of the proposed approach, which can be applicable for any other domains. A case study related to the medical domain is the cardiac pacemaker, which is a grand challenge in the area of verification and validation, which is proposed by the software quality research laboratory at McMaster University as a pilot project for the Verified Software Initiative [Woodcock 2006; Macedo 2008]. The challenge is characterised by system aspects, including hardware requirements and safety issues. Another interesting case study related to the automotive domain is the adaptive cruise control (ACC), which is based on a hybrid controller, where we have applied our development methodology for implementing the ACC system. In both case studies, we have carried out effectiveness of our proposed approach (see Part-I) in form of formal specification, verification and automatic code generation. Both systems are very complex, ambiguous and highly critical in use. Formal methods based development process ensures successful development of the selected case studies and meets the requirements. It should be noted that, we have proposed the methodology for a critical system development, including safety assessment approach, but we have not applied yet any safety assessment technique in these case studies. It is part of our future work. In this thesis, we have only focused on the development life-cycle methodology and associated techniques and tools of a critical system development using formal techniques.

The last chapter of this section presents a formal development medical protocol, where we have selected the ECG interpretation protocol that covers a wide range of protocol characteristics related to the heart diseases. The Electrocardiogram (ECG or EKG) interpretation is a common technique to trace abnormalities in the heart system and various levels of tracing help to find severe diseases. All kinds of medical guidelines and protocols to differ from each other along several dimensions, which can be referred to the contents of the protocols or to its form. The main objective is to use this case study to model the medical protocols using refinement approach and to discover different kinds of anomalies like ambiguity, inconsistency and incompleteness.

CHAPTER 9

The Cardiac Pacemaker

"It doesn't matter how beautiful your theory is, it doesn't matter how smart you are. If it doesn't agree with experiment, it's wrong."

(Richard Feynman)

Building high quality and zero defects medical software-based devices is a critical task, and formal modeling techniques can effectively help to achieve this target at the certain level. Formal modeling of a high-confidence medical device, such as that is too much error prone in operating, is an international Grand Challenge in the area of Verified Software. Modeling a pacemaker is one of the proposed challenges, and we are considering the complete description of pacemaker's functionalities using an incremental proof-based approach. To assess the effectiveness of our proposed development methodology and associated techniques and tools, we have selected this case study. This chapter presents the development of the cardiac pacemaker using our proposed development life-cycle methodology from requirement analysis to automatic code generation. In this development, we have used formal verification to verify the correctness of the requirements, model checking to verify the correctness of the system behaviors, real-time animator to check the system behaviors according to the domain experts (i.e. medical experts), and finally the code generation tool EB2ALL for generating the codes into several programming languages. The refinement charts are used to handle the complexity of the system, where it helps to organize the code structure according to the different operating modes. Formal models are expressed in the Event-B modeling language, which integrates conditions (called proof obligations) for checking their internal consistency with respect to the invariants and safety properties. The generated proof obligations of models are proved by the Rodin tool and desired behavior of the system is validated by the ProB tool and real-time animator according to the medical experts. Additionally, we also present a closed-loop model of the heart and pacemaker systems.

9.1 Introduction

Development and production of medical device software and systems are common crucial issues [Woodcock 2006] for ensuring safe advances in healthcare. The lack of uniform standard and formalism in the engineering of medical-device software leads many deficiencies in developing relatively low cost trustworthy software under a limit time frame. For

decades, software failures have costed billions of dollars a year [Woodcock 2007]. During this period, softwares have been delivered with restricted warranties of failures and errors, resulting in the well-known software crisis. Due to software crisis, various formalisms and rigorous techniques (VDM, Z, Event-B, Alloy, etc.) have been used in the development process of safety-critical systems. These approaches provide a given level of reliability and confidence to develop the error-free systems. Formal methods and their tools have achieved some usability that could be applied even in industrial-scale applications allowing software developers to provide more meaningful guarantees to their projects.

Tony Hoare suggested Grand challenge for Computing Research [Hoare 2003] to integrate the research community to work together towards a common goal, agreed to be valuable and achievable by a team effort within a predicted timescale. Verification Grand Challenge is one of them. From the Verification, Grand Challenges, many application areas were proposed by the Verified Software Initiative [Hoare 2009]. The pacemaker specification [Scientific 2007; Goldman 1974] has been proposed by the software quality research laboratory at McMaster University as a pilot project for the Verified Software Initiative [Woodcock 2006; Macedo 2008]. The challenge is characterised by system aspects including hardware requirements and safety issues. Such a system demands high integrity to achieve safety requirements. The pacemaker device is highly sensitive, and lots of operating defects are coming day by day.

The contribution of this chapter is to give a complete idea of formal development of the cardiac pacemaker using our proposed framework and a set of techniques and tools (see Part-I). The cardiac pacemaker is a critical system, which is used here to show the usefulness of proposed approaches. Our approach is based on the Event-B modeling language which is supported by the Rodin platform integrating tools for proving models and refinements of models. Here, we present an incremental proof-based development to model and verify such interdisciplinary requirements in Event-B [Cansell 2007; Abrial 2010]. Validation of the system is done by model checker as well as the real- time animator. The model checker, ProB tool [ProB ; Leuschel 2003] is used for validating and analyzing the developed formal specifications. The cardiac pacemaker models must be validated to ensure that they meet requirements of the pacemaker. Hence, validation must be carried out by both formal modeling and domain experts. The real-time animator helps to the medical experts to verify the functional behaviour of the system. If medical experts are not agreed on the system behavior, then the system specification is modified and again verify it. In addition, we have proposed the system integration approach using refinements charts to help a code designer to improve the code structure and code optimization, and the code generation for synthesizing and synchronizing the software codes of the cardiac pacemaker. We have also used our proposed environment model of the heart to specify a closed-loop system of the heart and cardiac pacemaker. Finally, we have used our translator (EB2ALL) [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b; Méry 2011d; Méry 2011a] to generate the source code in multiple languages (C, C++, Java, C#). In the rest of the sections of this chapter, we describe step by step a development of the one-and two electrodes cardiac pacemaker.

9.1.1 Why Model-Checker?

Model checking[Baier 2008] and theorem proving are both applicable in medical device development. This approach requires a model of the system under consideration together with a desired property and systematically checks whether the given model satisfies this property. The basic technique of model checking is a systematic, usually exhaustive, state-space search to check whether the property is satisfied in each state of the model, thereby using effective methods to combat the infamous state-space explosion problem. Using model checking with formal verification for medical device has several benefits:

- To understand the formal verification of any system is not an easy task. A group of non-formal people (doctors, engineering, coder and so on) cannot understand it due to lack of knowledge of formal mathematics. Non-formal people can understand the desirable system behavior through model checker and can give the proper feedback.

- A model-checker is also useful for a model designer to improve the system. A model-checker may provide a counter-example showing under which circumstance the error can be generated. The counter-example provides evidence that the system is faulty and needs to be revised. This allows the user to locate the error and to repair the system before continuing. If no error is found, the user can refine the model description and can restart the verification process.

9.1.2 Related Work for the Cardiac Pacemaker

Macedo et al. [Macedo 2008] have developed a distributed real-time model of a cardiac pacemaker using a formal tool VDM [Bjørner 1978], where they have modeled the subset of pacemaker functionalities. In another pacemaker case study, Manna et al. [Manna 2009] have shown a simple pacemaker implementation. Gomes et al. [Gomes 2009] have presented a formal specification of a cardiac pacemaker using Z modeling language, and they have modelled the sequential model similar to Macedo et al. work [Macedo 2008]. A detailed formalisation of the one- and two electrode pacemaker is represented in [Méry 2011g; Méry 2009; Méry 2010c]. The model has been developed in an incremental way using refinements in the Event-B modelling language. Tuan et al. [Tuan 2010] have proposed a formal model of the pacemaker based on its behaviour including the communication with the external environment. They have designed a real-time model of the pacemaker using timed extensions of CSP and used the model checker Process Analysis Toolkit (PAT) in order to verify the critical properties, such as deadlock freeness and heart rate limits. Recently, Gomes et al. [Gomes 2010] have presented the pacemaker case study by providing a means to execute the model using a translation of Z model into Perfect Developer [Crocker 2003]. They have used the existing tool Perfect Developer [Crocker 2003] to generate an executable code of Z model.

Our models are superior than the sequential model of H.D. Macedo, et al. [Macedo 2008] and Gomes et al [Gomes 2009]. We have added the *threshold*, *hysteresis* and *rate adaptive* bradicardia operating modes in our formal specification. We have developed the

parametric and functional incremental development of bradicardia operating modes. Incremental development is based on refinement approach and at every level of the development, we have proved all the required safety properties (*refinement* and *consistency checking*). Other specifications [Macedo 2008; Gomes 2009] of the pacemaker developed as a one-shot model, means those are not based on the refinement and the correctness of a model is not checked by any model checker, for safely desired behavior of the pacemaker system. We use the formal verification for consistency checking and a model checker tool to check desired behavior of the system is validated by the ProB tool according to the medical experts at each refinement level of the formal development. In this chapter, we have presented formal development of a cardia pacemaker model [Méry 2011g] as well as source code generation from Event-B model to programming languages [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b].

The outline of the remaining chapter is as follows. We give a brief outline of the pacemaker and the heart system in Section 2. Section 3 presents patterns for modeling the cardiac pacemaker. Refinement structure of the cardiac pacemaker is given in Section 4. Section 5 presents development of the cardiac pacemaker using refinement charts. Section 6 and 7 explores stepwise formal development of the one- and two-electrodes cardiac pacemakers. Section 8 presents model validation using the ProB model checker. Section 9 presents a closed-loop formal model for the heart and cardiac pacemaker. Section 10 presents use of the real-time animator for validating the pacemaker models according to the domain experts. Section 11 presents code generation process from formal specifications of the cardiac pacemaker using EB2ALL tool and finally Section 12 concludes the chapter with some discussions.

9.2 Basic Overview of Pacemaker system

The conventional pacemakers serve two major functions, namely *pacing* and *sensing*. The pacemaker actuator is pacing by the delivery of a short, intense electrical pulse into the heart. However, the pacemaker sensor uses the same electrode to detect the intrinsic activity of the heart. So, the pacemaker function of pacing and sensing activities are dependent on the behavior of the heart. The sensing and pacing functions regulate the heart rhythm.

The pacemaker system is a small electronic device that helps the heart to maintain the regular heart beat. In this study, the pacemaker is treated as an embedded system operating in an environment containing the heart. We first review the heart system that interact with the pacemaker (Section 2.1) and then consider elements of the pacemaker system itself (Section 2.2).

9.2.1 The Heart System

The human heart is wondrous in its ability to pump blood to the circulatory system continuously throughout a lifetime. The heart consists of four chambers: right atrial, right ventricle, left atrial and left ventricle, which contract and relax periodically. Atria form one unit and ventricles form another. The heart's mechanical system (the pump) requires at the very least impulses from the electrical system. An electrical stimulus is generated by

the sinus node (see Fig. 9.1), which is a small mass of specialized tissue located in the right atrium of the heart. This electrical stimulus travels down through the conduction pathways and causes the heart's lower chambers to contract and pump out blood. The right and left atrial are stimulated first and contract for a short period of time before the right and left ventricles. Each contraction of the ventricles represents one heartbeat. The atria contract for a fraction of a second before the ventricles, so their blood empties into the ventricles before the ventricles contract.

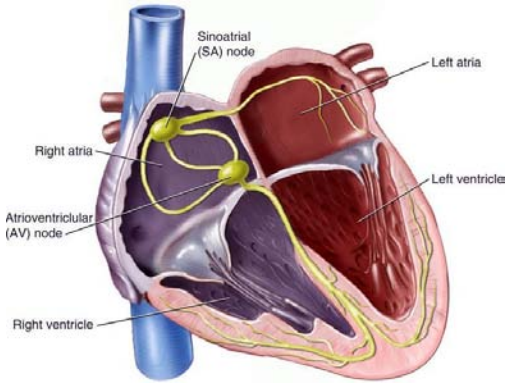


Figure 9.1: Heart or Natural Pacemaker

An artificial pacemaker is implanted to assist the heart in case of an arrhythmias condition to control the heart rate [Malmivuo 1995]. Arrhythmias are due to the cardiac problems producing abnormal heart rhythms. In general, arrhythmias reduce hemodynamic performance, including situations where the heart's natural pacemaker develops an abnormal rate or rhythm or when normal conduction pathways are interrupted, and a different part of the heart takes over control of the rhythm. An arrhythmia can involve an abnormal rhythm increase (tachycardia; $>$ 100 bpm) or decrease (bradycardia; $<$ 60 bpm), or may be characterized by an irregular cardiac rhythm, e.g. due to asynchrony of the cardiac chambers. The irregularity of the heartbeat, called bradycardia and tachycardia. The bradycardia indicates that the heart rate falls below the expected level while in tachycardia indicates that the heart rate goes above the expected level of the heart rate. An artificial pacemaker can restore synchrony between the atria and ventricles. In an artificial pacemaker system, the firmware controls the hardware such that an adequate heart rate is maintained, which is necessary either because the heart's natural pacemaker is insufficiently fast or slow or there is a block in the heart's electrical conduction system [Barold 2004; Ellenbogen 2005; Hesselson 2003; Lee 2006b; Love 2006; Malmivuo 1995]. Beats per minute (bpm) is a basic unit to measure the rate of heart activity.

9.2.2 The Pacemaker System

The basic elements of the pacemaker system [Barold 2004; Ellenbogen 2005] are:

1. **Leads:** One or more flexible coiled metal wire normally two, that transmits electrical signals between the heart and the pacemaker. Each pacemaker lead is classified

Category	Chambers Paced	Chambers Sensed	Response to Sensing	Rate Modulation
Letters	**O**-None **A**-Atrium **V**-Ventricle **D**-Dual(A+V)	**O**-None **A**-Atrium **V**-Ventricle **D**-Dual(A+V)	**O**-None **T**-Triggered **I**-Inhibited **D**-Dual(T+I)	**R**-Rate Modulation

Table 9.1: Bradycardia operating modes of pacemaker system

by its configuration: either one ("unipolar") or two ("bipolar") separated points of electrical contact with the heart.

2. **The Pacemaker Generator:** The pacemaker is both the power source and the brain of the pacing and sensing systems. It contains an implanted battery and controller as an electronic circuitry.

3. **Device Controller-Monitor (DCM) or Programmer:** An external unit that interacts with the pacemaker device using a wireless connection. It consists of a hardware platform and the pacemaker application software.

4. **Accelerometer:** It is an electromechanical device inside the pacemaker that measures the body motion or acceleration of motion of a body in order to allow modulated pacing.

In the pacemaker, an electrode is attached to the right atrium or the right ventricle. It has several operational modes that regulate the heart functioning. The specification document [Scientific 2007] describes all possible operating modes that are controlled by the different programmable parameters of the pacemaker. All the programmable parameters are related to the real-time and action-reaction constraints, that are used to regulate the heart rate.

9.2.3 Bradycardia Operating Modes

In order to understand the *language* of pacing, it is necessary to comprehend the coding system that produced by a combined working party of the North American Society of Pacing and Electrophysiology (NASPE) and the British Pacing and Electrophysiology Group (BPEG) known as NASPE/BPEG generic (NBG) pacemaker code [Writing Committee 2008]. This is a code of five letters of which the first three are most often used. The code provides a description of the pacemaker pacing and sensing functions. The sequence is referred to as *bradycardia operating modes* (see Table-9.1). In practice, only the first three or four-letter positions are commonly used to describe bradycardia pacing functions. The first letter of the code indicates which chambers are being paced; the second letter indicates which chambers are being sensed; the third letter of the code indicates the response to sensing and the final letter, which is optional indicates the presence of rate modulation in response to the physical activity measured by the accelerometer. An accelerometer is an additional sensor in the pacemaker system that detects a physiological result of exercise or emotion,

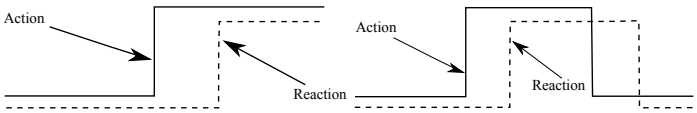


Figure 9.2: Action-Reaction Patterns

and increases the pacemaker rate on the basis of a programmable algorithm. "X" is a wild-card used to denote any letter (i.e. "O", "A", "V" or "D"). $Triggered\ (T)$ refers to deliver a pacing stimulus and $Inhibited\ (I)$ refers to an inhibition from further pacing after sensing of an intrinsic activity from the heart chambers.

9.3 Event-B Patterns for Modeling Cardiac Pacemaker

Considering design patterns [Gamma 1994], the purpose is to capture structures and to make decisions within a design that are common to similar modeling and analysis tasks. They can be re-applied when undertaking similar tasks in order to reduce the duplication of effort. The design pattern approach is the possibility to reuse solutions from earlier developments in the current project. This will lead to a *correct refinement* in the chain of models, without producing proof obligations. Since the correctness (i.e proof obligations are proved) of the pattern has been proved during its development, nothing is to be proved again when using this pattern.

Pacemaker systems are characterized by their functions, which can be expressed by analyzing *action-reaction* and *real-time* patterns. Sequences of inputs are recognized, and outputs can be emitted in response within a fixed time interval. So, the most common elements in the pacemaker system are bounded time interval for every action, reaction and action-reaction pair. The action-reaction within a time limit can be viewed as an abstraction of the pacemaker system. We recognize the following two design patterns when modeling this kind of system according to the relationship between the action and corresponding reaction.

9.3.1 Action-Reaction Pattern

Under action-reaction chapter [Abrial 2010] two basic types of design patterns are,

Action and Weak Reaction: Once an action emits, a reaction should start in response. For a quick instance, if an action stops, the reaction should follow. Sometimes reaction does not change immediately according to the action because the action moves too quickly (the continuance of an action is too short, or the interval between actions is too short). This is known as a pattern of action and weak reaction.

Action and Strong Reaction: When every reaction follows every action and there is proper synchronization between action and corresponding reaction then this pattern is known as action and strong reaction.

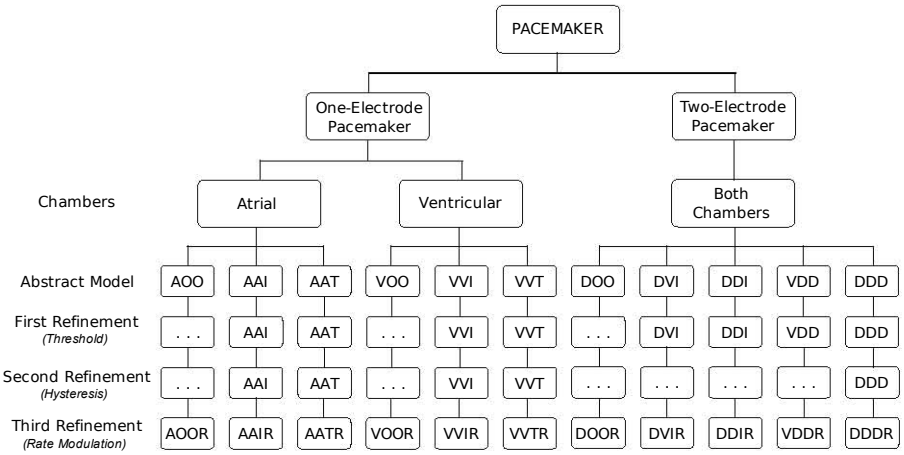


Figure 9.3: Refinement structure of bradycardia operating modes of the pacemaker

9.3.2 Time-based Pattern

The action-reaction events of a pacemaker system are based on the time constraint pattern in IEEE 1394 proposed by Cansell et. al and on the 2-Slots Simpson Algorithm case studies [Cansell 2006; Rehm 2010]. This time pattern is fully based on a timed automaton. The timed automaton is a finite state machine that is useful to model the components of real-time systems. In a model, timed automata interacts with each other and defines a timed transition system. Besides ordinary action transitions that can represent input, output and internal actions. A timed transition system has time progress transitions. Such time progress transitions result in synchronous progress of all clock variables in the model. Here, we apply the time pattern in modeling to synchronize the sensing and pacing stimulus functions of the pacemaker system in continuous progressive time constraint. In the model, events are controlled under time constraints, which means action of any event activates only when time constraint satisfies on a specific time. The time progress is also an event, so there is no modification of the underlying Event-B language. It is only a modeling technique instead of a specialized formal system. The timed variable is in $\mathbb{N}$ *(natural numbers)*, but time constraint can be written in terms involving unknown constants or expressions between different times. Finally, the timed event observations can be constrained by other events, which determine future activations.

9.4 Refinement Structure of a Cardiac Pacemaker

We present a block diagram (see Fig. 9.3) of a hierarchical tree structure of the possible bradycardia operating modes for a pacemaker. The hierarchical tree structure depicts a stepwise refinement from abstract to concrete models of the formal development for a pacemaker. Each level of refinement introduces new features of a pacemaker as functional and parametric requirements.

The root node indicates a cardiac pacemaker system. The next two branches show two classes of pacemaker: one-electrode pacemaker and two-electrode pacemaker. The one-electrode pacemaker branch is divided into two parts to indicate different chambers of the heart: atrium and ventricular. Atrium and ventricular are the right atrium and the right ventricular. The atrium chamber uses the three operating modes; AOO, AAI and AAT (see Table-9.1). Similarly, the ventricular chamber uses three operating modes: VOO, VVI and VVT (see Table-9.1). In the part of two-electrode pacemaker, there is only one branch for both chambers. Both chambers of the heart use the five operating modes: DOO, DVI, DDI, VDD and DDD. In the abstract model, we introduce the bradycardia operating modes of the pacemaker abstractly with required properties. From first refinement to the last refinement, there is only one branch in every operating mode of the pacemaker. In one and two-electrode pacemaker, there are three refinements: first *threshold* refinement; second *hysteresis* refinement; and third *rate adaptive or rate modulation* refinement. The subsequent refinement models introduce new features or functional requirements for the resulting system. The triple dots (...) in the hierarchical tree represents that there is no refinement at that level, in particular, operating modes (AOO, VOO, DOO, etc.). In the last refinement level, we have achieved the additional rate adaptive operating modes (i.e.AOOR, AAIR, VVTR, DOOR, DDDR, etc). These operating modes are different from the previous levels of operating modes. This refinement structure is very helpful to model the functional requirements of the cardiac pacemaker.

9.5 Development of the Cardiac Pacemaker using Refinement Chart

A formal specification serves as the central role of the development and evolution process. A refinement typically embodies a well-defined unit of programming knowledge. Fig. 9.4 and Fig. 9.5 present the diagrams of the most abstract modal system for the one and two-electrode pacemaker system (A) and the resulting models of three successive refinement steps (B to D). The diagrams use a visual notation to represent the bradycardia operating modes of the pacemaker under functional and parametric requirements. An operating mode is represented by a box with a mode name; an operating mode transition is an arrow connecting two operating modes. The direction of an arrow indicates the previous and next operating modes in a transition. Refinement is expressed by nesting boxes [Méry 2011e].

A refined diagram of an abstract mode is equivalent to a concrete mode. These block wise refinements are similar to the hierarchical tree structure (see Fig. 9.3) of the bradycardia operating modes of the pacemaker. The nesting boxes in one- and two-electrode pacemakers (Fig. 9.4, Fig. 9.5) represent equivalent to every refinement level of the hierarchical tree structure (see Fig. 9.3). Special initiating and terminating modes are *on* and *off* respectively of the pacemaker, which are omitted here in the refinement chart block diagram. At the most abstract level, we introduce *pacing* activity into single and both heart chambers. In Fig. 9.4(A) and Fig. 9.5(A), *pacing* is represented by transitions *Pace ON* and *Pace OFF* for single chamber or both chambers. It is the basic transitions for all bradycardia operating modes. During a pacing cycle, it is ensured that no other pacing activity

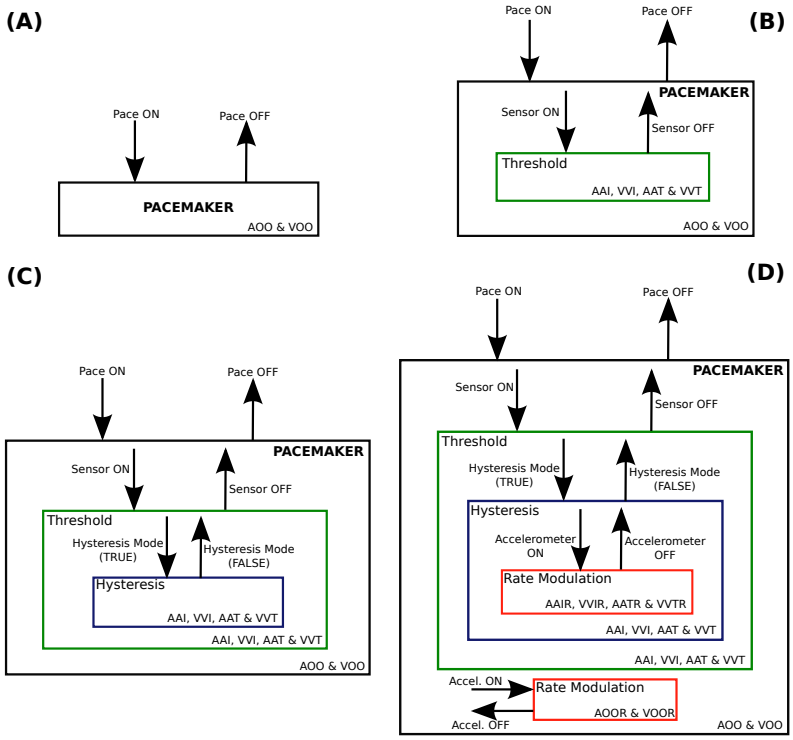


Figure 9.4: Refinements of one-electrode pacemaker using the refinement chart

has occurred. The model includes: the state of *pacing* (on/off) modelled by a boolean flag *Pacemaker_Actuator*; the current time control of the pacemaker, is stored in variable *tic*; a safe pacing interval is *Pace_Int* in which the pacemaker should not be paced.

In the next refinement (Fig. 9.4(B), Fig. 9.5(B)) step *pacing* is refined by *sensing*, corresponding to the activity of the heart, when sensing period is not under refractory period (RF[1]). In the first refinement of two-electrode pacemaker, sensors are introduced in both chambers. In Fig. 9.4(B) of one-electrode, sensing is represented by transitions *Sensor ON* and *Sensor OFF*, while in Fig. 9.5(B) of two-electrode, sensing is represented by transitions *Sensor ON Atria*, *Sensor ON Ventricle*, *Sensor OFF Atria* and *Sensor OFF ventricle*. This refinement introduces: the state of pacemaker sensor (on/off), is modelled by a boolean flag *Pacemaker_Sensor*. The pacemaker's actuator and sensor are synchronizing to each other under the real-time constraints. The block diagrams (Fig. 9.4(B), Fig. 9.5(B)) represent the *threshold* refinement, that is a measuring unit which measures a stimulation threshold voltage value of the heart and a pulse generator for delivering stimulation pulses to the heart. The pacemaker's sensor starts sensing after the refractory period (RF) but pacemaker's actuator delivers a pacing stimulus when sensing value is greater than an equal to the standard threshold constant. Sensor-related transitions are available in all operating modes except AOO, VOO and DOO modes.

Third refinement step (Fig. 9.4(C), Fig. 9.5(C)) introduces different operating strategies under *hysteresis* interval: if the *hysteresis* mode is TRUE, then the pacemaker paces at a

[1]RF : Atria Refractory Period (ARP) or Ventricular Refractory Period (VRP)

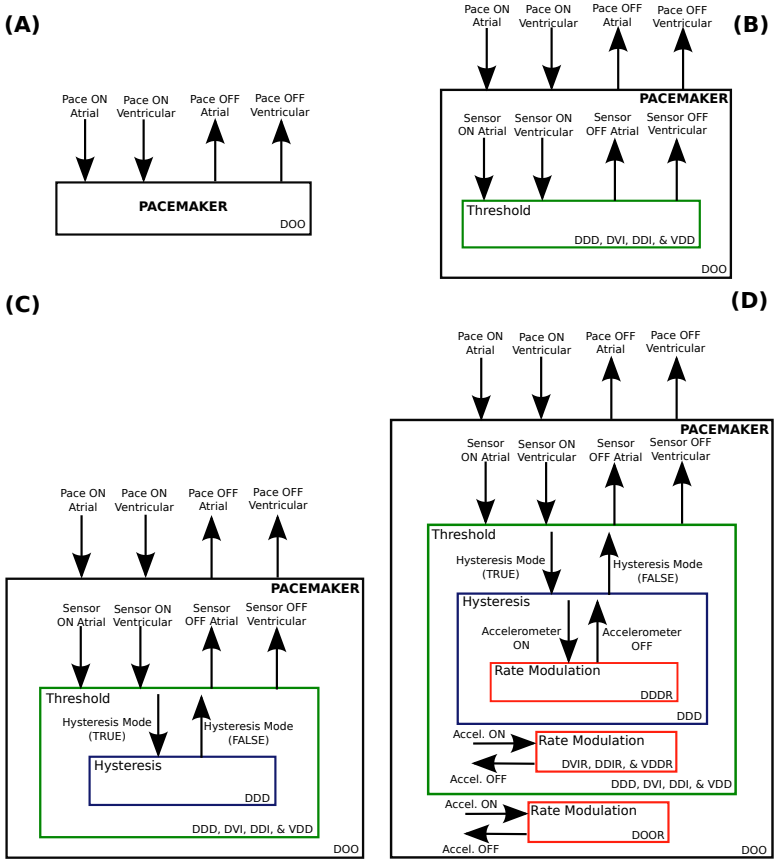


Figure 9.5: Refinements of two-electrode pacemaker using the refinement chart

faster rate than the sensing rate to provide consistent pacing in one chamber(atria or ventricle) or both chambers (atria and ventricle), or prevents constant pacing in one chamber (atria or ventricle) or both chambers (atria and ventricle). In case of FALSE state of *hysteresis* mode of the pacemaker's sensor and actuator are working in normal state or does not try to maintain the consistent pacing. Hysteresis mode is represented by transitions *Hysteresis Mode TRUE* and *Hysteresis Mode FALSE*. The main objective of hysteresis is to allow the patient to have his or her own underlying rhythm as much as possible. The *hysteresis* operating mode is available in AAI,AAT,VVI,VVT and DDD modes.

According to the last refinement step (Fig. 9.4(D), Fig. 9.5(D)), it introduces the rate adapting pacing technique in the bradycardia operating modes of the pacemaker. The rate modulation mode is represented by transitions *Accel. ON* and *Accel. OFF*. The rate modulation operating modes are available in all pacemaker operating modes which are given under multiple refinements. The pacemaker uses the accelerometer sensors to sense the physiologic need of the heart and increase and decrease the pacing rate. The amount of rate increases is determined by the pacemaker based on maximum exertion is performed by the patient. This increased pacing rate is sometimes referred to as the "sensor indicated rate". When exertion has stopped the pacemaker will progressively decrease the pacing rate down to the lower rate.

The next section presents only selected parts of our formalization and omit proof details. For instance, we have omitted the specification of refinement of every event from all operating modes. Only newly introduced event specifications are given in all refinements. To find more detailed information see the published papers and research reports [Méry 2009; Méry 2010c; Méry 2011g].

9.6 Formal development of the one-electrode cardiac pacemaker

9.6.1 Context and Initial Model

9.6.1.1 Abstraction of AOO and VOO modes:

We begin by defining the Event-B context. The context uses sets and constants to define axioms and theorems. Axioms and theorems represent the logical theory of a system. The logical theory is the static properties and properties of the system. In the context, we define constants LRL and URL that relate to the lower rate limit (minimum number of pace pulses delivered per minute by pacemaker) and upper rate limit (how fast the pacemaker will allow the heart to be paced). These constants are extracted from the pacemaker specification document [Scientific 2007]. The lower rate limit (LRL) must be between 30 and 175 pulse per minute (ppm) and upper rate limit (URL) must be between 50 and 175 pulse per minute (ppm).

The two new constants URI and LRI represent the corresponding upper rate interval and lower rate interval, respectively. The pacemaker (or pacing) rate is programmed in milliseconds. To convert a heart rate from beats per minute (bpm) to milliseconds, 60,000 is divided by the heart rate. For example, a heart rate of 70 bpm equals 857 milliseconds. In the context, the axioms ($axm3$ and $axm4$) represent the upper rate interval (URI) and lower rate interval (LRI). Additionally, we define an enumerated set $status$ of an electrode as ON or OFF states. Finally, we introduce some basic initial properties using defined constants of the system by axioms($axm6$ and $axm7$).

$$
\begin{array}{l}
axm1 : LRL \in 30 \mathinner{..} 175 \\
axm2 : URL \in 50 \mathinner{..} 175 \\
axm3 : URI \in \mathbb{N}_1 \ \wedge URI = 60000/URL \\
axm4 : LRI \in \mathbb{N}_1 \ \wedge LRI = 60000/LRL \\
axm5 : status = \{ON, OFF\} \\
axm6 : LRL < URL \\
axm7 : URI < LRI
\end{array}
$$

In the one-electrode pacemaker system, the pacemaker delivers a pacing stimulus in the atria or the ventricular chamber. In our initial model, we formalize the functional behaviors of the pacemaker system, where a new variable *Pace_Actu* is a pacemaker's actuator, represents the presence or absence of pulse. A variable *Pace_Int* is an interval between two paces, which is initialized by the system before starting the pacing. A variable sp represents the current *clock counter* and a variable *last_sp* represents the last interval (in ms.) between two paces. An invariant $(inv5)$ states that the clock counter sp should

be less than or equal to the lower rate interval (LRI). The next two invariants ($inv6, inv7$) introduce two variables *Pace_Int_flag* and *last_sp*. The variable *Pace_Int_flag* is defined as a boolean type to represent changing state of the pacing interval (*Pace_Int*), and the variable *last_sp* is used to store an interval between last two heart beats or paces. An invariant ($inv8$) represents safety properties: the pacemaker delivers a pacing stimulus into the heart chamber between upper rate interval (URI) and lower rate interval (LRI). Similarly, the next invariant ($inv9$) represents the states of the pacemaker's actuator under the heart environment as the safety properties, and state that it is never activated between two heart beats, means pacemaker's actuator is OFF during pace interval *Pace_Int*, and pace changing flag is FALSE. The last invariant ($inv10$) states that if pace changing flag is FALSE and the pacemaker's actuator is ON, then the current clock counter sp is equal to the pace interval *Pace_Int*.

$inv1 : Pace\_Actu \in status$
$inv2 : Pace\_Int \in URI \mathinner{..} LRI$
$inv3 : sp \in 1 \mathinner{..} \mathbb{N}$
$inv4 : last\_sp \in \mathbb{N}$
$inv5 : sp \leq LRI$
$inv6 : Pace\_Int\_flag \in BOOL$
$inv7 : last\_sp \in \mathbb{N}$
$inv8 : last\_sp \geq URI \ \wedge last\_sp \leq LRI$
$inv9 : Pace\_Int\_flag = FALSE \wedge sp < Pace\_Int \quad \Rightarrow \quad Pace\_Actu = OFF$
$inv10 : Pace\_Int\_flag = FALSE \wedge Pace\_Actu = ON \Rightarrow sp = Pace\_Int$

In our abstract specification, there are four events *Pace_ON* to start pacing, *Pace_OFF* to stop pacing, tic to increment the current clock counter under the time constraints and *Change_Pace_Int* to update the pace interval. The events *Pace_ON* and *Pace_OFF* start and stop the pulse stimulating into the heart chamber, respectively. The guards of these events synchronize the ON and OFF states of the pacemaker system under the time constraints. The action part of event *Pace_ON* sets the ON state of the pacemaker's actuator and assigns the value of current clock counter sp to a new variable *last_sp*. Similarly, an action part of the event *Pace_OFF* sets OFF state of the pacemaker's actuator and resets the current clock counter variable sp to 1.

EVENT Pace_ON
WHEN
grd1 : $Pace\_Actu = OFF$
grd2 : $sp = Pace\_Int$
THEN
act1 : $Pace\_Actu := ON$
act2 : $last\_sp := sp$
END

EVENT Pace_OFF
WHEN
grd1 : $Pace\_Actu = ON$
grd2 : $sp = Pace\_Int$
THEN
act1 : $Pace\_Actu := OFF$
act2 : $sp := 1$
END

EVENT tic
WHEN
grd1 : $sp < Pace\_Int$
THEN
act1 : $sp := sp + 1$
END

The pacing and sensing events update a state every millisecond. We model this increment by the event tic, that increments time in 1 ms. The event tic progressively increases the current clock counter sp under pre-defined pace interval *Pace_Int*. The predicate in guard $(grd1)$ of the event tic represents an upper bound time limit.

EVENT Change_Pace_Int
WHEN
grd1 : $Pace\_Int\_flag = TRUE$
THEN
act1 : $Pace\_Int :\in URI \mathinner{..} LRI$
END

A new event *Change_Pace_Int* is introduced to update the current value of the pace interval. This event is defined abstractly, which is used in further refinement levels to modify the pace interval according to the introduction of different operating modes like hysteresis and rate modulation. This event represents that when the pace changing flag (*Pace_Int_flag*) is TRUE then the pace interval (*Pace_Int*) can be chosen from URI to LRI range.

9.6.1.2 Abstraction of AAI and VVI modes:

In the abstract model of AAI and VVI modes, all the constants, variables and events are common as the abstract model of AOO and VOO modes. In this section, we introduce only extra added new constants, variables and events. We introduce a new constant refractory period RF (*Refractory Period of Atria(ARP) or Ventricular(VRP)*) that represents a period during which pacemaker timing in the heart chamber is not affected by events. Two new

axioms ($axm1$, $axm2$) are introduced in this abstract level. The first axiom ($axm1$) represents a refractory period as a constant and the second axiom ($axm2$) represents a static property.

axm1 : $RF \in 150 \mathinner{..} 500$
axm2 : $URI > RF$

A new variable *Pace_Sensor* as a pacemaker's sensor, is defined as an enumerated type that represents the presence or absence of sensing pulse from the heart chamber and a variable *last_ss* represents the last interval (in ms.) between two sensed pulses. Some new invariants are added here that are common to all other operating modes (except AOO and VOO) of the pacemaker system. An invariant $(inv3)$ states that the interval between two sensed pulses is greater than or equal to the refractory period RF and less than or equal to the pace interval (*Pace_Int*). Invariants ($inv4$, $inv5$) state that the pacemaker's sensor and actuator are always in OFF state during the refractory period RF. These are the essential safety properties for the refractory period during which the pacemaker timing must not to be affected by any events that occur. The last invariant ($inv6$) states that when pace changing flag (*Pace_Int_flag*) is FALSE, current clock counter (sp) is greater than the refractory period (RF) and less than the pacing interval (*Pace_Int*), then the pacemaker's sensor should be ON, means the pacemaker's sensor is ON and continuously sensing the intrinsic activities from the heart chamber within an alert period *(Pace_Int-RF)*.

inv1 : $Pace\_Sensor \in status$
inv2 : $last\_ss \in \mathbb{N}$
inv3 : $last\_ss \geq RF \wedge last\_ss \leq LRI$
inv4 : $sp < RF \Rightarrow Pace\_Sensor = OFF$
inv5 : $sp < RF \Rightarrow Pace\_Actu = OFF$
inv6 : $Pace\_Int\_flag = FALSE \wedge sp > RF \wedge sp \leq Pace\_Int \Rightarrow$
$Pace\_Sensor = ON$

We introduce extra events *Pace_OFF_with_Sensor* and *Sense_ON* in the abstraction model of the AAI and VVI modes. The guards of event *Pace_OFF_with_Sensor* state that, when the pacemaker's actuator is OFF, pacemaker's sensor is ON and current clock counter sp is greater than or equal to the refractory period RF then it stores the value of the current clock counter sp to the new variable *last_ss*, resets the current clock counter sp to 1 and sets OFF state of the pacemaker's sensor. It means that during the alert period *(Pace_Int - RF)*, the pacemaker inhibits a pacing stimulus and resets the current clock counter whenever it senses an intrinsic activity from the heart chamber.

EVENT Pace_OFF_with_Sensor
WHEN
grd1 : $Pace\_Actu = OFF$
grd2 : $Pace\_Sensor = ON$
grd3 : $sp \geq RF$
THEN
act1 : $last\_ss := sp$
act2 : $sp := 1$
act3 : $Pace\_Sensor := OFF$
END

The event *Sense_ON* starts the sensing process of pacemaker's sensor when the sensor is OFF and the current clock counter sp is greater than or equal to the refractory period RF and lower than the pace interval *Pace_Int*. We have added some new guards in the events *Pace_OFF* of AAI and VVI operating modes to control the sensor under the current clock counter sp.

EVENT Sense_ON
WHEN
grd1 : $Pace\_Sensor = OFF$
grd2 : $sp \geq RF$
grd3 : $sp < Pace\_Int$
THEN
act1 : $Pace\_Actu := ON$
END

We have added more real time constraints in the guard $(grd1)$ of the event tic in all operating modes that control the progressive increment of the current clock counter sp as follows:

grd1 : $(sp < RF \wedge Pace\_Sensor = OFF \wedge$
$Pace\_Actu = OFF)$
$\vee$
$(sp \geq RF \wedge sp < Pace\_Int \wedge$
$Pace\_Sensor = ON \wedge Pace\_Actu = OFF)$

9.6.1.3 Abstraction of AAT and VVT modes:

The abstract model of AAT and VVT modes are similar to AAI and VVI modes. We introduce a new event *Pace_ON_with_Sensor* in place of event *Pace_OFF_with_Sensor* in this abstract model. The guards of this event state, when the pacemaker's actuator is OFF, pacemaker's sensor is ON and current clock counter sp is greater than or equal to the refractory period RF and less than or equal to pace interval *Pace_Int* then it stores the value of current clock counter sp to the variable *last_ss* and pacemaker's actuator sets ON under the sensing process. During the alert period *(Pace_Int - RF)*, the pacemaker delivers a pacing stimulus whenever it senses an intrinsic activity from the heart chamber.

EVENT Pace_ON_with_Sensor
WHEN
grd1 : $Pace\_Actu = OFF$
grd2 : $Pace\_Sensor = ON$
grd3 : $sp \geq RF \ \wedge \ sp \leq Pace\_Int$
THEN
act1 : $Pace\_Actu := ON$
act2 : $last\_ss := sp$
END

9.6.2 First refinement: Threshold

The pacemaker control unit delivers stimulation to the heart chamber, on the basis of measured value under the safety margin. We define a new constant THR as $THR \in \mathbb{N}_1$ to hold standard threshold value, and we use the constant nominal threshold value for modeling that is different for the atria and the ventricular chambers.

The pacemaker's sensor starts sensing after the refractory period RF but pacemaker's actuator delivers a pacing stimulus when sensing value is greater than or equal to the standard threshold constant THR[2]. The first invariant is introduced in operating modes (AAI, VVI) that states that the pacemaker's actuator is OFF, when the pace changing flag (*Pace_Int_flag*) is FALSE, the pacemaker's sensor is ON; obtained sensor value is greater than or equal to the standard threshold value, the current clock counter sp is within the alert period *(Pace_Int-RF)* and state of threshold *thr_val_state* is TRUE. Similarly, the second invariant is introduced in the operating modes (AAT, VVT) that states that the pacemaker's actuator is ON, when the pace changing flag (*Pace_Int_flag*) is the FALSE, the pacemaker's sensor is ON, obtained sensor value is greater than or equal to the standard threshold constant THR, the current clock counter sp within the alert period *(Pace_Int-RF)* and the state of threshold *thr_val_state* is TRUE.

inv1 : $Pace\_Int\_flag = FALSE \wedge Pace\_Sensor = ON \wedge thr \geq THR \wedge$
$sp > RF \wedge sp < Pace\_Int \wedge$
$thr\_val\_state = TRUE \Rightarrow$
$Pace\_Actu = OFF$

inv2 : $Pace\_Int\_flag = FALSE \wedge Pace\_Sensor = ON \wedge thr \geq THR \wedge$
$sp > RF \wedge sp < Pace\_Int \wedge$
$thr\_val\_state = TRUE \Rightarrow$
$Pace\_Actu = ON$

A new event *Thr_value* is introduced in all operating modes (AAI,AAT,VVI and VVT), which obtains a measured value of the heart activities using the pacemaker's sensor. Guards of this event state that when the pacemaker's sensor is ON, the current clock counter sp is within the alert period *(Pace_Int-RF)* and state of threshold value *thr_val_state* is TRUE

[2]Standard threshold constant values of atria and ventricular chambers are different.

then the sensed value th is assigned to the threshold variable thr and state of threshold variable *thr_val_state* sets FALSE.

EVENT Thr_value
ANY
th
WHERE
grd1 : $Pace\_Sensor = ON$
grd2 : $sp \geq RF \ \wedge \ sp < Pace\_Int$
grd3 : $thr\_val\_state = TRUE$
grd4 : $th \in \mathbb{N}$
THEN
act1 : $thr := th$
act2 : $thr\_val\_state := FALSE$
END

In this refinement, we have added a new guard $thr \geq THR$ in events *(Pace_OFF_-with_Sensor* and *Pace_ON_with_Sensor)* and modified the guard $(grd1)$ of the event (tic) to synchronize the pacing-sensing events with a new threshold functional behavior under the real-time constraints.

grd1 : $(sp < RF \wedge Pace\_Sensor = OFF \wedge$
$Pace\_Actu = OFF)$
$\vee$
$(sp \geq RF \wedge sp < Pace\_Int \wedge$
$Pace\_Sensor = ON \wedge Pace\_Actu = OFF \wedge$
$thr < THR \wedge thr\_val\_state = FALSE)$

9.6.3 Second refinement: Hysteresis

Hysteresis is a programmed feature whereby the pacemaker paces at a faster rate than the sensing rate. For example, pacing at 80 pulses a minute with a hysteresis rate of 55 means that the pacemaker will be inhibited at all rates down to 55 beats per minute, having been activated at a rate below 55, the pacemaker then switches on and paces at 80 pulses a minute [Malmivuo 1995; Hesselson 2003]. The application of the hysteresis interval provides consistent pacing of the atrial or ventricle, or prevents constant pacing of the atrial or ventricle. The main purpose of hysteresis is to allow the patient to have his or her own underlying rhythm as much as possible. Two new variables (*Hyt_Pace_Int_flag*, *HYT_-State*) are introduced to define functional properties of the hysteresis operating modes. Both variables are defined as boolean types. The hysteresis state *HYT_State* is used to set the hysteresis functional parameter as TRUE or FALSE, to apply the hysteresis operating modes.

inv1 : $Hyt\_Pace\_Int\_flag \in BOOL$
inv2 : $HYT\_State \in BOOL$

A new event *Hyt_Pace_Updating* is introduced to implement the functional properties of the hysteresis operating modes. In the hysteresis operating modes, the pacemaker trying to maintain own heart rhythm as much as possible. Hence, this event can change the pacing interval and set pacing length longer than existing, which changes the pacing length of the cardiac pacemaker. This event is only used for updating the pacing interval (*Pace_Int*). Guards of this event state that pace changing flag (*Pace_Int_flag*) is TRUE, hysteresis pacing flag (*Hyt_Pace_Int_flag*) is TRUE and hysteresis pace interval (*Hyt_Pace_Int*) should be lied between pace interval (*Pace_Int*) and lower rate interval (LRI). Actions of this event state that a new hysteresis pace interval (*Hyt_Pace_Int*) updates the pace interval *Pace_Int*, hysteresis pacing flag (*Hyt_Pace_Int_flag*) sets to FALSE and hysteresis state (*HYT_State*) sets to TRUE.

EVENT Hyt_Pace_Updating Refines Change_Pace_Int
ANY
Hyt_Pace_Int
WHERE
grd1 : $Pace\_Int\_flag = TRUE$
grd2 : $Hyt\_Pace\_Int\_flag = TRUE$
grd3 : $Hyt\_Pace\_Int \in Pace\_Int \mathinner{..} LRI$
THEN
act1 : $Pace\_Int := Hyt\_Pace\_Int$
act2 : $Hyt\_Pace\_Int\_flag := FALSE$
act3 : $HYT\_State := TRUE$
END

9.6.4 Third refinement: Rate Modulation

Rate modulation term is used to describe the capacity of a pacing system to respond to physiologic needs by increasing and decreasing pacing rate. The rate modulation mode of the pacemaker can progressively pace faster than the lower rate, but no more than the upper sensor rate limit, when it determines that heart rate needs to increase. This typically occurs with exercise in patients who cannot increase their own heart rate. The amount of rate increases is determined by the pacemaker based on maximum exertion is performed by the patient. This increased pacing rate is sometimes referred to as the "sensor indicated rate". When exertion has stopped the pacemaker will progressively decrease the paced rate down to the lower rate.

In this last refinement, we introduce the rate modulation function and found some new operating modes (AOOR,VOOR,AAIR,VVIR,AATR and VVTR) of the pacemaker system. For modeling the rate modulation, we introduce some new constants maximum sensor rate MSR as $MSR \in 50..175$ and *acc_thr* as $acc\_thr \in \mathbb{N}_1$ using axioms ($axm1, axm2$). The maximum sensor rate (MSR) is the maximum pacing rate allowed as a result of sensor control, and it must be between 50 and 175 pulse per minute (ppm). The constant *acc_thr* represents the activity threshold. Axiom ($axm3$) represents a static property for the rate modulation operating modes.

axm1 : $MSR \in 50 \mathinner{..} 175$
axm2 : $acc\_thr \in \mathbb{N}_1$
axm3 : $MSR = URL$

Two new variables *acler_sensed* and *acler_sensed_flag* are defined as to store a measured value from the accelerometer and boolean stats of the accelerometer sensor. Boolean state of the accelerometer sensor is used to synchronize with other functionalities of the system. The accelerometer is used to measure the physical activities of the body in the pacemaker system. Two new invariants $(inv3, inv4)$ provide the safety margin and state that the heart rate never falls below the lower rate limit (LRL) and never exceeds the maximum sensor rate (MSR) limit.

inv1 : $acler\_sensed \in \mathbb{N}$
inv2 : $acler\_sensed\_flag \in BOOL$
inv3 : $HYT\_State = FALSE \wedge acler\_sensed < acc\_thr \wedge$
$acler\_sensed\_flag = TRUE \Rightarrow Pace\_Int = 60000/LRL$
inv4 : $HYT\_State = FALSE \wedge acler\_sensed =\geq acc\_thr \wedge$
$acler\_sensed\_flag = TRUE \Rightarrow Pace\_Int = 60000/MSR$

In this final refinement, we introduce two new events *Increase_Interval* and *Decrease_Interval*, which are a refinement of the event *Change_Pace_Int*. These new events are used to control the pacing rate of the one-electrode pacemaker in the rate modulating operating modes. The new events *Increase_Interval* and *Decrease_Interval* control the value of the pace interval variable *Pace_Int*, whenever a measured value (*acler_sensed*) from the accelerometer sensor goes higher or lower than the activity threshold *acc_thr*.

EVENT Increase_Interval Refines Change_Pace_Int
WHEN
grd1 : $Pace\_Int\_flag = TRUE$
grd1 : $acler\_sensed \geq threshold$
grd1 : $HYT\_State = FALSE$
THEN
act1 : $Pace\_Int := 60000/MSR$
act1 : $acler\_sensed\_flag := TRUE$
END

EVENT Decrease_Interval Refines Change_Pace_Int
WHEN
grd1 : $Pace\_Int\_flag = TRUE$
grd1 : $acler\_sensed < threshold$
grd1 : $HYT\_State = FALSE$
THEN
act1 : $Pace\_Int := 60000/LRL$
act1 : $acler\_sensed\_flag := TRUE$
END

A new event (*Acler_sensed*) is defined as to simulate the behaviour of accelerometer sensor. This event is continued sensing the motion of the body to increase or decrease the length of the pace interval (*Pace_Int*). In this event, the guards state that the accelerometer sensor flag is TRUE and hysteresis state is FALSE. A new variable *acl_sen* is used to store the current sensing value. Actions of this event state that local variable *acl_sen* updates accelerometer sensor (*acler_sensed*) and accelerometer sensor flag (*acler_sensed_flag*) sets FALSE.

EVENT Acler_sensed
ANY
acl_sen
WHERE
grd1 : $acl\_sen \in \mathbb{N}$
grd1 : $acler\_sensed\_flag = TRUE$
grd1 : $HYT\_State = FALSE$
THEN
act1 : $acler\_sensed := acl\_sen$
act1 : $acler\_sensed\_flag := FALSE$
END

In the next section, we explore the formal model of the two-electrode pacemaker system using incremental refinements.

9.7 Formal Development of the Two-Electrode Cardiac Pacemaker

9.7.1 Context and Initial Model

In this section, we describe the formal development of initial modes of the two-electrode pacemaker system using the basic notion of action-reaction and real-time constraints, to focus on pacing and sensing activities of the pacemaker's actuator and sensor. The initial context of the two-electrode pacemaker is similar to the one-electrode pacemaker. We give here only newly defined constants and axioms. A new constant atrioventricular (AV) interval (FixedAV) is defined in $axm8$. Refractory period constants Atria Refractory Period (ARP), Ventricular Refractory Period (VRP) and Post Ventricular Atria Refractory Period (PVARP) are defined by axioms ($axm9$,$axm10$ and $axm11$). Another new constant V_Blank is defined as blanking period as an initial period of VRP. Finally, we introduce some basic initial properties using defined constants of the system by axioms($axm13$, $axm14$ and $axm15$).

$axm8 : FixedAV \in 70 .. 300$
$axm9 : ARP \in 150 .. 500$
$axm10 : VRP \in 150 .. 500$
$axm11 : PVARP \in 150 .. 500$
$axm12 : V\_Blank \in 30 .. 60$
$axm13 : URI > PVARP$
$axm14 : URI > VRP$
$axm15 : VRP \geq PVARP$

9.7.1.1 Abstraction of DDD mode:

In the two-electrode pacemaker system, the pacemaker delivers a pacing stimulus in both atrial and ventricular chambers. In DDD operating mode, the first letter 'D' represents that the pacemaker paces in both atrial and ventricle chambers; second letter 'D' represents that the pacemaker senses intrinsic activities from both atrial and ventricle chambers and final letter 'D' represents two conditional meaning that depends on atrial and ventricular sensing; first is that atrial sensing inhibits atrial pacing and triggers ventricular pacing and second is that ventricular sensing inhibits ventricular and atrial pacing[Hesselson 2003; Lee 2006b].

Two new variables *PM_Actuator_A* and *PM_Actuator_V* are defined that represent ON or OFF state of the pacemaker's actuators for pacing in both atrial and ventricular chambers. Similarly next two variables *PM_Sensor_A* and *PM_Sensor_V* represent ON or OFF state of the pacemaker's sensor for sensing an intrinsic pulse from both atrial and ventricular chambers. An interval between two paces is defined by a new variable *Pace_Int* that must be between upper rate interval (URI) and lower rate interval (LRI), is represented by an invariant ($inv5$). A variable sp represents the current *clock counter*. A variable *last_sp* represents the last interval (in ms.) between two paces, and a safety property in invariant ($inv7$) states that the last interval must be between PVARP and pace interval *Pace_Int*. Another new variable *AV_Count_STATE* is defined as boolean type to control the atrioventricular (AV) interval state and the next variable *AV_Count* is defined as a natural number to count the atrioventricular (AV) interval. A variable (*Pace_Int_flag*) is defined as boolean type to represent changing state of the pace interval (*Pace_Int*). Invariants ($inv11$,$inv12$ and $inv13$) represent the safety properties. The invariant $inv11$ states that, when the clock counter sp is less than VRP and atrioventricular (AV) counter state *AV_Count_State* is FALSE, then the pacemaker's actuators and sensors of both chambers are OFF. Similarly, the next invariants ($inv12$ and $inv13$) represent the conditions of ON state of the pacemaker's actuators in the both chambers.

$inv1 : PM\_Actuator\_A \in status$
$inv2 : PM\_Actuator\_V \in status$
$inv3 : PM\_Sensor\_A \in status$
$inv4 : PM\_Sensor\_V \in status$
$inv5 : Pace\_Int \in URI \mathinner{..} LRI$
$inv6 : sp \in 1 \mathinner{..} Pace\_Int$
$inv7 : last\_sp \geq PVARP \wedge last\_sp \leq Pace\_Int$
$inv8 : AV\_Count\_STATE \in BOOL$
$inv9 : AV\_Count \in \mathbb{N}$
$inv10 : Pace\_Int\_flag \in BOOL$

$inv11 : sp < VRP \wedge AV\_Count\_STATE = FALSE$
$\Rightarrow$
$PM\_Actuator\_V = OFF \wedge$
$PM\_Sensor\_A = OFF \wedge$
$PM\_Sensor\_V = OFF \wedge$
$PM\_Actuator\_A = OFF$

$inv12 : Pace\_Int\_flag = FALSE \wedge PM\_Actuator\_V = ON$
$\Rightarrow$
$sp = Pace\_Int \vee (sp < Pace\_Int \wedge$
$AV\_Count > V\_Blank \wedge AV\_Count \geq FixedAV)$

$inv13 : Pace\_Int\_flag = FALSE \wedge PM\_Actuator\_A = ON$
$\Rightarrow$
$(sp \geq Pace\_Int - FixedAV)$

In the abstract specification of DDD operating mode, there are ten events *Actuator_ON_A* to start pacing in atrial, *Actuator_OFF_A* to stop pacing in atrial, *Actuator_ON_V* to start pacing in ventricular, *Actuator_OFF_V* to stop pacing in ventricular, *Sensor_ON_V* to start sensing in ventricular, *Sensor_OFF_V* to stop sensing in ventricular, *Sensor_ON_A* to star sensing in atrial, *Sensor_OFF_A* to stop sensing in atrial, tic to increment the current clock counter sp under real time constraints and *tic_AV* to count the atrioventricular (AV) interval.

EVENT Actuator_ON_V
WHEN
grd1 : $PM\_Actuator\_V = OFF$
grd2 : $(sp = Pace\_Int)$
$\vee$
$(sp < Pace\_Int \wedge AV\_Count > V\_Blank \wedge$
$AV\_Count \geq FixedAV)$
grd3 : $sp \geq VRP \wedge sp \geq PVARP$
THEN
act1 : $PM\_Actuator\_V := ON$
act2 : $last\_sp := sp$
END

The events *Actuator_ON_V* and *Actuator_OFF_V* are used to start and stop the Pacemaker's actuator in the ventricular chamber under the real-time constraints. In the event (*Actuator_ON_V*), the first guard states that the Pacemaker's actuator *(PM_Actuator_V)* of the ventricular is OFF, the next guard $(grd2)$ states that the current clock counter sp is either equal to the pace interval *Pace_Int* or the clock counter sp is less than the pace interval *Pace_Int*, the atrioventricular counter is greater than the blanking period *V_Blank* and the atrioventricular counter is greater than the fixed atrioventricular interval *FixedAV*. The last guard $(grd3)$ states that the clock counter sp is greater than or equal to the VRP and PVARP. The actions of this event show that when all guards are true then the Pacemaker's actuator *(PM_Actuator_V)* of ventricular sets ON and assigns a value of the clock counter sp into other variable *last_sp*.

EVENT Actuator_OFF_V
WHEN
grd1 : $PM\_Actuator\_V = ON$
grd2 : $(sp = Pace\_Int)$
$\vee$
$(sp < Pace\_Int \wedge AV\_Count > V\_Blank \wedge$
$AV\_Count \geq FixedAV$
grd3 : $AV\_Count\_STATE = TRUE$
grd4 : $PM\_Actuator\_A = OFF$
grd5 : $PM\_Sensor\_A = OFF$
THEN
act1 : $PM\_Actuator\_V := OFF$
act2 : $AV\_Count := 0$
act3 : $AV\_Count\_STATE := FALSE$
act4 : $PM\_Sensor\_V := OFF$
act5 : $sp := 1$
END

First two guards of the event *(Actuator_OFF_V)* state that the Pacemaker's actuator *(PM_Actuator_V)* of ventricular is ON, and the clock counter (sp) is equal to the pace interval *(Pace_Int)*, or less than the pace interval *(Pace_Int)*, the atrioventricular (AV) counter *(AV_Count)* is greater than the blanking period *(V_Blank)* and the atrioventricular (AV) counter is greater than or equal to the atrioventricular (AV) interval *(FixedAV)*. Third guard $(grd3)$ states that the atrioventricular (AV) counter state *(AV_Count_STATE)* is TRUE, and last two guards represent that the Pacemaker's actuator and sensor *(PM_Actuator_A, PM_Sensor_A)* of atrial is OFF. In action's part, sets OFF state of the Pacemaker's actuator *(PM_Actuator_V)* of ventricular, reassigns the value of variable *(AV_count)* as 0, sets FALSE state to the AV counter state *(AV_Count_STATE)*, sets OFF state to the Pacemaker's Sensor *(PM_Sensor_V)* of the ventricular chamber and finally assigns the value of the clock counter (sp) as 1.

EVENT Actuator_ON_A
WHEN
grd1 : $PM\_Sensor\_V = ON$
grd2 : $sp \geq Pace\_Int - FixedAV \wedge$
$sp \geq VRP \wedge sp \geq PVARP$
grd3 : $PM\_Actuator\_A = OFF$
grd4 : $PM\_Sensor\_A = ON$
THEN
act1 : $PM\_Actuator\_A := ON$
act2 : $PM\_Sensor\_V := OFF$
act3 : $PM\_Sensor\_A := OFF$
END

A set of new events *Actuator_ON_A* and *Actuator_OFF_A* are introduced to start and stop the Pacemaker's actuator in the atrial chamber. Actions $(act1 - act3)$ of the event *(Actuator_ON_A)* state that the Pacemaker's actuator *(PM_Actuator_A)* of the atria sets ON and the Pacemaker's sensors *(PM_Sensor_V,PM_Sensor_A)* of the ventricular and atrial set OFF when all the guards satisfy. The first guard of this event states that the Pacemaker's sensor *(PM_Sensor_V)* of ventricular is ON, next guard $(grd2)$ states that the clock counter (sp) is greater than or equal to the ventriculoatrial (VA) interval, VRP and PVARP, the third guard shows that the Pacemaker's actuator *(PM_Actuator_A)* of atrial is OFF and last guard states that the Pacemaker's sensor *(PM_Sensor_A)* of atrial is ON.

EVENT Actuator_OFF_A
WHEN
grd1 : $PM\_Actuator\_A = ON$
grd2 : $AV\_Count\_STATE = FALSE$
grd3 : $sp \geq Pace\_Int - FixedAV \wedge$
$sp \geq VRP \wedge sp \geq PVARP$
THEN
act1 : $PM\_Actuator\_A := OFF$
act2 : $AV\_Count\_STATE := TRUE$
END

First two actions $(act1, act2)$ of the event *(Actuator_OFF_A)* state that the Pacemaker's actuator *(PM_Actuator_A)* of atria sets OFF and the atrioventricular (AV) counter state *(AV_Count_STATE)* sets TRUE. The guards $(grd1, grd2)$ of this event state that the Pacemaker's actuator *(PM_Actuator_A)* of the atria is ON, the current clock counter (sp) is greater than or equal to the ventriculoatrial (VA) interval, VRP and PVARP. The last guard presents that the atrioventricular (AV) counter state *(AV_Count_STATE)* is FALSE.

EVENT Sensor_ON_V
WHEN
grd1 : $PM\_Sensor\_V = OFF$
grd2 : $(sp \geq VRP \wedge sp < Pace\_Int - FixedAV \wedge$
$PM\_Sensor\_A = ON)$
$\vee$
$(sp \geq Pace\_Int - FixedAV \wedge$
$AV\_Count\_STATE = TRUE)$
grd3 : $PM\_Actuator\_A = OFF$
THEN
act1 : $PM\_Sensor\_V := ON$
END

The events (*Sensor_ON_V* and *Sensor_OFF_V*) are used to control the sensing activities from the ventricular chamber. The Pacemaker's sensor *(PM_Sensor_V)* of ventricular chamber synchronizes ON and OFF states under the real-time constraints. The Pacemaker's sensor *(PM_Sensor_V)* of the ventricular is ON when all guards are satisfied. The event *(Sensor_OFF_V)* is used to set the Pacemaker's sensor *(PM_Sensor_V)* of the ventricular as OFF and resets all other variables in the action part when all guards fulfill the required conditions. The guards represent the different states of the Pacemaker's actuators and sensors under the real-time constraints with various time interval parameters (i.e. VRP, PVARP, *Pace_Int* etc.).

EVENT Sensor_OFF_V
WHEN
grd1 : $PM\_Sensor\_V = ON$
grd2 : $sp \geq VRP \wedge sp \geq PVARP$
grd3 : $(sp < Pace\_Int - FixedAV)$
$\vee$
$(sp \geq Pace\_Int - FixedAV \wedge$
$sp < Pace\_Int)$
grd4 : $PM\_Actuator\_V = OFF$
grd5 : $PM\_Actuator\_A = OFF$
THEN
act1 : $PM\_Sensor\_V := OFF$
act2 : $AV\_Count := 0$
act3 : $AV\_Count\_STATE := FALSE$
act4 : $last\_sp := sp$
act5 : $sp := 1$
act6 : $PM\_Sensor\_A := OFF$
END

The event *(Sensor_OFF_V)* is used to set the Pacemaker's sensor *(PM_Sensor_V)* of ventricular in OFF state. The guards $(grd1, grd2)$ of this event represent that the Pacemaker's sensor *(PM_Sensor_V)* of ventricular is ON, and the clock counter (sp) is greater than or equal to the VRP and PVARP. Next guard $(grd3)$ represents that the clock counter

sp is less than the ventriculoatrial (VA) interval, or greater than or equal to the ventriculoatrial (VA) interval and less than the pace interval *(Pace_Int)*. The last two guards $(grd4, grd5)$ state that the Pacemaker's actuators *(PM_Actuator_V, PM_Actuator_A)* of the ventricular and atrial are OFF. The actions $(act1 - act6)$ of this event state that the Pacemaker's sensor *(PM_Sensor_V)* of ventricular is OFF, assigns the value of variable *(AV_count)* as 0, the atrioventricular (AV) counter state *(AV_Count_STATE)* sets FALSE, assigns the value of the clock counter (sp) to new variable *(last_sp)*, assigns the value of the clock counter (sp) as 1 and sets OFF state of the Pacemaker's actuator *(PM_Actuator_A)* of atrial.

EVENT Sensor_ON_A
WHEN
grd1 : $PM\_Sensor\_A = OFF$
grd2 : $PM\_Sensor\_V = OFF$
grd3 : $sp < Pace\_Int - FixedAV \wedge sp \geq VRP \wedge sp \geq PVARP$
THEN
act1 : $PM\_Sensor\_A := ON$
END

The events (*Sensor_ON_A* and *Sensor_OFF_A*) are used to control the sensing activities from the atrial chamber. The Pacemaker's sensor *(PM_Sensor_A)* of the atrial chamber synchronizes ON and OFF states under the real time constraints. A guard $(grd1)$ of the event *(Sensor_ON_A)* represents that if the Pacemaker's sensor *(PM_Sensor_A)* of atrial is OFF and second guard $(grd2)$ represents that the clock counter (sp) is less than the ventriculoatrial (VA) interval and greater than or equal to the VRP and PVARP. The last guard $(grd3)$ represents that the Pacemaker's sensor *(PM_Sensor_V)* of ventricular is OFF. If all guards are true, then in action part of this event the Pacemaker's Sensor *(PM_Sensor_A)* of atrial sets ON.

EVENT Sensor_OFF_A
WHEN
grd1 : $PM\_Sensor\_A = ON$
grd2 : $sp < Pace\_Int - FixedAV \wedge sp \geq VRP \wedge sp \geq PVARP$
THEN
act1 : $PM\_Sensor\_A := OFF$
act2 : $AV\_Count\_STATE := TRUE$
END

The event *(Sensor_OFF_A)* is used to set the Pacemaker's sensor *(PM_Sensor_A)* of atrial in OFF state. The guards of this event represent that the Pacemaker's sensor *(PM_Sensor_A)* of atrial is ON, and the clock counter (sp) is less than the ventriculoatrial (VA) interval and greater than or equal to the VRP and PVARP. In actions of this event state that the Pacemaker's sensor *(PM_Sensor_A)* of atrial sets OFF and the atrioventricular (AV) counter state *(AV_Count_STATE)* sets TRUE.

EVENT tic
WHEN
grd1 : $(sp < VRP)$
$\vee$
$(sp \geq VRP \wedge sp < Pace\_Int - FixedAV \wedge$
$PM\_Sensor\_A = ON \wedge PM\_Sensor\_V = ON$
THEN
act1 : $sp := sp + 1$
END

The event (tic) of this abstraction progressively increases the current clock counter sp under the pre-defined pace interval *(Pace_Int)*. This event controls the time line of pacing and sensing events. A guard $(grd1)$ of this event provides the required conditions to increase the clock counter sp by 1 (ms.). The last event *(tic_AV)* of this abstraction progressively counts the atrioventricular (AV) interval and increases the current clock counter sp is represented by actions $act1$ and $act2$, respectively.

EVENT tic_AV
WHEN
grd1 : $AV\_Count < FixedAV$
grd2 : $AV\_Count\_STATE = TRUE$
grd3 : $(sp \geq VRP \wedge sp \geq PVARP \wedge$
$sp < Pace\_Int - FixedAV)$
$\vee$
$(sp \geq Pace\_Int - FixedAV \wedge$
$sp < Pace\_Int)$
THEN
act1 : $AV\_Count := AV\_Count + 1$
act2 : $sp := sp + 1$
END

A new event *Change_Pace_Int* is introduced to update the current value of the pace interval. This event is defined abstractly, which is used in further refinement levels to modify the pace interval according to the introduction of different operating modes like hysteresis and rate modulation. This event represents that when pace changing flag (*Pace_Int_flag*) is TRUE then the pace interval (*Pace_Int*) can be chosen from URI to LRI ranges.

EVENT Change_Pace_Int
WHEN
grd1 : $Pace\_Int\_flag = TRUE$
THEN
act1 : $Pace\_Int :\in URI \,..\, LRI$
END

9.7.1.2 Abstraction of DVI mode:

In DVI operating mode of the two-electrode pacemaker system, the first letter 'D' represents that the pacemaker paces both atrial and ventricle; second letter 'V' represents that the pacemaker only senses the ventricle and final letter 'I' represents that the ventricular sensing inhibits atrial and ventricular pacing [Hesselson 2003; Lee 2006b].

In this subsection, we formalize the operating mode (DVI) of the two-electrode pacemaker system. Variables, constants and some invariants ($inv1, inv2$ and $inv4 - inv10$) are similar to the previous operating mode; DDD. More invariants are introduced in this operating mode (DVI) as the safety properties. Invariant $(inv11)$ states that, when the clock counter sp is less than VRP, then the pacemaker's actuator of both chambers and pacemaker's sensor of ventricular are OFF. Next two invariants ($inv12$ and $inv13$) state that, when the pacemaker's actuator *(PM_Actuator_A)* of atrial is ON, then the clock counter sp is greater than or equal to the ventriculoatrial (VA) interval *Pace_Int-FixedAV* and when the pacemaker's actuator *(PM_Actuator_V)* of ventricular is ON, then the clock counter sp is equal to the pace interval *Pace_Int*, respectively.

$$
\begin{aligned}
&inv11 : sp < VRP \Rightarrow PM\_Actuator\_A = OFF \wedge \\
&\qquad PM\_Actuator\_V = OFF \wedge \\
&\qquad PM\_Sensor\_V = OFF \\
&inv12 : Pace\_Int\_flag = FALSE \wedge PM\_Actuator\_A = ON \Rightarrow \\
&\qquad sp \geq Pace\_Int - FixedAV \\
&inv13 : Pace\_Int\_flag = FALSE \wedge PM\_Actuator\_V = ON \Rightarrow sp = Pace\_Int
\end{aligned}
$$

In the abstract specification of DVI operating mode, there are eight events *Actuator_ON_A* to start pacing in atrial, *Actuator_OFF_A* to stop pacing in atrial, *Actuator_ON_V* to start pacing in ventricular, *Actuator_OFF_V* to stop pacing in ventricular, *Sensor_ON_V* to start sensing in ventricular, *Sensor_OFF_V* to stop sensing in ventricular, tic to increment the current clock counter sp under the real time constraints and *tic_AV* to count the atrioventricular (AV) interval. All these events are similar to the DDD operating modes, which are already described. The guards of events are not exactly similar to the DDD operating modes. Guards and actions are changed according to the requirements of the DVI operating mode.

9.7.1.3 Abstraction of DDI mode:

In DDI operating mode of the two-electrode pacemaker system, the first letter 'D' represents that the pacemaker paces both atrial and ventricle; second letter 'D' represents that the pacemaker senses both atrial and ventricle and final letter 'I' represents two conditional meaning that depends on atrial and ventricular sensing; first, atrial sensing inhibits atrial pacing and does not trigger ventricular pacing and second, ventricular sensing inhibits ventricular and atrial pacing[Hesselson 2003; Lee 2006b].

We formalize the operating mode DDI of the two-electrode pacemaker system. Variables, constants and some invariants ($inv1 - inv10$) are similar to the previous operating mode; DDD. Invariant $(inv11)$ states that, when the clock counter sp is less than the VRP,

and the atrioventricular (AV) counter state *(AV_Count_STATE)* is TRUE, then the pacemaker's actuators and sensors of both chambers are OFF. The next invariant ($inv12$) represents that, when the pacemaker's actuator of ventricular is ON, then the clock counter sp is equal to the pace interval *Pace_Int*.

$$
\begin{aligned}
&inv11 : sp < VRP \wedge AV\_Count\_STATE = FALSE \Rightarrow \\
&\qquad PM\_Actuator\_A = OFF \wedge \\
&\qquad PM\_Actuator\_V = OFF \wedge \\
&\qquad PM\_Sensor\_A = OFF \wedge \\
&\qquad PM\_Sensor\_V = OFF \\
&inv12 : Pace\_Int\_flag = FALSE \wedge PM\_Actuator\_V = ON \Rightarrow sp = Pace\_Int
\end{aligned}
$$

In the abstract specification of the DDI operating mode, there are ten events exactly similar to the DDD operating mode, which are already described. The guards and actions of the events are changed according to the DDI operating mode requirements.

9.7.1.4 Abstraction of VDD mode:

In VDD operating mode of the two-electrode pacemaker system, the first letter 'V' represents that the pacemaker only paces ventricle; second letter 'D' represents that the pacemaker senses both atrial and ventricle and final letter 'D' represents two conditional meanings that depend on atrial and ventricular sensing; first, atrial sensing triggers ventricular pacing and second, ventricular sensing inhibits ventricular pacing[Hesselson 2003; Lee 2006b].

In this model, we formalize the functional behaviors of the pacemaker system in VDD operating mode, where all variables, constants and invariants ($inv2 - inv10$) are similar to the previously described DDD operating mode. Here, a new invariant $(inv11)$ states that, when the clock counter sp is less then VRP and atrioventricular (AV) counter state *(AV_Count_STATE)* is FALSE, then the pacemaker's actuator *(PM_Actuator_V)* of the ventricular is OFF, and the pacemaker's sensors of both chambers are OFF. Next invariant ($inv12$) represents that, when the pacemaker's actuator (*PM_Actuator_V*) of ventricular is ON, then the clock counter sp is either equal to the pace interval *Pace_Int* or the clock counter sp is less than the pace interval *Pace_Int* and the atrioventricular (AV) counter *(AV_Count)* is greater than the blanking period *(V_Blank)*, and greater than or equal to the fixed atrioventricular (AV) period *(FixedAV)*.

$$
\begin{aligned}
&inv11 : sp < VRP \wedge AV\_Count\_STATE = FALSE \Rightarrow \\
&\qquad PM\_Actuator\_V = OFF \wedge \\
&\qquad PM\_Sensor\_A = OFF \wedge \\
&\qquad PM\_Sensor\_V = OFF \\
&inv12 : Pace\_Int\_flag = FALSE \wedge PM\_Actuator\_V = ON \Rightarrow \\
&\qquad (sp = Pace\_Int \\
&\qquad \vee \\
&\qquad (sp < Pace\_Int \wedge \\
&\qquad AV\_Count > V\_Blank \wedge \\
&\qquad AV\_Count \geq FixedAV))
\end{aligned}
$$

In the abstract specification of VDD operating mode, there are eight events *Actuator_ON_V* to start pacing in ventricular, *Actuator_OFF_V* to stop pacing in ventricular, *Sensor_ON_V* to start sensing in ventricular, *Sensor_OFF_V* to stop sensing in ventricular, *Sensor_ON_A* to star sensing in atrial, *Sensor_OFF_A* to stop sensing in atrial, tic to increment the current clock counter sp under the real time constraints and *tic_AV* to count the atrioventricular (AV) interval. All these events are similar to the DDD operating modes, which are already described.

9.7.1.5 Abstraction of DOO mode:

In DOO operating mode of the two-electrode pacemaker system, the first letter 'D' represents that the pacemaker paces both atrial and ventricle, second letter 'O' represents that the pacemaker does not sense the atrial and ventricular chambers and final letter 'O' represents that there is no any inhibits or triggers modes in both chambers[Hesselson 2003; Lee 2006b].

In this model, we formalize the functional behaviors of the pacemaker system of DOO operating mode, where all variables, constants and invariants ($inv1, inv2$ and $inv5 - inv10$) are similar to the previous operating mode; DDD. New invariant $(inv11)$ states that the pacemaker's actuator of the atrial and ventricular chambers are OFF, when the clock counter sp is less than the ventriculoatrial (VA) interval, and the atrial state *(Atria_state)* is FALSE. The next invariant ($inv12$) states that the pacemaker's actuators of both chambers are OFF, when the clock counter sp is greater than the atrioventricular (AV) interval, and the atrial state *(Atria_state)* is TRUE. The last invariants ($inv13$ and $inv14$) state that, when the pacemaker's actuator of atrial is ON, then the clock counter sp is greater than or equal to the ventriculoatrial (VA) interval *(Pace_Int-FixedAV)* and when the pacemaker's actuator of the ventricular is ON, then the clock counter sp is equal to the pace interval *Pace_Int*, respectively.

$$
\begin{aligned}
&inv11 : Pace\_Int\_flag = FALSE \wedge sp < (Pace\_Int - FixedAV) \wedge \\
&\qquad Atria\_state = FALSE \Rightarrow \\
&\qquad PM\_Actuator\_V = OFF \wedge \\
&\qquad PM\_Actuator\_A = OFF \\
&inv12 : Pace\_Int\_flag = FALSE \wedge sp > (Pace\_Int - FixedAV) \wedge \\
&\qquad sp < Pace\_Int \wedge \\
&\qquad Atria\_state = TRUE \Rightarrow \\
&\qquad PM\_Actuator\_A = OFF \wedge \\
&\qquad PM\_Actuator\_V = OFF \\
&inv13 : Pace\_Int\_flag = FALSE \wedge PM\_Actuator\_A = ON \Rightarrow \\
&\qquad sp = Pace\_Int - FixedAV \\
&inv14 : Pace\_Int\_flag = FALSE \wedge PM\_Actuator\_V = ON \Rightarrow sp = Pace\_Int
\end{aligned}
$$

In the abstract specification of DOO operating mode, there are five events *Pace_ON_A* to start pacing in atrial, *Pace_OFF_A* to stop pacing in atrial, *Pace_ON_V* to start pacing in ventricular, *Pace_OFF_V* to stop pacing in ventricular and tic to increment the current clock counter sp under the real time constraints. These events are similar to the previously

described events but guards, and actions are changed according to the requirements of the DOO operating mode.

9.7.2 First refinement:Threshold

The pacemaker control unit delivers stimulation to the heart chambers, on the basis of measured threshold value under the safety margin. We define two new constants *STA_THR_A* and *STA_THR_V* to hold the standard threshold value in axioms ($axm1$ and $axm2$). The threshold constants are different for the atrial and the ventricular chambers.

$$axm1 : STA\_THR\_A \in nat_1 \wedge STA\_THR\_A = 75$$
$$axm1 : STA\_THR\_V \in nat_1 \wedge STA\_THR\_V = 250$$

The pacemaker's sensor starts sensing after the refractory period but the pacemaker's actuator delivers a pacing stimulus, when sensing value is greater than or equal to the standard threshold constants *STA_THR_A* or *STA_THR_V*. In the DOO operating mode only the pacemaker's actuators paces in the atrial and ventricular chambers under the automatic pace interval without using any pacemaker's sensors, so in this mode none of the refinement is given related to the threshold. Table-3 shows a list of invariants common in this refinement of other operating modes. First column shows the group of operating modes and second column shows corresponding common invariants.

Modes	Common Invariants
DDD VDD DDI DVI	1. $Pace\_Int\_flag = FALSE \wedge sp > Pace\_Int - FixedAV \wedge sp < Pace\_Int \wedge AV\_Count\_STATE = TRUE \Rightarrow PM\_Sensor\_V = ON$
DDD DVI	2. $Pace\_Int\_flag = FALSE \wedge sp > VRP \wedge sp < Pace\_Int - FixedAV \Rightarrow PM\_Sensor\_V = ON$
DDD VDD	3. $Pace\_Int\_flag = FALSE \wedge PM\_Actuator\_V = ON \Rightarrow (sp = Pace\_Int) \vee (sp < Pace\_Int \wedge AV\_Count > V\_Blank \wedge AV\_Count \geq FixedAV)$ 4. $Pace\_Int\_flag = FALSE \wedge sp > Pace\_Int - FixedAV \wedge sp < Pace\_Int \wedge AV\_Count\_STATE = TRUE \Rightarrow PM\_Sensor\_A = OFF$
DDD DVI DDI	5. $Pace\_Int\_flag = FALSE \wedge sp > Pace\_Int - FixedAV \wedge sp < Pace\_Int \wedge AV\_Count\_STATE = TRUE \Rightarrow PM\_Actuator\_A = OFF$
DVI DDI	6. $Pace\_Int\_flag = FALSE \wedge sp > Pace\_Int - FixedAV \wedge sp < Pace\_Int \wedge AV\_Count\_STATE = TRUE \Rightarrow PM\_Actuator\_V = OFF$

Table 3 : Common Invariants List

9.7.2.1 First refinement of DDD mode:

A pacemaker has a stimulation threshold measuring unit which measures a stimulation threshold voltage value of heart and a pulse generator for delivering stimulation pulses to

the heart. The pulse generator is controlled by a control unit to deliver the stimulation pulses with respective amplitudes related to the measured threshold value under the safety margin. We introduce two new variables *Thr_A* and *Thr_V* to hold the sensing threshold value of the pacemaker's sensor from the atrial and ventricular chambers. Similarly, next two variables *Thr_A_State* and *Thr_V_State* represent boolean states as TRUE or FALSE of the pacemaker's sensor to sense the intrinsic activity from the atrial and ventricular chambers.

$$
\begin{array}{l}
inv1 : Thr\_A \in \mathbb{N}_1 \\
inv2 : Thr\_V \in \mathbb{N}_1 \\
inv3 : Thr\_A\_State \in BOOL \\
inv4 : Thr\_V\_State \in BOOL \\
inv5 : Pace\_Int\_flag = FALSE \wedge PM\_Actuator\_A = ON \Rightarrow \\
\qquad sp \geq Pace\_Int - FixedAV
\end{array}
$$

Invariants are given in Table-3. An additional invariant ($inv5$) is introduced and states that, when the pacemaker's actuator of the atrial chamber is ON, then the current clock counter sp is greater than or equal to the ventriculoatrial (VA) interval *(Pace_Int- FixedAV)*.

EVENT Thr_Value_V
ANY Thr_V_val
WHERE
grd1 : $Thr\_V\_val \in \mathbb{N}$
grd2 : $PM\_Sensor\_V = ON$
grd3 : $Thr\_V\_State = TRUE$
grd4 : $Thr\_V < STA\_THR\_V$
grd5 : $(sp \geq VRP \wedge sp < Pace\_Int - FixedAV)$
$\vee$
$(sp \geq Pace\_Int - FixedAV \wedge sp < Pace\_Int)$
grd6 : $(Thr\_A\_State = FALSE \wedge$
$Thr\_A < STA\_THR\_A)$
$\vee$
$(PM\_Sensor\_A = OFF \wedge$
$AV\_Count < FixedAV)$
THEN
act1 : $Thr\_V := Thr\_V\_val$
act2 : $Thr\_V\_State := FALSE$
END

In this refinement, we introduce two new events (*Thr_Value_V* and *Thr_Value_A*) for sensing the intrinsic activities from the ventricular and atrial chambers. These events are synchronized with all other events of the operating mode under all the safety properties and real time constraints. The guards of the event (*Thr_Value_V*) are introduced as to fulfill all the requirements of the sensing intrinsic activities from the ventricular chamber and actions *(act1-act2)* of this event state that the actual sensed value from a chamber is assigned to the variable *Thr_V* and sets FALSE state of the variable threshold ventricular state *(Thr_V_State)*, respectively.

EVENT Thr_Value_A
ANY Thr_A_val
WHERE
grd1 : $Thr\_A\_val \in \mathbb{N}$
grd2 : $PM\_Sensor\_A = ON$
grd3 : $Thr\_A\_State = TRUE$
grd4 : $Thr\_A < STA\_THR\_A$
grd5 : $(sp \geq VRP \wedge sp < Pace\_Int - FixedAV)$
THEN
act1 : $Thr\_A := Thr\_A\_val$
act2 : $Thr\_A\_State := FALSE$
END

In the event *Thr_Value_A*, the guards $(grd2 - grd4)$ state that the pacemaker's sensor *(PM_Sensor_A)* of the atrial chamber is ON; the threshold state *(Thr_A_State)* of the atrial chamber is TRUE and the sensed value *(Thr_A)* from the atrial chamber is less than the standard threshold *(STA_THR_A)* of the atrial chamber. The last guard of this event states that the clock counter sp is greater than or equal to the VRP and less than the ventriculoatrial (VA) interval. Actions *(act1-act2)* of this event state that the actual sensed value *(Thr_A_val)* of the atrial chamber is assigned to the variable *(Thr_A)* and sets FALSE state of the variable threshold atrial state (*Thr_A_State*).

EVENT Actuator_OFF_V
$\oplus$ act6 : $Thr\_A := 0$
$\oplus$ act7 : $Thr\_V := 0$
$\oplus$ act8 : $Thr\_A\_State := FALSE$
$\oplus$ act9 : $Thr\_V\_State := FALSE$

EVENT Sensor_ON_A
$\oplus$ act2 : $Thr\_A\_State := TRUE$

EVENT Sensor_OFF_A
$\oplus$ grd3 : $Thr\_A \geq STA\_THR\_A$

EVENT Sensor_OFF_V
$\oplus$ grd6 : $Thr\_V \geq STA\_THR\_V$
$\oplus$ act7 : $Thr\_A := 0$
$\oplus$ act8 : $Thr\_V := 0$
$\oplus$ act9 : $Thr\_A\_State := FALSE$
$\oplus$ act10 : $Thr\_V\_State := FALSE$

We have introduced some new actions and guards in events *(Actuator_OFF_V*, *Sensor_ON_A*, *Sensor_OFF_A* , and *Sensor_OFF_V)* to synchronize the sensing activities using events *(Thr_Value_V* and *Thr_Value_A)* under the real time constraints. These events are already defined in the abstract model[3].

[3] $\oplus$: To add a new guard and an action in the model. $\ominus$: To remove a new guard and an action in the model.

EVENT tic
WHEN
grd1 : $(sp < VRP \wedge AV\_Count\_STATE = FALSE$
$\vee$
$(sp \geq VRP \wedge sp < Pace\_Int - FixedAV \wedge$
$PM\_Sensor\_V = ON \wedge$
$PM\_Sensor\_A = ON \wedge$
$Thr\_V\_State = FALSE \wedge$
$Thr\_V < STA\_THR\_V))$
grd2 : $AV\_Count\_STATE = FALSE$
THEN
$\oplus$ act2 : $Thr\_A\_State := TRUE$
$\oplus$ act3 : $Thr\_V\_State := TRUE$
END

The event (tic) of this refinement model progressively increases the current clock counter sp. We have modified the guard in this event to properly synchronize with new introduced threshold events, and pacing and sensing activities of the both chambers. Some new actions $(act2$ and $act3)$ are added in this event. The additional guards and actions handle the behavior of the events *(Thr_Value_A* and *Thr_Value_V)* to sense the intrinsic activities from the atrial and ventricular chambers.

EVENT tic_AV
WHEN
$\oplus$ grd4 : $PM\_Sensor\_V = ON$
$\oplus$ grd5 : $Thr\_V\_State = FALSE$
$\oplus$ grd6 : $Thr\_V < STA\_THR\_V$
$\oplus$ grd7 : $PM\_Actuator\_V = OFF$
$\oplus$ grd8 : $PM\_Sensor\_A = OFF$
$\oplus$ grd9 : $PM\_Actuator\_A = OFF$
THEN
$\oplus$ act3 : $Thr\_V\_State := TRUE$
END

We have introduced some new guards $(grd4 - grd9)$ and an action $(act3)$ in event *(tic_AV)* of this refinement. New guards provide more specific conditions and some specific states of the pacemaker's actuators and sensors to count the atrioventricular (AV) interval. An extra action $(act3)$ sets TRUE state of the variable threshold state of ventricular (*Thr_V_State*).

9.7.2.2 First refinement of DVI mode:

In this refinement, we introduce two new variables *Thr_V* and *Thr_V_State* to hold the sensing threshold value as similar to the DDD operating mode. We introduce few more invariants except some defined common invariants (see in Table-3).

$$
\begin{aligned}
&inv1 : Pace\_Int\_flag = FALSE \wedge PM\_Actuator\_V = ON \Rightarrow sp = Pace\_Int \\
&inv2 : Pace\_Int\_flag = FALSE \wedge sp > VRP \wedge sp < Pace\_Inta \wedge \\
&\qquad Thr\_V \geq STA\_THR\_V \wedge \\
&\qquad Thr\_V\_State = TRUE \Rightarrow \\
&\qquad PM\_Sensor\_V = OFF \\
&inv3 : Pace\_Int\_flag = FALSE \wedge PM\_Actuator\_A = ON \Rightarrow \\
&\qquad sp \geq Pace\_Int - FixedAV \wedge sp \geq VRP \wedge sp < Pace\_Int
\end{aligned}
$$

The first invariant $(inv1)$ states that when the pacemaker's actuator (*PM_Actuator_V*) of the ventricular is ON, then the current clock counter sp is equal to the pace interval *Pace_Int*. Second invariant $(inv2)$ represents that the pacemaker's sensor *(PM_Sensor_V)* of the ventricular is OFF, when the clock counter sp is greater than the VRP, less than the pace interval *(Pace_Int)*, the sensed value *(Thr_V)* is greater than or equal to the standard threshold *(STA_THR_V)* value of the ventricular chamber and the threshold ventricular state *(Thr_V_State)* is TRUE. The last invariant $(inv3)$ states that, when the pacemaker's actuator of the atrial chambers is ON, then the current clock counter sp is within the ventriculoatrial (VA) interval *(Pace_Int - FixedAV)* and greater than or equal to the VRP and less than the pace interval *Pace_Int*.

In this refinement, we introduce a new event *(Thr_Value_V)* for sensing the intrinsic activities of the ventricular chamber, and it is similar to the first refinement of DDD operating mode. This event is synchronized with all other events of this operating mode under all the safety properties and real time constraints. The other events tic and *tic_AV* are also modified in this refinement to synchronize sensors and actuators behavior.

9.7.2.3 First refinement of DDI mode:

We introduce some new variables (*Thr_A*, *Thr_V*, *Thr_A_State* and *Thr_V_State*) as similar to the refinement of the DDD operating modes. In this refinement, we introduce some new invariants except some defined common invariants (see in Table-3).

$$
\begin{aligned}
&inv1 : Pace\_Int\_flag = FALSE \wedge PM\_Actuator\_V = ON \Rightarrow sp = Pace\_Int \\
&inv2 : Pace\_Int\_flag = FALSE \wedge sp > VRP \wedge sp < Pace\_Int - FixedAV \Rightarrow \\
&\qquad PM\_Actuator\_A = OFF \\
&inv3 : Pace\_Int\_flag = FALSE \wedge PM\_Actuator\_A = ON \Rightarrow \\
&\qquad sp = Pace\_Int - FixedAV
\end{aligned}
$$

The first invariant states that when the pacemaker's actuator *(PM_Actuator_V)* of the ventricular is ON, then the clock counter sp is equal to the pace interval *Pace_Int*. The next invariant $(inv2)$ states that the pacemaker's actuator *(PM_Actuator_A)* of the atrial is OFF, when the current clock counter sp is greater than the VRP and less than the ventriculoatrial(VA) ventriculoatrial(VA interval. The last invariant states that, when the pacemaker's actuator of the atrial chamber is ON, then the current clock counter sp is equal to ventriculoatrial (VA) interval *(Pace_Int - FixedAV)*.

In this refinement, we introduce two new events *(Thr_Value_V* and *Thr_Value_A)* for sensing the intrinsic activities from the ventricular and atrial chambers that are similar to

the first refinement of DDD operating mode. These events are synchronized with all other events of this operating mode under all the safety properties and real time constraints. Other events are also modified in this refinement to synchronize the sensors and actuators behavior as similar to the DDD operating mode.

9.7.2.4 First refinement of VDD mode:

We introduce four new variables (*Thr_A*, *Thr_V*, *Thr_A_State* and *Thr_V_State*) as similar to the refinement of the DDD operating mode. In this refinement, we introduce an extra invariant except some defined common invariants (see in Table-3). Invariant $(inv1)$ states that the pacemaker's sensor *(PM_Sensor_A)* of the atrial is ON, when the clock counter sp is greater than the VRP and less than the pace interval *(Pace_Int)* and the atrioventricular (AV) counter state *(AV_Count_STATE)* is FALSE.

$$inv1 : Pace\_Int\_flag = FALSE \wedge sp > VRP \wedge sp < Pace\_Int \wedge AV\_Count\_STATE = FALSE \Rightarrow PM\_Sensor\_A = ON$$

In this refinement, we introduce two new events *(Thr_Value_V* and *Thr_Value_A)* as similar to the DDD operating mode. These events are synchronized with all other events of this operating mode under all the safety properties and real time constraints. Some guards and actions are added in the old events as defined in the DDD operating mode.

9.7.3 Second refinement of DDD mode: Hysteresis

In the two electrode pacemaker, *hysteresis* mode is applicable only in the DDD operating mode. *Hysteresis* is a programmed feature whereby the pacemaker paces at a faster rate than the sensing rate. For example, pacing at 80 pulses a minute with a hysteresis rate of 55 means that the pacemaker will be inhibited at all rates down to 55 beats per minute, having been activated at a rate below 55, the pacemaker then switches on and paces at 80 pulses a minute [Malmivuo 1995; Hesselson 2003]. The application of the hysteresis interval provides consistent pacing of the atrial or ventricle, or prevents constant pacing of the atrial or ventricle. The main purpose of hysteresis is to allow a patient to have his or her own underlying rhythm as much as possible. Two new variables (*Hyt_Pace_Int_flag*, *HYT_State*) are introduced to define functional properties of the hysteresis operating modes. Both variables are defined as boolean types. The hysteresis state *HYT_State* is used to set the hysteresis functional parameter as TRUE or FALSE, to apply the hysteresis operating modes.

$$\text{inv1} : Hyt\_Pace\_Int\_flag \in BOOL$$
$$\text{inv2} : HYT\_State \in BOOL$$

A new event *Hyt_Pace_Updating* is introduced to implement the functional properties of the hysteresis operating modes, which is a refinement of the event *Change_Pace_Int*. In

the hysteresis operating modes, the pacemaker is trying to maintain own heart rhythm as much as possible. Hence, this event can change the pacing interval and sets pacing length longer than existing, which changes the pacing length of the cardiac pacemaker. This event is only used for updating the pacing interval (*Pace_Int*). Guards of this event state that the pace changing flag (*Pace_Int_flag*) is TRUE, the hysteresis pacing flag (*Hyt_Pace_-Int_flag*) is TRUE and the hysteresis pace interval (*Hyt_Pace_Int*) should be lied between the pace interval (*Pace_Int*) and lower rate interval (LRI). The actions of this event state that a new hysteresis pace interval (*Hyt_Pace_Int*) updates the pace interval *Pace_Int*, hthe ysteresis pacing flag (*Hyt_Pace_Int_flag*) sets FALSE and hysteresis state (*HYT_State*) sets TRUE.

EVENT Hyt_Pace_Updating Refines Change_Pace_Int
ANY
Hyt_Pace_Int
WHERE
grd1 : $Pace\_Int\_flag = TRUE$
grd2 : $Hyt\_Pace\_Int\_flag = TRUE$
grd3 : $Hyt\_Pace\_Int \in Pace\_Int \mathinner{..} LRI$
THEN
act1 : $Pace\_Int := Hyt\_Pace\_Int$
act2 : $Hyt\_Pace\_Int\_flag := FALSE$
act3 : $HYT\_State := TRUE$
END

9.7.4 Third refinement: Rate Modulation

Rate modulation is the final refinement of the two-electrode pacemaker. Rate modulation refers to the ability of the pacemaker to increase the rate of pacing on its own. The manner that the pacemaker does this is by having its own special sensor's measure such as things as vibration or minute ventilation (volume of air moved in 1 minute's time). The pacemaker uses these measurements as a determination of at least how fast heart rate should be. This rate is termed the "sensor indicated rate."

Rate modulation is typically used when a patient's heart does not appropriately increase its own rate with exertion or stress. This intrinsic inability to increase heart rate is called "chronotropic incompetence." Use of rate modulation also demands to set an upper limit on how fast the heart may be paced.

This refinement is similar to the one-electrode pacemaker and pacing rate control both chambers according to the required physiologic need. Here, we introduce the rate modulation function and found some new operating modes (DDDR, DVIR, AAIR, DDIR, VDDR and DOOR) of the two-electrode pacemaker system. For modeling the rate modulation, we introduce some new constants maximum sensor rate MSR as $MSR \in 50 \mathinner{..} 175$ and acc_thr as $acc\_thr \in \mathbb{N}_1$ using axioms ($axm1, axm2$). The maximum sensor rate (MSR) is the maximum pacing rate allowed as a result of sensor control, and it must be between 50 and 175 pulse per minute (ppm). The constant *acc_thr* represents the activity threshold. Axiom ($axm3$) represents a static property for the rate modulation operating modes.

axm1 : $MSR \in 50 \mathinner{..} 175$
axm2 : $acc\_thr \in \mathbb{N}_1$
axm3 : $MSR = URL$

Two new variables *acler_sensed* and *acler_sensed_flag* are defined as to store the measured value from the accelerometer and boolean stats of the accelerometer sensor. Boolean state of the accelerometer sensor is used to synchronize with other functionalities of the system. The accelerometer is used to measure the physical activities of the body in the pacemaker system. Two invariants $(inv3, inv4)$ provide the safety margin and state that the heart rate never falls below the lower rate limit (LRL) and never exceeds the maximum sensor rate (MSR) limit.

inv1 : $acler\_sensed \in \mathbb{N}$
inv2 : $acler\_sensed\_flag \in BOOL$
inv3 : $HYT\_State = FALSE \wedge acler\_sensed < acc\_thr \wedge$
$acler\_sensed\_flag = TRUE \Rightarrow Pace\_Int = 60000/LRL$
inv4 : $HYT\_State = FALSE \wedge acler\_sensed =\geq acc\_thr \wedge$
$acler\_sensed\_flag = TRUE \Rightarrow Pace\_Int = 60000/MSR$

In this final refinement, we introduce two new events *Increase_Interval* and *Decrease_Interval*, which are the refinements of the event *Change_Pace_Int*. These new events are used to control the pacing rate of the one-electrode pacemaker in the rate modulating operating modes. The new events *Increase_Interval* and *Decrease_Interval* control the value of the pace interval variable *Pace_Int*, whenever a measured value (*acler_sensed*) from the accelerometer sensor goes higher or lower than the activity threshold *acc_thr*.

EVENT Increase_Interval Refines Change_Pace_Int
WHEN
grd1 : $Pace\_Int\_flag = TRUE$
grd1 : $acler\_sensed \geq threshold$
grd1 : $HYT\_State = FALSE$
THEN
act1 : $Pace\_Int := 60000/MSR$
act1 : $acler\_sensed\_flag := TRUE$
END

EVENT Decrease_Interval Refines Change_Pace_Int
WHEN
grd1 : $Pace\_Int\_flag = TRUE$
grd1 : $acler\_sensed < threshold$
grd1 : $HYT\_State = FALSE$
THEN
act1 : $Pace\_Int := 60000/LRL$
act1 : $acler\_sensed\_flag := TRUE$
END

A new event (*Acler_sensed*) is defined as to simulate the behaviour of the accelerometer sensor. This event is continued sensing the motion of the body to increase or decrease the length of the pace interval (*Pace_Int*). In this event guards state that the accelerometer sensor flag is TRUE and the hysteresis state is FALSE. A new variable *acl_sen* is used to store the current sensing value. Actions of this event state that the local variable *acl_sen* updates the accelerometer sensor (*acler_sensed*) and the accelerometer sensor flag (*acler_sensed_flag*) sets FALSE.

EVENT Acler_sensed
ANY
acl_sen
WHERE
grd1 : $acl\_sen \in \mathbb{N}$
grd1 : $acler\_sensed\_flag = TRUE$
grd1 : $HYT\_State = FALSE$
THEN
act1 : $acler\_sensed := acl\_sen$
act1 : $acler\_sensed\_flag := FALSE$
END

Finally, we have completed the formal specifications of the one- and two-electrode cardiac pacemaker. To find more detailed information about formalisation of all other operating modes of the pacemaker, see the published papers and technical reports [Méry 2011g; Méry 2009; Méry 2010c].

9.8 Model Validation and Analysis

There are two main validation activities in Event-B, and both are complementary for designing a consistent system:

- *consistency checking*, which is used to show that the events of a machine preserve the invariant, and *refinement checking*, which is used to show that one machine is a valid refinement of another. A list of automatically generated proof obligations should be discharged by the proof tool of the Rodin platform.

- *model analysis*, which is done by the ProB tool and consists in exploring traces or scenarios of our consistent Event-B models. For instance, the ProB may discover possible deadlocks or hidden properties that are not expressed by generated proof obligations.

This section conveys the validity of the model by using ProB tool [ProB ; Leuschel 2003] and Proof Statistics. "Validation" refers to the activity of gaining confidence that the developed formal models are consistent with the requirements, which expressed in the requirements document [Scientific 2007]. We have used the ProB tool [ProB] that supports *automated consistency checking* of Event-B machines via model checking [Clarke 1999]

and constraint-based checking [Jackson 2002]. Animation using ProB worked very well, and we have then used ProB to validate the Event-B machine. This tool assists us to find potential problems, to improve invariant's expressions in our Event-B models, for instance, by generating counter-examples when it discovers an invariant violation. ProB may help in improving invariant expression by suggesting hints for strengthening the invariant and each time an invariant is modified; new proof obligations are generated by the Rodin platform. It is the complementary use of both techniques to develop formal models of critical systems, where high safety and security are required. More errors are corrected during the elaboration of the specifications while discharging the proof obligations and careful cross-reading than during the animations. We have validated all operating modes of the pacemaker in each refinement of models. The pacemaker specification is developed and formally proved by the Rodin tool.

ProB was very useful in the development of the pacemaker specification, and was able to animate all of our models and able to prove the absence of error (no counter example exist). The ProB model checker also discovered several invariant violations, e.g., related to incorrect responses or unordered pacing and sensing activities. It was also able to discover a deadlock in two of the models, which was due to the fact that "clock counter" were not properly recycled, meaning that after a while no pacing or sensing activities occur into the system. Such kind of errors would have been more difficult to uncover with the prover of Rodin tool.

Model	Total number of POs	Automatic Proof	Interactive Proof
One-electrode pacemaker			
Abstract Model	203	199(98%)	4(2%)
First Refinement	48	44(91%)	4(9%)
Second Refinement	12	8(66%)	4(34%)
Third Refinement	105	99(94%)	6(6%)
Two-electrode pacemaker			
Abstract Model	204	195(95%)	9(5%)
First Refinement	234	223(95%)	11(5%)
Second Refinement	3	3(100%)	0(0%)
Third Refinement	83	74(89%)	9(11%)
Total	892	845(94%)	47(6%)

Table 9.2: Proof Statistics

Table 9.2 is expressing proof statistics for the formal development of the pacemaker using the Rodin platform. These statistics measure the size of the model, the proof obligations generated and discharged by the Rodin prover, and those are interactively proved. The complete development of the pacemaker system results in 892(100%) proof obligations, in which 845(94%) are proved automatically by the Rodin tool. The remaining 47(6%) proof obligations are proved interactively using the Rodin tool. In the Event-B models, many proof obligations are generated due to the introduction of new functional behaviors and their parameters (threshold, hysteresis and rate modulation) under the real-time constraints.

In order to guarantee the correctness of these functional behaviors, we have established various invariants in the stepwise refinements. As it can be seen, the abstract model in one electrode required by far the largest number of proofs: it is due to the large number of invariants (57), together with the number of events (26) which shows a size of the model. Similarly, large numbers of proofs are in the abstract model and the first refinement of two electrodes, where a large number of invariants (36 (abstract), 30 (refinement 1)), together with the number of events (41 (abstract), 43 (refinement 1)). It should be noted that the manual proofs were not difficult. Proofs are quite simple, and have been achieved with the help of *do case* operation. Guards of some events are very complex, so for proving invariants and theorems; we simplify guards using *do case*. The stepwise refinement of the pacemaker system helps to achieve a high degree of automatic proofs.

9.9 Closed-Loop Model (Heart & Cardiac Pacemaker)

The closed-loop model is a combined formal development of the cardiac pacemaker and the heart. A detailed description of the heart model based on electrocardiography analysis [Harrild 2000; Khan 2008; Bayes 2006] and cellular automata is given in the techniques and tools section. The heart model is based on logico-mathematical theory. The logico-mathematical based heart model is developed using refinement approach in Event-B modeling language [Abrial 2010; RODIN 2004]. In this investigation, we present a methodology for modeling a heart model, to extract a set of biological nodes (i.e. SA node, AV node, etc.), impulse propagation speed between nodes, impulse propagation time between nodes and cellular automata for propagating impulses at the cellular level. A main key feature of this heart model is a representation of all the possible morphological states of the electrocardiogram (ECG) [Bayes 2006; Artigou 2007]. The morphological states represent the normal and abnormal states of the electrocardiogram (ECG). The morphological representation generates any kind of heart model (patients model or normal heart model using ECG). This model can observe a failure of the impulse generation and a failure of the impulse propagation. This model is also verified through electro-physiologist and cardiac experts.

Formal specification of the cardiac pacemaker is expressed in this chapter. But this cardiac pacemaker is modeled without any biological environment like the heart system. In this section, we describe a closed-loop model of the cardiac pacemaker and formal model of the heart, where the cardiac pacemaker responses according to the functional behavior of the heart [Méry 2011f; Méry 2011i]. We have developed a combined model of the cardiac pacemaker and the heart together for developing the closed-loop model for the verification purpose. Refinement based formal development of the cardiac pacemaker, and the heart model is almost similar to the developed independent model. Fig. 9.6 represents a block diagram of the cardiac pacemaker and the heart system, where the cardiac pacemaker responses when it senses some values from the heart. In this system specification, the heart model simulates the functional behavior of the normal and abnormal heart. The heart model generates always a value, which is sensed by the cardiac pacemaker. The sensed value is known as threshold value for the atria and ventricular chambers corresponding to the time

interval. The pacemaker model uses this threshold value to response in form of pacing according to the different kinds of operating modes (see Table-9.1).

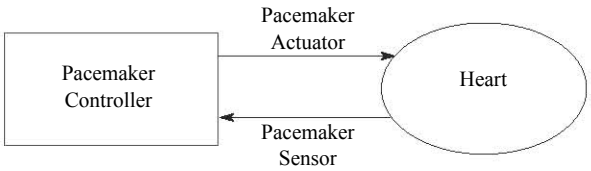


Figure 9.6: Closed-loop model

9.9.1 Formal Development of The Cardiac Pacemaker

The closed-loop model of the cardiac pacemaker is also based on *action-reaction* and *time patterns*. We apply the action-reaction and time patterns in modeling to synchronize the sensing and pacing stimulus functions of the pacemaker system in a continuous progressive time constraint. We present here only summary informations about each refinement of one- and two-electrode pacemakers and omit detailed formalization and proof details. The following outline is given about every refinement level to understand the basic formalism structure of the closed-loop model of the cardiac pacemaker. We have combined the model of the heart and the cardiac pacemaker to formalise the closed-loop model. To know more about detailed formalism see individual model of the heart system and the cardiac pacemaker. We have described here only summary informations about each refinement in form of very basic description of the heart modeling and the cardiac pacemaker modeling incremental refinement-based approach and omit detailed formalisation of events and proof details due to repetition of the formalism. To find more detailed information about developed formal model of the cardiac pacemaker and the heart model, see the published papers and research reports [Méry 2011i; Méry 2011g; Méry 2009; Méry 2010c].

9.9.1.1 Abstract Model: Introducing, Pacing and Sensing Activities with Normal and Abnormal Heart Behavior.

Abstract Model : Specifies the *pacing* and *sensing* under the real-time properties using *action-reaction* and *real-time* patterns for defining abstractly initial events like *Pace_ON*, *Pace_OFF*, *Sense_ON*, *Sense_OFF* and *tic* events with an observation model of the heart, which specifies a heart state in form of *true* and *false*, where *true* represents a normal rhythm and *false* represents an abnormal rhythm of the heart.

9.9.1.2 Refinement 1: Introducing *threshold* in Cardiac Pacemaker and Impulse Propagation in the Heart System.

This refinement is a conduction model of the heart, which specifies beginning of the impulse propagation at the SA node and ending of the impulse propagation at the Purkinje fibers in the both left and right ventricles. Refinement expresses step by step impulse propagation through all landmark nodes, where the electrical impulse must pass through a

number of intermediate landmark nodes before reaching to the terminal nodes (C, G, H). This refinement also specifies impulse propagation between landmark nodes with a global clock counter to model a real-time system to satisfy the temporal properties of the impulse propagation, where several events are introduced to simulate the impulse propagation into the heart conduction network. New events are formalizing impulse flow between two landmark nodes separately; for instance, the electrical impulse moves from SA node (A) to AV node (B). This refinement also introduces a logical clock to synchronise all the states of the heart system and checks the heart states under a required time length in the conduction network.

In the part of the cardiac pacemaker development, this refinement introduces additional features for filtering the exact sensing value through the pacemaker's sensor by introducing standard threshold constants for the both atrial and ventricular chambers, and some new events are introduced as a refinement of *skip* for capturing the sensors value from the single or both chambers. The threshold value for the cardiac pacemaker is generated by the formal model of the heart. The heart model continue produced some threshold value using an event, which is used by the pacemaker model. A pacemaker has a stimulation threshold measuring unit which measures a stimulation threshold voltage value of the heart and a pulse generator for delivering stimulation pulses to the heart. The pulse generator is controlled by a control unit to deliver the stimulation pulses with respective amplitudes related to the measured threshold value and a safety margin.

9.9.1.3 Refinement 2: Introduction of Hysteresis for cardiac pacemaker model and Perturbation the Conduction for the heart model.

This is the perturbation model of the heart, which specifies perturbation in the heart conduction system and helps to discover exact block into the heart conduction system. This refinement introduces a set of possible blocks in the heart conducting system. These blocks can occur into the conduction network and give trouble into electrical impulse propagation. A set of landmark nodes partition the different regions for all possible heart blocks.

In the cardiac pacemaker model, this refinement introduces a *hysteresis* operating mode to prevent constant pacing. The *hysteresis* is a programmed feature whereby the pacemaker paces at a faster rate than the sensing rate. For example, pacing at 80 pulses a minute with a hysteresis rate of 55 means that the pacemaker will be inhibited at all rates down to 55 beats per minute, having been activated at a rate below 55, the pacemaker then switches on and paces at 80 pulses a minute [Malmivuo 1995; Hesselson 2003]. The main purpose of hysteresis is to allow a patient to have his or her own underlying rhythm as much as possible. In this refinement, only new variables are introduced for applying *hysteresis* operating modes.

9.9.1.4 Refinement 3: Introduction of Rate Modulation for the Cardiac Pacemaker Model and a Cellular Model for the Heart system.

This is a simulation model of the heart, which introduces an impulse propagation at the cellular level using cellular automata. The cellular level modeling is used to model the

electrical impulse propagation at the cell level. The formalisation uses cellular autamata theory to model the micro-structure based cell model. To formalise the cellular automata, we introduce mathematical properties (see Definition 2 and 3 in Chapter 5) in a context model.

In the part of the cardiac pacemaker model, we describe a rate adapting pacing technique of the cardiac pacemaker. Rate modulation term is used to describe the capacity of a pacing system to respond to physiologic needs by increasing and decreasing pacing rate. The rate modulation mode of the pacemaker can progressively pace faster than the lower rate, but no more than the upper sensor rate limit, when it determines that the heart rate needs to increase. This typically occurs with exercise in patients who cannot increase their own heart rate. The amount of increasing rate is determined by the pacemaker based on maximum exertion performed by the patient. This increased pacing rate is sometimes referred to as the *sensor indicated rate*. When exertion has stopped, the pacemaker will progressively decrease the paced rate down to the lower rate. Two new events are introduced as refinement of $skip$ for increasing and decreasing the pacing rate using an accelerometer. We have given the following proof static table of the closed-loop model:

Model	Total number of POs	Automatic Proof	Interactive Proof
Closed-loop model of One-electrode pacemaker			
Abstract Model	304	258(85%)	46(15%)
First Refinement	1015	730(72%)	285(28%)
Second Refinement	72	8(11%)	64(89%)
Third Refinement	153	79(52%)	74(48%)
Closed-loop model of Two-electrode pacemaker			
Abstract Model	291	244(84%)	47(16%)
First Refinement	1039	766(74%)	273(26%)
Second Refinement	53	2(4%)	51(96%)
Third Refinement	122	60(49%)	62(51%)
Total	3049	2147(70%)	902(30%)

Table 9.3: Proof Statistics

The Table 9.3 is expressing proof statistics for the formal development of the closed-loop model of the cardiac pacemaker with the heart system. These statistics measure the size of the model, the proof obligations generated and discharged by the Rodin prover, and those are interactively proved. The complete development of the closed-loop model of the cardiac pacemaker results in 3049(100%) proof obligations, in which 2147(70%) are proved automatically by the Rodin tool. The remaining 902(30%) proof obligations are proved interactively using Rodin tool. Integration of the heart model and the cardiac pacemaker model generates lots of extra POs. Main reason of these new POs is to use common variables in both models. After integration of the both systems, invariants corresponding to the common variables generate new POs. For example, current clock counter variable (*sp*) is common in both the heart and pacemaker models, which has been used in events of the heart and pacemaker model. The combined invariants of the heart and pacemaker mod-

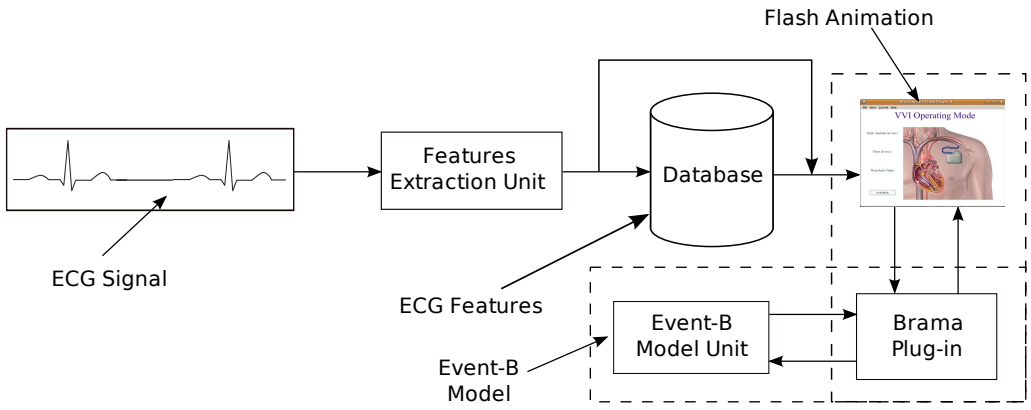


Figure 9.7: Real-time animation of cardiac pacemaker

els generate new POs corresponding to the current clock counter variable (*sp*). The whole system represents functional properties of the cardiac pacemaker operating modes under the biological environment in the heart. The heart model represents normal and abnormal states of the heart, which is estimated by the physiological analysis. In order to guarantee the correctness of these functional behaviors, we have established various invariants using stepwise refinement.

9.10 Real-Time Animation Using Pacemaker Case Study

This section shows an applicability of the real-time animator [Méry 2010b] through animation of formal models of the pacemaker using real-time data sets. Fig.9.7 represents an implementation of the given architecture for the formal model of a cardiac pacemaker case study. We have mainly used this case study to experiment on our proposed architecture, which enables the animation of a proved specification with real-time data set without generating the source code in any target language. According to the proposed architecture (see Fig.9.7) for this experiment, we have not used any data-acquisition device to collect the ECG (electrocardiogram) signal. We have done this experiment in off-line mode, means we have used our architecture to test the real-time data set of ECG signal that is already collected. ECG signal collection and features extraction in on-line mode is too expensive due to complex data-acquisition process and limitation of feature extracting algorithms. So, we have used the ECG signal and feature extraction algorithms for our experiment from the MIT-BIH Database Distribution [MIT-BIH].

We have downloaded the ECG signal from ECG data bank [MIT-BIH]. The ECG signals are freely available for academic experiments. We have applied some algorithms to extract the features (P, QRS, PR, etc.) from the ECG signal and stored it into a database. We have written down some Macromedia Flash scripts to interface between Flash tool and Brama component, to pass the real data set as a parameter from a database to the Event-B model. No any tool is available to interface between database and the Event-B model. Extra Macromedia Flash script coding and the Brama animation tool help to test the Event-B formal model of the cardiac pacemaker on the real-time data set. We

have designed an animated graphic of heart and pacemaker in Macromedia Flash, where this animated model represents the pacing activity in the right ventricular chamber. This animated model simulates the behaviour of heart according to the bradycardia operating modes and animates the graphic model. The animation of the model is fully based on the Event-B model. Event-B model is executing all events according to the parametric value. These parametric values are the extracted features from the ECG signal, which are passing into the Event-B model.

Fig.9.7 represents an implementation of proposed architecture on the formal model of a cardiac pacemaker case study. According to the architecture, data-acquisition unit collects the ECG signal and features extractions are done by the feature extraction or parameter estimation unit. The extracting features are stored in the database as the XML file format. Macromedia Flash tool helps to design the animated graphics of the heart and pacemaker. In next unit, Brama plug-in helps to communicate between animated graphics and Event-B formal model of the single electrode cardiac pacemaker. Finally, we have tested a real-time data set in the formal models without generating the source code with the help of Brama existing animation tool.

One- and two-electrode Pacemaker's pacing and sensing behaviors are validated through cardiologist experts using real-time data; ECG signal. We have found some unexpected behaviors of the formal model according to the cardiologist experts in visualization. We have modified the pacemaker formal model according to cardiologist experts and verify through the real-time animation tool. So, we consider that the real-time animation tool has a very important role in the area of development of the formal methods, and it can help to obtain a trust-able formal model, which can be helpful to obtain the certification assurances [Méry 2010d; ISO ; IEEE-SA ; FDA ; CC ; NITRD 2009].

9.11 Code Generation for A Cardiac Pacemaker using EB2ALL Tool

We have presented a proof-based an incremental formal development of a cardiac pacemaker in [Méry 2011e; Méry 2011g; Méry 2009; Méry 2010c] using our proposed tool and techniques. This section presents an automatic code generation from developed and proved formal specification of a cardiac pacemaker. We now illustrate the use of EB2C, EB2C++, EB2J and EB2C# tools [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b] by means of the automatic generation of C, C++, Java and C# codes for the cardiac pacemaker system described with EVENT B in [Méry 2011d; Méry 2011a]. This tool has a technique of automatic support of safety assurance of a generated code. To achieve a verified source code of the cardiac pacemaker, we have done further refinement of the concrete models of the cardiac pacemaker using a new context, which has some data ranges (see Table 7.1) corresponding to the programming languages. This context file provides deterministic ranges for all kinds of data types. This refinement makes the model deterministic and generates some proof obligations due to defining the fixed data ranges of all constants and variables of the cardiac pacemaker model. The generated proof obligations are discharged by automatic as well as manual, and all these proofs are necessary to verify the specification in

order to guarantee the consistency and correctness of the system. We have discharged all the generated proof obligations before generating the source code. This level of refinement complies system specification abstractly. Now, we move to the next level of code translation methodology as to pass the concrete model for continuing translation process.

The code translation from Event-B formal specification into any programming language using EB2ALL is straightforward using a set of plugins (EB2C, EB2C++, EB2J and EB2C#) [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b]. The main idea is to translate an Event-B model into any programming language code using the last concrete model. The EB2ALL tool generates programming language files corresponding to the concrete models. A generated source file using EB2ALL tool has a basic structure: a set of constants, variables and functions. A set of constants and variables are extracted from the context and machines sections of the Event-B model of the cardiac pacemaker, respectively. Data type of a constant is defined as an axiom in Event-B model. Similarly, data type of a variable is extracted from the invariant section of the model. Initial value of the constants and variables are initialized, if their initial values are declared. A set of constants and variables are given as follows, which are excerpted from the translated 'C' codes of the cardiac pacemaker model.

```
enum status {ON,OFF}; /* Enumerated definition */
const int FixedAV=90; /* Integer in range 70-300 */
const int LRL=60; /* Integer in range 30-175 */
const int ARP=200; /* Integer in range 50-175 */
const int URL=120; /* Integer in range 50-175 */
const int VRP=250;
const int PVARP=150;
const int V_Blank=50;
...

enum status PM_Actuator_V; /* Enumerated type variable */
enum status PM_Sensor_V; /* Enumerated type variable */
unsigned long int Thr_V; /* Integer in range undefined */
unsigned long int AV_Count; /* Integer in range undefined */
BOOL AV_Count_STATE;
unsigned long last_sp;
unsigned long int sp;
unsigned long int Pace_Int;
...
```

A set of functions are extracted equivalent to a set of events of the pacemaker formal model. All the events of Event-B are translated into equivalent programming language functions. An event INITIALIZATION is a programming language function, which initialize default values of all the variables. An event of Event-B model has fixed organization of the internal components; local variables, guards (pre-conditions) and actions. An event may contain some local variables. The global constants and variables are declared on the top of the programming language source file, while local variables are declared within the function body. All events of a formal model is translated as a set of programming language functions. This function has the similar structure as an event. During the translation

of the events, the guards are translated into equivalent to 'if' statement using logical conjunction, disjunction, implication and equivalence. Each guard represents into a separate 'if' statement like nested 'if' structure. All these guards represent a set of preconditions, which are required to satisfy for executing the action predicates. All action predicates of a formal model event are directly translatable equivalent into programming language assignment expressions. The EB2ALL tool is capable to analyse the syntax of Event-B guards and actions predicate. In the cardiac pacemaker formal model, their predicates are simple, which are obtained through several refinements. All pre-conditions or guards are required to be TRUE for executing all actions. However, despite being a complex system, the pacemaker pre-conditions are fairly simple to calculate. If all guards are true, then the action's predicates execute and return TRUE for successful execution of the function. If any 'if' condition false, then the function returns FALSE and action's part of the function does not execute.

```
...
BOOL Actuator_ON_V(void)
{
  /* Guards No. 1*/
  if(PM_Actuator_V == OFF){
  /* Guards No. 2*/
  if((sp == Pace_Int) || ((sp < Pace_Int) &&
  (AV_Count > V_Blank) && (AV_Count >= FixedAV))){
  /* Guards No. 3*/
  if((sp >= VRP)&&(sp >= PVARP) && (sp >= URI)){
  /* Actions */
  PM_Actuator_V = ON;
  last_sp = sp;
  return TRUE;
}}}
  return FALSE;
}
...
```

To make the generated code executable, the EB2ALL tool generated an $Iterate$ function that contains a list of all functions as in form of a function call. Another function is a main body of the program like $main()$ in 'C', which calls $Iterate$ function. These two extra functions are used to compile and execute the generated code.

The source code is automatically generated in any programming language (C, C++, Java and C#) from the verified specification in less than five seconds. The generated code resulted in over 5000 lines in all operating modes. Here, we have presented a brief overview of the translation from the Event-B specification of the cardiac pacemaker formal model into 'C' using EB2C tool. Based on this translation, we were able to automatically generate 'C' code and execute a simulation of the pacemaker.

9.12 Discussion and Conclusion

9.12.1 Discussion

New development methodology is successfully applied for developing the cardiac pacemaker from requirement analysis to code implementation. The whole system development life-cycle is based on formal techniques. The complete system is designed using different kinds of tools related to the formal techniques. The Event-B modeling language is used for formalizing the pacemaker system using refinement techniques. Each level of refinements is validated through the ProB model checker and the real-time animator for verifying the correctness of the system behaviors against requirements and according to the medical experts, respectively. If any error is discovered during verification, validation or domain experts reviews, then the pacemaker specification is modified and again follow the verification, validation and domain experts reviews. This process is continued applied in a loop until not find the correct proved formal specification of the cardiac pacemaker. The verification, validation and domains experts reviews are applied on each refinement level for modeling the whole system. To handle the complexity of a system according to the refinements, we have used the refinement chart to model the cardiac pacemaker system. The refinement charts of the pacemaker present integration architecture of the system in form of all possible operating modes. Some operating modes are an extension of the existing operating modes, its clearly expresses-able from the refinement charts. This technique is not only for code integration, but also it helps for analyzing the operating modes and code structuring of the system. Finally, we have used the tool EB2ALL for generating the source code into multiple languages (C, C++, C#, and Java) from the formal specifications. In this development process, we have not considered the safety assessment approach.

According to the existing development life cycle, we use formal methods only on the selected part of the system for verifying the correctness of the system against requirement. No formal methods are likely to be suitable for describing and analyzing every aspect of a complex system; a practical approach is to use different methods in combination. In this thesis, we have provided some possible solutions for emerging problems in area of software engineering related to the development of critical systems, where we have proposed a development life-cycle and associated a set of techniques and tools to develop the highly critical systems using formal techniques from requirements analysis to automatic source code generation. There is not a set of supporting tools, which can be used for system development using only formal techniques. We have developed a set of new tools, which support a rigorous framework for the system development and finally; we have applied this new development life-cycle methodology and associated tools for developing the cardiac pacemaker system for assessing the usability of our proposed approach. This methodology use only refinement approach to build the complete system and each refinement level is verified using different techniques. Last level of the system is the concrete model, which has been used for producing the source code. The code generation tool EB2ALL is very simple in use, which can generate the optimized codes for future use. The process for system development is user friendly, but a developer has required the strong knowledge of formalisation and refinement techniques to build the correct system using this new system

development methodology and associated techniques and tools.

9.12.2 Conclusion

In this chapter, we have presented the pacemaker specification, one of the challenges proposed by the Verified Software Initiative [Hoare 2009]. We have developed the formal model of the pacemaker system in Event-B and discovered the exact functional behavior of the pacing and sensing events. Our approach for formalizing and reasoning about action-reaction is based on real-time as a pacemaker system. The pacemaker case study suggests that such an approach can yield a viable model that can be subjected to useful validation against system-level properties at the early stage of the development process. The proposed techniques based on development patterns intend to assist in the design process of the system where correctness and safety are important issues.

A series of high confidence medical devices of increasing scope and complexity will follow the pacemaker system. Main advantage of proposed development methodology and a set of associated techniques and tools is the ability to develop the whole system from requirement analysis to code generation. Proposed methodology exploits the advance capabilities of the combined approach of formal verification and model validation using a model-checker, use of real time animation to test system behavior and, finally automatic source code generation from a verified formal model in order to achieve the considerable advantages for a critical system design.

The proposed approach has also involved the use of the real-time animator for executing formal specification to validate the actual requirements. The main objectives of this real-time animator [Méry 2010b] are to promote the use of such kinds of tool to bridge a gap between software engineers and stakeholders to build quality system and to discover all the ambiguous information from the requirements. Moreover, this tool helps to verify the correctness of behavior of the system according to the stakeholders requirements. The combined approach of the formal verification and real-time animation allows the systematic development of a clear, concise, precise and unambiguous specification of a software system and enables to the software engineers to animate the formal specification at the early stage of the development. The formal specification animation is supported by both software engineers and stakeholders. Our case study on cardiac pacemaker illustrates the potential value which is a formal specification, and its subsequent animation can bring to the comprehension and clarification of the informal requirements.

System integration methodology using refinements charts (see Fig. 9.4, Fig. 9.5) are also used for system development, which helps a code designer to improve the code structure and code optimization, and the code generation for synthesizing and synchronizing the software codes of a critical system like the cardiac pacemaker. In the pacemaker case study, each operating mode (see Table 9.1) have different kinds of functional requirements, and all the operating modes are decomposed in the refinement chart using multiple refinements. The refinement chart and formal specifications support more systematic and error-free designing and implementation rather than other traditional approaches of system designing and implementation. Here, the refinement chart is tightly coupled with the formal specifications. A set of requirements are formally represented in the specifications.

The complexity of formal specifications is the amounts of proof obligations (see Table 9.2). Therefore use of the refinement chart, and formal specifications states the correctness of the system design and implementation.

The refinement chart specially covers component-based design frameworks and decomposition, integration of the critical infrastructure and device integration. We can see from our pacemaker case study that all these claims help to design error-free system and different phase of the pacemaker has been shown by refinements in form of formal development as well as refinement charts. We have presented evidence that such an analysis is fruitful for both formal and non-formal group of people. The second observation from our experiments is that the development of multiple models helped us not only find errors in the requirement documents but also gave us an opportunity to better understand intricate requirements such as the control algorithm of a medical system. Moreover, we believe that the effort needed is commensurate with the benefits we derive from developing the multiple models. An ideal critical system has the following characteristics:

- Embedded real-time system's design.
- To obtain the certification for providing the higher safety integrity level.
- Helps to domain experts to analyse work process guidelines;
- Sufficient complexity that traditional methods, such as testing and code reviews, are inadequate to establish its correctness.
- Model-based development and component-based design frameworks.
- Animator assists to regulatory agencies and helps to meet ISO/IEC and IEEE standards;
- Ability to monitor a real-time environment using animator at animation time and analyse the requirements, violations of goals, expectations on the environments, and domain properties;
- Real-time animation of a specification supplements inspection and reasoning as means for validation. This is especially important for the validation of non-functional behaviour;
- Real-time animation technique is available in early phase of the system development life-cycle, which can be used to correct validation errors immediately, without incurring costly redevelopment;
- Infrastructure for critical system integration and inter-operation.
- System integration of critical infrastructure.
- Possibility of annotating models for different purposes, (e.g., directing synthesis or hooking to verification tools).
- To discover the complex situation using refinement approach.

- Decomposition of the complex system into different independent subsystems.
- To reduce the gap between software engineers and stakeholders requirements using real-time animator and easy to explain model behavior to domain experts as well stakeholders;
- Ambiguous and incomplete requirements can be clarified and completed by hands-on experience with the specifications using our approach.

Code generation is a process, which is used to transform a formal specification into any programming language like C, C++, C# and Java etcetera. Code generation from the verified formal model is our main objective. For generating the source code into different kinds of programming languages (C, C++, Java and C#), we have used a tool EB2ALL [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b]. We have developed a set of plugin tools [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b], which provides fully automatic code generation from Event-B formal specification into programming languages. The adaptations of the translation rules are required more complete experiments, especially with the large formal models for checking the impact on the execution time for some specific platforms. Finally, we have shown a satisfactory result and demonstrate the ability to generate automatically source code from EVENT B specification of the cardiac pacemaker in C, C++, Java and C# languages, which are comparable to a code written by hand with ordinary programming languages. The gains rely then on the guarantees provided using formal methods and on the certification level which can be obtained by this way. As far as we know, only few formal methods support code generation, which is as time/space efficient as handwritten code.

Proposed development methodology and associated techniques and tools enable us to design a new environment for medical device modeling and simulating and offers to obtain that challenge of complying with FDA's QSR and ISO's 13485 quality system directives [Keatley 1999; NITRD 2009].

In order to assess the overall utility of our approach, a selection of the results of the formalisation and verification steps have been presented to a group of pacemaker developers (French-Italian based pacemaker development company). The developers are satisfied by the results of pacemaker development using all proposed approaches in sense of incremental development, real-time animation of formal model, integration of hardware and softwares and automatically code generation approach from verified formal specifications. They are really agreed on the refinement charts for showing operating mode relation and their mode transitions. Based on the experiment described above and our conclusions we are convinced of the usefulness on certain areas, and therefore, we are considering to use all these methodology and tools, which are very helpful to design not even in a medical domain but also for other industrial domains, such as avionic and automotive domains.

CHAPTER 10

Adaptive Cruise Control (ACC)

"A theory is something nobody believes,except the person who made it. An experiment is something everybody believes,except the person who made it."
(Albert Einstein)

In this chapter, we focus on the formalization of the hybrid control system using *"separation of concern"* based on refinement approach. We have proposed an approach for formalising the control system and verifying all their required safety properties. For verifying a controller system using formal methods, we have decomposed the whole system in two parts; first is *control function* and second is *control laws*. We have formalised all the controller functions like *uninterpreted functional* blocks. The uninterpreted function blocks have same input and output parameters as the original controller functions, and we have assumed that the original control functions are correct, which are provided by the control engineers. The control laws are represented by the discrete control logic. To assess the effectiveness of our proposed development methodology and the associated techniques and tools in the area of automotive domain, we have selected a real-life hybrid control system; Adaptive Cruise Control (ACC). This chapter presents a development of the adaptive cruise control using our proposed development life-cycle methodology from requirement analysis to automatic code generation. In this development, we have used formal verification to verify the correctness of the requirements and model checking to verify the correctness of the system behaviors according to the domain experts (i.e. automotive engineers), and finally EB2ALL for generating the codes into several languages. Refinement charts are used to handle the complexity of the system, where it helps to organize the code structure according to the different switching modes. Formal models are expressed in the Event-B modeling language, which generates proof obligations for checking their internal consistency and safety properties of the system. The Rodin tool is used for formalizing the whole system and ProB model checker is used to validate the system requirements and system behaviors. In addition, we have generated Simulink model from the proved Event-B specification.

10.1 Introduction

A promising and challenging application area of formal methods in the automotive domain is specifying and designing software for embedded systems, which are growing more and

more complex [Bowen 1993]. It has become a decisive factor in the automotive industries. Therefore, a tool-supported software technology adapted to a vehicle specific need is required, which facilitates a timely and cost effective development.

Increasing demands for high quality, safety requirements, and the shortcomings of the informal techniques applied in traditional development motivate an integrated use of semi-formal and formal methods, thus facilitating a high degree of automation and tool support. In fact, ensuring safety is the primary preoccupation of the automation industries.

Software is a key factor for innovative functions in automotive industries. The largely growing complexity of the functions realized by embedded software requires a systematic and effective software-development process. To ensure continuity within the current development departments, all improvements must be based on established development cycles. Approved aspects must be preserved and inherent limitations need to be overcome.

The controller is most important part for automotive industries. PID control is by far the most common way of using feedback in natural and man-made systems. PID controllers are commonly used in industries, and a large factory may have thousands of them, in instruments and laboratory equipment. 90% of the industrial controllers are Proportional-Integral-Derivative (PID) controllers [Knospe 2006]. Most controllers do not use derivative action. Practical experiences have shown that PID controls with appropriate filtering and modifications – like gain scheduling and adaptive tuning satisfy many automotive controller requirements [Hrovat 1991].

This work presents a modeling of an informal description of the automotive system into a formal language, which is based on PID controllers, with the aim of analysing the essential requirements. In addition to the advantages of such a kind of formal verification, making these descriptions more formal can serve to expose problematic parts in the requirements. Based on the formal semantics of the system, the analysis techniques should allow for the determination of consistency (no contradictions) and correctness (objectives are satisfied).

The contribution of this work is to give a complete idea of formal development of a hybrid control system. This approach can be applicable to designing any new hybrid control system. Here, we present an incremental proof-based development to model and verify such interdisciplinary requirements in the Event-B [Cansell 2007; Abrial 2010; Abrial 1996a]. Finally, we will also discuss modeling of hybrid controllers; similar methods can be used to model many other controllers. In order to assess the feasibility of this approach, we have carried out a case study in the formalisation and the verification of the ACC (Adaptive Cruise Control) system. In this chapter, throughout our case study of the ACC system, we have shown effectiveness of our proposed approach (see Part-I) in form of formal specification, verification and automatic code generation from the proved formal specifications. The ACC system is very complex and ambiguous. The controller based ACC models must be validated to ensure that they meet requirements. Hence, validation must be carried out by both formal modeling and domain experts. The current work intends to explore those problems related to the modeling of controller systems. Moreover, an incremental development of the hybrid control system helps to discover the ambiguous, incomplete or even inconsistent elements in the ACC.

The outline of the remaining chapter is as follows. Section 2 contains basic intro-

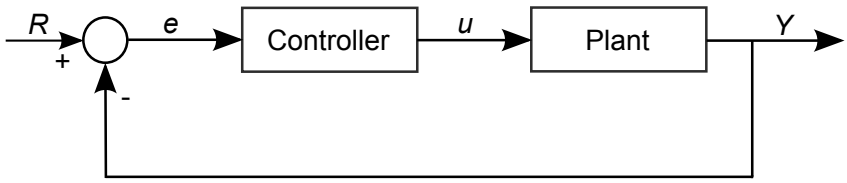


Figure 10.1: Block diagram of a unity feed back system

duction of PID controllers. An approach for modeling control design is presented in Section 3. We give a brief outline of the ACC in Section 4. Section 5 presents development of the ACC system using refinement charts. In Section 6, we explore the incremental proof-based formal development of the ACC system. The verification results are analyzed by statistical proof in Section 7. Section 8 represents transformation from Event-B formal model into equivalent *Simulink* model. Automatic code generation into C, C++, Java and C# from proved formal specifications is done using the code generation tool EB2ALL [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b] in Section 9. Finally, in Section 10, we conclude the chapter with some discussions.

10.2 PID Controller

Control theory is an interdisciplinary branch of engineering and mathematics, that deals with the behavior of dynamical systems. The desired output of a system is called the reference. When one or more output variables of a system need to follow a certain reference over time, a controller manipulates the inputs to a system to obtain the desired effect on the output of the system. A related theory known as perceptual control theory has been used to model the living systems on the premise that outputs are manipulated to obtain the desired effect on the input to the system [Bagaria 2010].

Proportional integral derivative (PID) controllers are still the most popular ones in the automotive industries. They are simple in structure, reliable in operation and robust in performances. One key factor for their success is that they act in the processes under control in a manner closely similar to human's natural responses to be outside stimuli, that is the combined effects of spontaneity (proportional action), post training (integral action) and projection into future (derivative action). Users can modify the dynamic properties of this controller by adjusting the three parameters: proportional, integral, and derivative.

In general, the synthesis of PID controller can be depicted by Fig. 10.1, where e is the error, R the reference value, and Y the process output. PID can be described by

$$u = K_p e + K_i \int e\, dt + K_d \frac{de}{dt} \tag{10.1}$$

A proportional controller (K_p) has the ability to reduce the rise time but cannot eliminate the steady-state error. An integral control (K_i) has the capability of removing the steady-state error, but may worsen the transient response. A derivative control (K_d) has the power to increase the stability of the system, reduce the overshoots and improves the

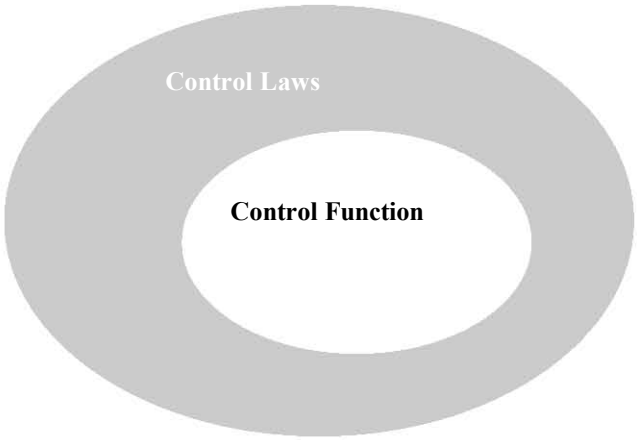


Figure 10.2: Control Function and Control Laws

transient responses. Tuning a system means adjusting three multipliers K_p, K_i and K_d adding in various amounts of these functions to get the system to behave the way you want.

10.3 Overview of Methodology

Formal verification of control systems is very crucial issue due to erratic behavior of the control functions. As far as we know that there are few tools are available based on theorem proving or model checking to verify the hybrid control systems [Platzer 2008; Frehse 2008; Ábrahám-Mumm 2001]. Formal representation of hybrid control systems, we have applied a rather different approach than the existing approaches. Refinement based formal methods techniques are used for verification of hybrid control systems. We have decomposed the whole system in two parts; first is *control function* and second is *control laws*. Block diagram (Fig. 10.2) represents a relationship between control function and control laws. Both are tightly coupled.

The control function is a transfer function based on control theory, which is designed by a control engineer. This function made from the combination of other three functions PID (P-Proportional, I-Integral, D-Derivative). The control function has some K_p, K_i and K_d multipliers, which are obtained through tuning. This tuning process updates parameters of the optimizing control function to achieve a best fit to current plant behavior, and updates parameters of the direct control function to achieve a good dynamic response of the closed-loop system. Another most important part of the hybrid control system is the control laws (Fig. 10.2). It represents sequencing, situation analysis and decisions, and integration of the control functions. All these are based-on software design principles.

In our formal development of the hybrid control system, we have assumed that all kinds of controller functions, which are provided by a control engineer are correct. Means, we have not required to verify it. However, in our formal modeling approach, we have defined to control functions as the *uninterpreted functions* for implementing an exact behavior of a given control function in the formal logic. The *uninterpreted functions* have a set of inputs and outputs, which have been taken as assumptions. Behavior of these *uninterpreted func-*

tions are similar to the original control functions. The control laws represent all discrete properties of a system. The composition of control functions and control laws represent complete behavior of a controller system. In summary, we can say that a software engineer develops a skeleton which models all control laws in discrete form and to define all the control functions in form of the *uninterpreted functions*. Finally, a control engineer just substitutes the control equation in place of an *uninterpreted function*.

Fig. 10.3 represents a controller architecture of the Adaptive Cruise Control (ACC). In this block diagram *ACC Control*, *Cruise Control*, *Driver Control* and *PLANT* are the uninterpreted control functions, which are represented using the shadow box. The input parameters of the *uninterpreted function ACC Control* (FACC) are the actual distance (*actGap*) and set distances (*setGapDist*) and output parameter is desired or reference speed (*refSpeedACC*). Similarly, input parameters of an *uninterpreted function Cruise Control* (FCC) are the actual speed (*speed*) and reference speed (*refSpeedACC*), and the output parameter is an ACC throttle angle (*ACC_th*). Another cruise control function without ACC control function has input parameters like actual speed (*speed*) and set speed (*setSpeed*), and output parameter is a CC throttle angle (*CC_th*). An uninterpreted function *Driver Control* has input parameter driver throttle angle and output parameter is the driver throttle (*Dri_th*). Finally, any throttle angle (ACC throttle angle, CC throttle angle and driver throttle) (*throttle*) is an input of the plant function according to the system requirements. There are multiple inputs from the sensors like sensed speed and forward vehicle speed. The distance error is the difference between desired distance and actual distance. Similarly, the velocity error is also the difference between desired speed and actual speed of the host vehicle. All other rest of the data flows represent as a feedback network to control the entire system. We have found some functionl behavior of the ACC system as follows:

When no forward vehicle,

$$Throttle\ Angle = FCC(setSpeed, speed)$$

In ACC mode,

$$Throttle\ Angle = FCC(FACC(setGapDist, actGap), speed)$$

In Driver control mode,

$$Throttle\ Angle = FD(Dri\_th)$$

We have applied a refinement based formalisation of the ACC using the Event-B modeling language. After verification of the complete system, we can replace the *uninterpreted functions* (*FACC*, *FCC*, *FD* and *PLANT*) using some control equations, where we have considered that the input and output parameters of a control function are equivalent to a new control equation, which can be replaced at the time of final implementation. This kind of approach helps to model the closed-loop system for verifying the desired behavior of the system without using the complexity of the PID functions. All PID functions are defined abstractly to model the final system.

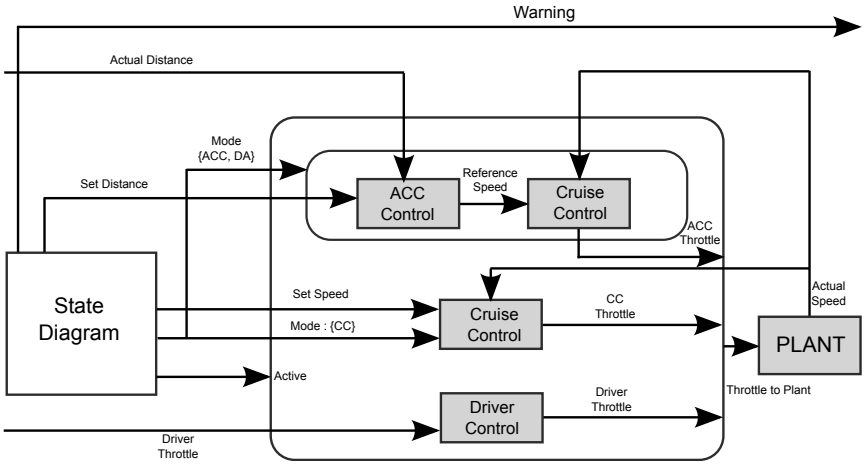


Figure 10.3: Adaptive Cruise Control

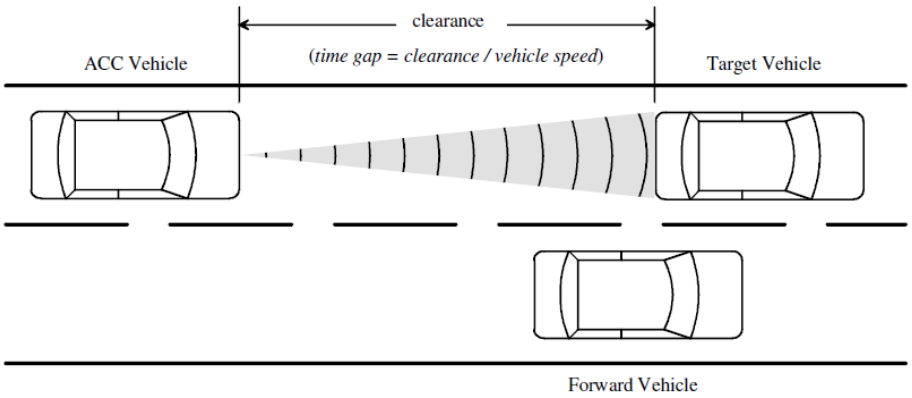


Figure 10.4: ACC Vehicle Relationship

10.4 Informal Description of ACC

Adaptive cruise control (ACC) system maintains the driver-set vehicle speed as well as tries to regulate the required pre-selected time gap between forward vehicle and the ACC vehicle for the safety reasons and avoids accidents. A radar system is attached to front of the vehicle to detect any other vehicle in the same lane or path. If any slower moving vehicle is detected, then ACC starts to control the current speed of the vehicle using throttle or braking system. If a system detects that the forward vehicle is no longer on the ACC vehicle's path, the ACC will accelerate the vehicle back to its set cruise control speed. Automatic speed down and speed up operations allow the ACC vehicle to maintain vehicle's speed according to the traffic environment without interrupting driver. Braking system carried out by the ACC can usually reach up to 30% of the vehicle maximum deceleration. When stronger deceleration is needed, driver is warned by auditory signal and display warning message on the screen. Driver can override anytime ACC system to take control of the vehicle.

10.4.1 Basic Components of ACC

Fig. 10.4 depicts a basic functionality of the ACC system. Basic components and transitions definitions of various states are given as follows:

1. **ACC off :** It is an initial state of ACC. Switch *On* can transit ACC into *Standby* operating mode. Any time, switch *Off* may transit into *off* state to stop the ACC system.

2. **ACC Standby :**When a switch *On* is pressed to start functioning of ACC then system enters into *Standby* operating mode. In this state, the system is ready for activation by a driver. If ACC is already in *Active* state and driver use any break, ACC automatically switches back to *Standby* operating mode. Basic conditions to switch back to *Standby* mode are as follows:

   - Brake pedal is pressed, or
   - Vehicle speed is less than 30 mph

3. **Set speed and time gap :** The desired cruise control travel speed (*setSpeed*) and time gap (*setGapTime*) are set by a driver.

4. **ACC Active state :** ACC transits from *Standby* mode to *Active* mode when a driver selects both desired speed (*setSpeed*) and time gap (*setGapTime*). For this transition some additional conditions are required to satisfy as follows:

   • Brake pedal should not be touched, and vehicle's speed should be $>= 30$ mph

   • When ACC is operating in *Standby* mode (after deactivation), then the *resume* button only can be used to switch back into *Active* mode.

5. **ACC with CC set speed :** Sub-state of *Active* state in which no forward vehicles are present and ACC is controlling the vehicle speed to be desired selected speed (*setSpeed*).

6. **ACC speed down** Sub-state of *Active* state in which a forward vehicle is present and the actual time gap (*actGap*) between host vehicle, and front vehicle is less than the desired time gap (*setGapTime*) then ACC applies throttle and brake for controlling the current vehicle speed.

7. **Time gap :** Current time interval between host vehicle[1] and forward vehicle. Time gap is related to the distance (or clearance) between host vehicle and forward vehicle and actual speed of the host vehicle. Where,

   Time gap = current distance/current ACC vehicle speed.

[1] ACC vehicle and host vehicle are used to reference same vehicle

8. **Change ACC speed and time gap :** When ACC is operating in *Active* state, driver can change desired speed (*setSpeed*) and desired time gap (*setGapTime*), which can be increased or decreased by two switches (*setSpeed* or *setGapTime*).

9. **ACC Failed State :** If any kind of fault (say, sensor or actuator fault) is detected in the ACC, then the ACC switches into *Failed* state meaning that repair work is required to rectify the faults.

10. **Overridden State :** During *Active* state of ACC, if throttle is pressed by a driver, then *Active* state moves to be new *Overridden* state and if throttle is released, then the ACC again returns to the *Active* state.

10.4.2 Basic I/Os of ACC

Fig. 10.3 describes basic input and output of the ACC system. There are several input parameters; those are required for controlling the ACC system. These parameters are given as follows:

1. **ACC ON/OFF :** It is an electronic switch which is used to set *on* or *off* state of the ACC. It passes the boolean states to the ACC in form of *TRUE* or *FALSE*.

2. **Vehicle Speed :** It is a sensor that measures an actual speed of the ACC vehicle to pass for the ACC system as a feedback.

3. **Brake Pedal :** It works also like a toggle switch and passes a boolean value in form of *TRUE* or *FALSE* to ACC. Boolean value of the Brake pedal becomes *TRUE* when a brake pedal is pressed.

4. **Throttle Pedal :** Behavior of this input parameter is also similar to the Brake pedal.

5. **Set speed :** It is a switch, which is used to increase or decrease the desired speed of the host vehicle.

6. **Set time gap :** It is also a switch similar to the *Set speed* which is used to increase or decrease the desired *time gap* interval between host vehicle and forward vehicle.

7. **Forward looking sensor :** This is a radar sensor which is used to detect the radar sensor value for calculating the required time gap between host and forward vehicles.

8. **Fault sensor state:** This sensor returns *FALSE* boolean state in the case of failure of the host vehicle or any kind of error detection in the host vehicle.

9. **Indicator State:** Whenever a driver wants to switch from one lane to another lane and to press an *indicator* switch, then this sensor passes boolean states for indicating current indicator status.

10. **Steering Input:** This is a sensor input parameter which is used to capture the movement of steering.

In Fig. 10.3 output parameters which are returned by ACC system are given as follows:

11. **Throttle out:** *Throttle out* returns required value of *throttle* according to the current status of the host vehicle for increasing or decreasing speed of the host vehicle.

12. **Audio Warning:** Whenever any error is detected or manual control is required for overriding to ACC; audio warnings are generated.

13. **Current Speed display:** ACC returns the current of the host vehicle under ACC control. Current speed of the vehicle appears on a display screen for driver assistance.

14. **Display of ACC Active/De-active** Active and De-active states of ACC always display on the screen for driver assistance.

10.5 Development of the ACC System using Refinement Chart

Fig. 10.5 presents the diagrams of the most abstract modal (A) and the resulting concretes models using three successive refinement steps (B to D). The diagrams use a visual notation to represent the operating modes of ACC under functional and parametric requirements. An operating mode is represented by a box with a mode name; an operating mode transition is an arrow connecting two operating modes. The direction of an arrow indicates previous and next operating modes of a transition. Refinement is expressed by nesting boxes to show an integration of new behavior of the system.

A refined diagram of an abstract mode is equivalent to a concrete mode. Fig. 10.5(A) presents an abstract model for the ACC system, which is composed of only *IGNITION CYCLE* mode. The *IGNITION CYCLE* mode is represented by the transitions *Switch ON* and *Switch OFF*. These are the basic transitions for all the ACC functioning modes. The model includes: the state of ignition (on/off), is modelled by a variable *switch*, where *swOn* and *swOff* are represented through an enumerated set (*SwStates*). Initially, throttle angle (*throttle*) is passed from either driver or controller modes. Speed and throttle values are defined abstractly. Independent operation of the plant or car has been ensured for safe operation either it is operating by driver or by the cruise control.

In the next refinement (Fig. 10.5(B) step *IGNITION CYCLE* is refined by *Driver Mode* and *Adaptive Cruise Control (ACC)* mode, corresponding to the activity of the system. When the ignition is turned *on*, control is with a driver. While the ignition is *on*, control can be passed from a driver to the cruise control and back. Two transitions *ACC On* and *ACC Off* are represented, which show driver and cruise control operating modes. The state *on* presents the ACC mode and *off* presents the driver control mode. Some parametric inputs like brake pedal, fault sensor, throttle pedal and much more variables are introduced. Detailed formalisation about all kinds of new introduced parameters are given in the next section. System is naturally represented with two modes: DRIVER and ACC modes. DRIVER is corresponding to an activity when the ACC is *off* or *On* state related to the ACC mode. Possible running modes of the system are depicted in Fig. 10.5(B). In

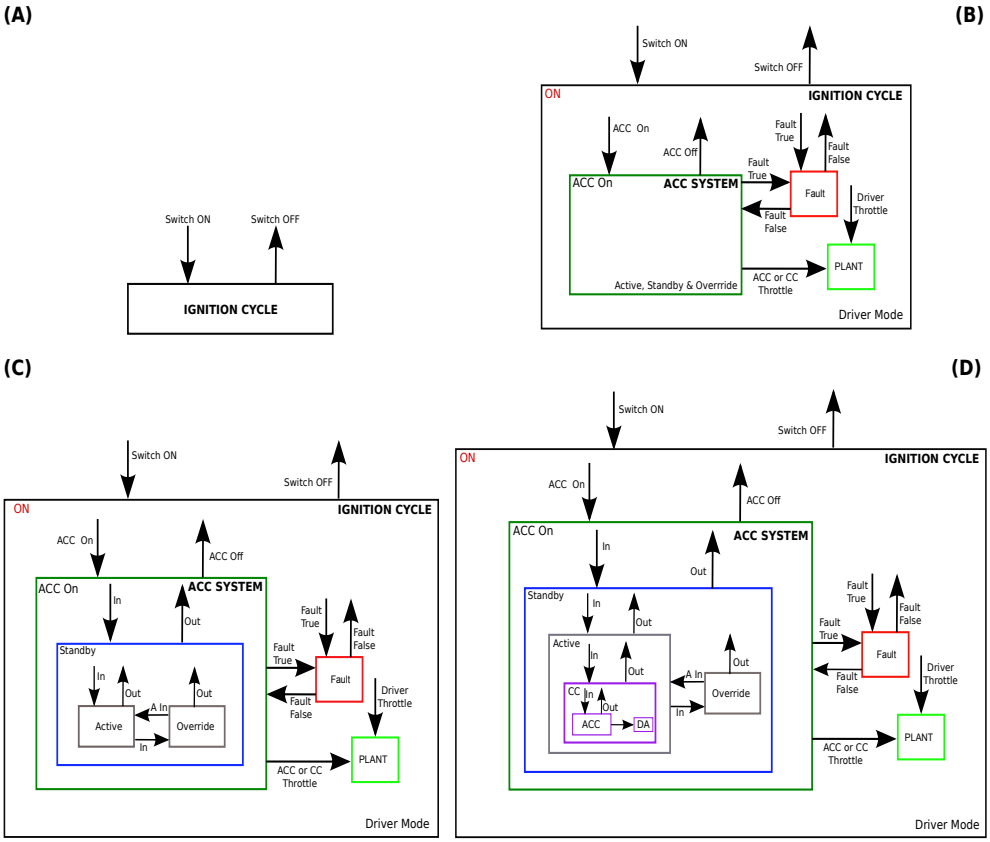


Figure 10.5: Refinements of ACC system using the refinement chart

this refinement *Fault* mode is represented by transitions *Fault True* and *Fault False*. In Fig. 10.5(B), there are two transitions for *Fault True* and two transitions for *Fault False*. One pair of transition (*Fault True, Fault False*) presents mode switching from driver mode to *Fault* mode, while another pair of transition (*Fault True, Fault False*) shows that the mode is switching from ACC operating mode to *Fault* mode. Whenever any fault occurs, system will switch into *Fault* state. *Plant* is a function which has two input transitions *ACC or CC Throttle* and *Driver Throttle* (see Fig. 10.5(B)).

Third refinement step (Fig. 10.5(C)) introduces different states of the ACC operating modes. The *Standby*, *Active* and *Override* are main functioning modes under the *ACC On* state. The ACC operating modes are switching from one state to another state according to the current operating conditions and parametric values. This refinement specially models the state-flow model of the operating modes. In this refinement, *Standby* mode is represented by transitions *In* and *Out*. Other two operating modes *Active* and *Override* are also represented by transitions *In* and *Out*, but *Override* mode has input *In* transition from *Active* mode and in reverse direction input transition *A in* for *Active* mode and output transition for *Override* mode. The *Active* and *Override* operating modes are sequentially connected, which is represented by the sequential refinement chart [Méry 2011e]. A set of parameters are introduced to set the target speed, time gap, desire distance between host

vehicle and forward vehicle, and forward vehicle sensor for modeling the whole system.

In the last refinement step (Fig. 10.5(D)), it introduces more detailed sub-states of the *Active* state. There are three sub-states *CC*, *ACC*, *DA*. This refinement specially models the detailed state-flow model of the *CC* operating mode. In this refinement cruise control (CC), mode is represented by transitions *In* and *Out*. Other two operating modes the adaptive cruise control (ACC) and drive alert (DA) are also represented by transitions *In* and *Out*, but the *DA* mode has only input *In* transition from *ACC* mode. The *ACC* and *DA* operating modes are sequentially connected, which is represented by the sequential refinement chart [Méry 2011e]. Control functions are introduced for the *CC* and *ACC* operating modes, which return a throttle angle. The throttle angle function for *CC* mode is *Throttle_Angle = FCC(setSpeed,speed)*, throttle angle for *ACC* operating mode is *Throttle_Angle = FCC(FACC(setGapDist,actGap),speed)* and throttle angle for Driver mode is *Throttle_Angle = FD(Dri_th)*.

The next section presents only selected parts of our formalization and omit proof details. For instance, we have omitted the specification of refinement of every event from all functioning modes. Only newly introduced event specifications are given in all refinements.

10.6 Formal development of the ACC

10.6.1 Abstraction of ACC system

We start the modeling by defining the Event-B context. The context represents static properties of a system using sets and constants through the definition of axioms and theorems. In the context, we define enumerated sets *States* and *SubStates* using axioms ($axm1$, $axm2$). The set *States* has three main possible states of a system corresponding to the *On*, *Off* and *Fault*. The set *SubStates* represents sub-states of the ACC system when the system is in *On* state. All these three sub-states are *Standby*, *Active* and *Override*. All these enumerated sets are extracted from the ACC requirements document [Houser Amy 2005] and through discussion with domain experts.

$$axm1 : partition(States, \{On\}, \{Off\}, \{Fault\})$$
$$axm2 : partition(SubStates, \{Active\}, \{Standby\}, \{Override\})$$

The ACC modeling approach is different than other modeling techniques because here our main motivation is to generate a *Simulink* [MATLAB] model directly from the proved Event-B models. In this context, we model the basic input/output parameters in an abstract way, which can be easily representable by the SL/SF models [MATLAB].

The ACC based automotive system maintains desired speed of the host vehicle. If a forward vehicle is present, then the host vehicle follows a speed of the forward vehicle. If the forward vehicle is not present, then the host vehicle is used to drive on the desired speed. The basic block diagram of the ACC is given in Fig. 10.6, which presents the ACC system abstractly. This block diagram shows the state-flow, throttle control, driver control and plant system. The throttle angle is an input parameter for the plant function. The throttle

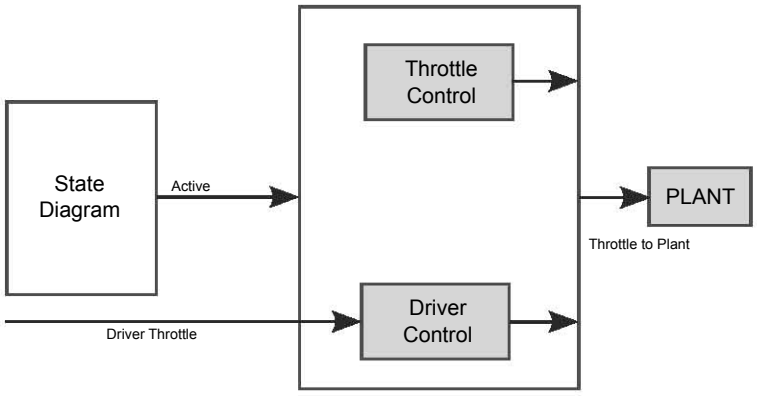


Figure 10.6: Abstract Representation of ACC

angle calculation is an atomic event, while several small steps are required to calculate the throttle angle. An incremental refinement based approach describes all hidden elements of the ACC system.

In our initial model, we formalize an abstract system behavior of the ACC, where three new variables *seqFl1*, *seqFl2* and *seqFl3* are defined to provide sequencing among functional blocks of the ACC system. All these sequence variables are declared as *boolean* type using invariants ($inv1 - inv3$). Four new variables (*speed*, *dri_th*, *throttle*, *state* and *substate*) are introduced to state the dynamic behavior of the system using invariants ($inv4 - inv8$). Invariant ($inv9$) represents a safety property and state that if *seqFl3* is *TRUE*, *state* is *On* and *substate* is *Active* then the throttle angle (*throttle*) should be equivalent to the driver throttle (*dri_th*). The ACC system never switches in any undesired mode. The abstract model is similar to the state-flow model. The last invariant ($inv10$) also expresses a safety property, which states that if *state* is *On* and *substate* is *Active*, then the throttle angle (*throttle*) is lied between 0 to 90.

$inv1 : seqFl1 \in BOOL$
$inv2 : seqFl2 \in BOOL$
$inv3 : seqFl3 \in BOOL$
$inv4 : speed \in 0 \mathinner{..} 60$
$inv5 : dri\_th \in 0 \mathinner{..} 90$
$inv6 : throttle \in 0 \mathinner{..} 90$
$inv7 : state \in States$
$inv8 : substate \in SubStates$
$inv9 : seqFl3 = TRUE \wedge \neg(state = On \wedge substate = Active) \Rightarrow throttle = dri\_th$
$inv10 : state = On \wedge substate = Active \Rightarrow throttle \in 0 \mathinner{..} 90$

EVENT get_inputs
ANY dth
WHERE
grd1 : $seqFl1 = FALSE$
grd2 : $seqFl2 = FALSE$
grd3 : $seqFl3 = FALSE$
grd4 : $dth \in 0 \mathinner{..} 90$
THEN
act1 : $dri\_th := dth$
act2 : $seqFl1 := TRUE$
END

The abstract specification contains five events *get_inputs* to get some input parameters, *update_state* to switch from one state to another state or sub-state, *throttle_by_controller* to get actual throttle value from the controller, *throttle_by_driver* to get actual throttle value from the driver, *plant_action* to give an output parameter in form of the speed.

The event *get_inputs* is the first event, which executes first time and receives all input values for the required variables. All these input values are equivalent to the input parameters for the SL/SF model [MATLAB]. First three guards ($grd1 - grd3$) of this event represent *FALSE* state of all sequencing boolean variables and the last guard represents a type of a local variable (*dth*) in form of numerical range (0..90). When all guards are satisfied then the first action determines a value of *dth*, which is assigned to the driver throttle (*dri_th*) and in the next action, boolean variable *seqFl1* sets *TRUE*.

The event *update_state* is the next event, which executes after the first event and updates flag of sequence boolean variables and sets state and sub-state of the system. The first three guards ($grd1 - grd3$) of this event represent boolean states of the sequence variables. The last two guards determine state and sub-state of the system indeterministically. Actions of this event state that the sequence variable (*seqFl2*) sets *TRUE* and the last two actions determine values of the state (*state*) and sub-state (*substate*) variables.

EVENT update_state
ANY $st, subst$
WHERE
grd1 : $seqFl1 = TRUE$
grd2 : $seqFl2 = FALSE$
grd3 : $seqFl3 = FALSE$
grd4 : $st \in States$
grd4 : $subst \in SubStates$
THEN
act1 : $seqFl2 := TRUE$
act2 : $state := st$
act3 : $substate := subst$
END

EVENT throttle_by_controller
ANY cth
WHERE
grd1 : $seqFl1 = TRUE$
grd2 : $seqFl2 = TRUE$
grd3 : $seqFl3 = FALSE$
grd4 : $cth \in 0 \,..\, 90$
grd5 : $state = On \wedge substate = Active$
THEN
act1 : $seqFl3 := TRUE$
act2 : $throttle := cth$
END

The event *throttle_by_controller* is used to assign current value of the throttle angle (*throttle*) for the plant function. Guards ($grd1 - grd3$) of this event represent three sequence variables in the boolean form, in which first and second sequence variables represent *TRUE* state and the last sequence variable represents *FALSE* state. The next guard ($grd4$) presents the type of control throttle (*cth*), which is lied between a numerical range (0..90). The last guard states that the system state (*state*) is *On* and sub-state of the system (*substate*) is *Active*. Actions of this event state that the sequence variable (*seqFl3*) sets to *TRUE* and determine a new value of the control throttle (*cth*) is assigned to the current throttle variable (*throttle*).

The event *throttle_by_driver* is similar to the last event *throttle_by_controller*. The guards of this event are similar to the last event. In the action's part of the event sets the sequence variable (*seqFl3*) as *TRUE* and driver throttle (*dri_th*) assigns to the current throttle variable (*throttle*) for the plant function.

EVENT throttle_by_driver
WHEN
grd1 : $seqFl1 = TRUE$
grd2 : $seqFl2 = TRUE$
grd3 : $seqFl3 = FALSE$
grd4 : $\neg(state = On \wedge substate = Active)$
THEN
act1 : $seqFl3 := TRUE$
act2 : $throttle := dri\_th$
END

EVENT plant_action
ANY sp
WHERE
grd1 : $seqFl1 = TRUE$
grd2 : $seqFl2 = TRUE$
grd3 : $seqFl3 = TRUE$
grd4 : $sp \in 0 \,..\, 60$
THEN
act1 : $speed := sp$
act2 : $seqFl1 := FALSE$
act3 : $seqFl2 := FALSE$
act4 : $seqFl3 := FALSE$
END

The last event *plant_action* models the functionality of the plant abstractly, where guards express sequence variables in the form of boolean states and local variable sp represents current speed of the host vehicle. Guards present that all sequence variables are *TRUE*. Actions of this event state that the current determined speed (sp) assigns to the actual speed (*speed*) of the system and rest of the actions set *FALSE* states for all the sequence variables.

10.6.2 First Refinement: State-flow Model

In the abstract model, we formalise the whole ACC system abstractly. In the first refinement, we introduce the basic state-flow model of the ACC to formalise functional behavior of the system. Fig. 10.7 represents a block diagram as an introduction of new elements in the abstract model, which represents various states and sub-states of the state-flow model. An enumerated set *SwStates* is defined as *SwStates = partition(SwStates, {swOn}, {swOff})*, which is used to model a switch for the ACC in form of a switch on (*swOn*) and switch off (*swOff*). Another constant *MINSPEED* is defined as *MINSPEED* $\in$ *7 .. 8* to set the minimum speed of the host vehicle.

Few more important associated components like brake sensor, resume button, brake pedal, throttle pedal and fault sensor are introduced in this level for modeling the ACC system. For defining all these associated components, we introduce some variables like *switch*, *faultSensor*, *brPedal*, *thPedal*, *setCC* and *resume*. All these variables are declared using invariants ($inv1 - inv6$). The variable *switch* defines *On* and *Off* switching functionality of the system. Means, system will be in a functional state when a variable *switch* is *On*, or *Off* state presents non-functioning state of the system. All other variables are introduced as the boolean type, which declare the boolean state of the associated system components. For example *TRUE* state of the *faultSensor* presents that system is in *Fault* state and *FALSE* state represents that the system has not any kind of fault. A set of invariants ($inv7 - inv18$) are introduced to define the safety properties, which verify the required system properties and states that the system will never show any undesired behavior. All these safety properties verify the functional behaviors of the state-flow model and switching behaviors from one state to another state.

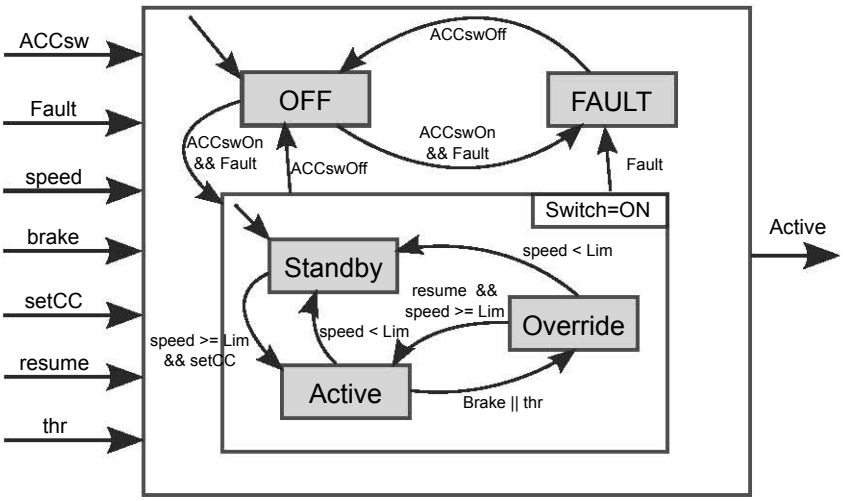


Figure 10.7: First Refinement of ACC

$inv1 : switch \in SwStates$
$inv2 : faultSensor \in BOOL$
$inv3 : brPedal \in BOOL$
$inv4 : thPedal \in BOOL$
$inv5 : setCC \in BOOL$
$inv6 : resume \in BOOL$
$inv7 : faultSensor = TRUE \wedge seqFl2 = TRUE \Rightarrow state \in \{Off, Fault\}$
$inv8 : switch = swOn \wedge faultSensor = FALSE \wedge seqFl2 = TRUE \Rightarrow$
$state \in \{On, Fault\}$
$inv9 : state = On \wedge speed < MINSPEED \wedge seqFl2 = TRUE \Rightarrow$
$substate \in \{Standby, Override\}$
$inv10 : switch = swOff \wedge seqFl2 = TRUE \Rightarrow state = Off$
$inv11 : switch = swOn \wedge faultSensor = TRUE \wedge seqFl2 = TRUE \Rightarrow$
$state = Fault$
$inv12 : switch = swOn \wedge speed < MINSPEED \wedge seqFl2 = TRUE \Rightarrow$
$substate \neq Active$
$inv13 : state = Off \vee state = Fault \Rightarrow substate \neq Active$
$inv14 : state = Off \vee state = Fault \Rightarrow substate \neq Override$
$inv15 : switch = swOff \Rightarrow substate \neq Active \vee substate \neq Override$
$inv16 : faultSensor = TRUE \Rightarrow substate \neq Active \vee substate \neq Override$
$inv17 : seqFl2 = TRUE \wedge substate = Active \wedge speed \geq MINSPEED \Rightarrow$
$faultSensor = FALSE$
$inv18 : seqFl2 = TRUE \wedge substate = Active \wedge speed \geq MINSPEED \Rightarrow$
$state = On$

This refinement level contains a group of new events, which models the state-flow model of the system. The event get_inputs is defined in the abstract machine, which is extended by a new input state variable. All new events are a refinement of the event $update\_state$, which specify the state-flow of the ACC system.

This refinement level contains fourteen new events, *switchOnFault* is to specify *On*

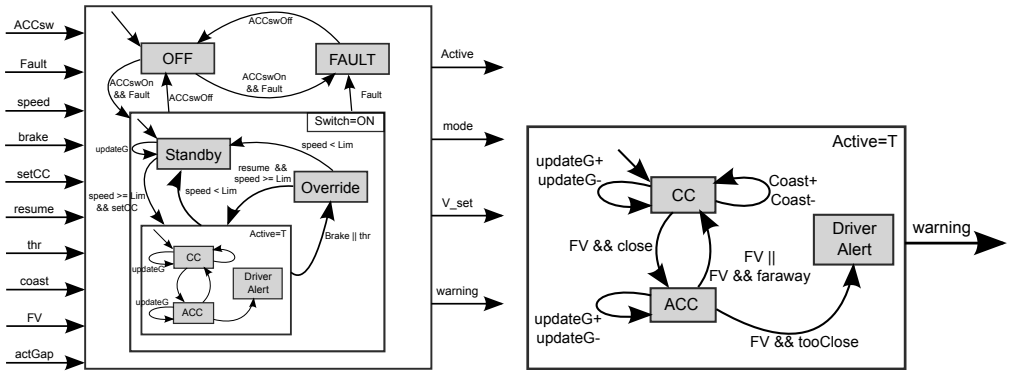
state of a switch during the *Fault* state, *FaultWhenOn* is to model a fault occurring state when switch is *On*, *switchOnNoFault* is to express *On* state of a switch when system is not in *Fault* state, *switchOff* is to model *Off* state of a switch, *remainATOff* is to express continue in *Off* state, *remainATfault* is to model continue in *fault* state, *remainATstandby* is to specify continue in *Standby* state, *remainATOverride* is to express continue in *Override* state, *standbyToActive* is to formalise switch from *Standby* to *Active*, *ActiveToStandby* is to model switch from *Active* to *Standby*, *remainATActive* is to specify continue in *Active* state, *ActiveToOverride* is to formalise switch from *Active* to *Override*, *OverrideToActive* is to formalise switch from *Override* to *Active* and *OverrideToStandby* is to specify switch from *Override* to *Standby*. All these events have similar kinds of refinements to formalize the possible states and sub-states of the ACC system. For example, in the following blocks, we have given formalisation of only two events *switchOnFault* and *standbyToActive* to understand the basic methodology of refinement and modeling of different states and sub-states of the system.

EVENT switchOnFault Refines update_state
WHEN
grd1 : $seqFl1 = TRUE$
grd2 : $seqFl2 = FALSE$
grd3 : $seqFl3 = FALSE$
grd4 : $state = Off$
grd5 : $switch = swOn$
grd6 : $faultSensor = TRUE$
WITNESS
st $st = Fault$
subst $subst = Standby$
THEN
act1 : $seqFl2 := TRUE$
act2 : $state := Fault$
act3 : $substate := Standby$
END

EVENT standbyToActive Refines update_state
WHEN
grd1 : $seqFl1 = TRUE \ \wedge \ seqFl2 = FALSE$
grd2 : $seqFl3 = FALSE$
grd3 : $state = On \ \wedge \ switch \neq swOff$
grd4 : $substate = Standby$
grd5 : $faultSensor = FALSE$
grd6 : $speed \geq MINSPEED \ \wedge \ setCC = TRUE$
WITNESS
st $st = On$
subst $subst = Active$
THEN
act1 : $seqFl2 := TRUE$
act2 : $substate := Active$
END

10.6.3 Second Refinement: Detailed modeling of State-flow

This refinement introduces granularity of the state-flow model through introduction of many new variables and events. In this level, we have formalised the concrete level of state-flow model using states and sub-states of the ACC system. A detailed description of the *Active* state is given in Fig. 10.8, which presents parametric details as well as required conditions for switching among sub-states. To model the various states of the ACC system like cruise control (CC), adaptive cruise control (ACC) and driver alert (DA), we have defined an enumerated constant *MODES* as *partition(MODES, {CC}, {ACC}, {DA})*. This enumerated constant is used to define possible operational modes (CC, ACC, DA).



(a) Detailed Modeling of State-flow Model (b) State $Active$ in more detail

Figure 10.8: Modeling of State-flow

To model the concrete level of the state-flow model with required conditions, we have declared some new variables using invariants ($inv1 - inv11$). A new variable *activeMode* is defined to keep current working mode of the system like CC, ACC or DA. The next variable *fvSensor* is declared as boolean to specify a boolean state of the forward vehicle. If a forward vehicle presents front of the host vehicle, then the value of *fvSensor* is *TRUE* else *FALSE*. Other variables *setSpeed* and *setGapTime* are used to set the value of speed and time gap for the host vehicle, which can be modified by a driver under the ACC mode.

A variable *DA_warning* is declared as boolean type to express warning for a driver. The *TRUE* state of *DA_warning* shows that warning alert is *ON* for a driver and *FALSE* state presents no warning alert for a driver. Another variable *actGap* is used to calculate the actual distance between host and forward vehicles and the next variable *setGapDist* is used to set a desired distance between host and forward vehicles. The next two variables *gapPlus* and *gapMinus* are defined as *boolean* type to specify the increase and decrease states of the gap value in the state-flow. Similarly last two variables *coastPlus* and *coastMinus* are used to model increase and decrease states of the *coast* value in form of boolean state. A set of invariants ($inv12 - inv17$) are introduced to define the safety properties and to verify that the system will not show any undesired behavior during switching among sub-states.

$inv1 : activeMode \in MODES$
$inv2 : fvSensor \in BOOL$
$inv3 : setSpeed \in MINSPEED \mathinner{..} 60$
$inv4 : setGapTime \in \{1, 2, 3\}$
$inv5 : DA\_warning \in BOOL$
$inv6 : actGap \in \mathbb{N}$
$inv7 : setGapDist \in 0 \mathinner{..} 180$
$inv8 : gapPlus \in BOOL$
$inv9 : gapMinus \in BOOL$
$inv10 : coastPlus \in BOOL$
$inv11 : coastMinus \in BOOL$
$inv12 : state = On \wedge substate = Active \wedge activeMode = DA \Rightarrow$
$DA\_warning = TRUE$
$inv13 : state = On \wedge substate = Active \wedge activeMode = ACC \Rightarrow$
$fvSensor = TRUE$
$inv14 : state = On \wedge substate = Active \wedge activeMode \neq DA \Rightarrow$
$DA\_warning = FALSE$
$inv15 : state = On \wedge substate = Override \Rightarrow DA\_warning = FALSE$
$inv16 : state = On \wedge substate = Standby \Rightarrow DA\_warning = FALSE$
$inv17 : state = Off \Rightarrow DA\_warning = FALSE$

Similarly, as the first refinement, we have extended the event *get_inputs* also in this refinement level and add more guards and actions to model new input parameters. A new set of events are the refinement of the events *remainATActive* and *remainATstandby*. A group of refinement events of *remainATActive* models all possible sub-states and switching behavior of the system, when a system is in *Active* state and other group of events of *remainATstandby* are used to model the sub-states and switching behavior of the system, when a system is working under the *Standby* state. Thus, in a summarized way we can say that, this refinement level models sub-states and switching behaviors of the system, when specially the system is working under the *Standby* or *Active* mode.

Fifteen new events are introduced as refinement of the event *remainATActive* to model the *Active* state of the system along with all possible system behaviors. A new event *CCtoACC* is used to model the switching behavior from *CC* to *ACC* operating modes, *ACCtoCC* is used to model the system switching behavior from *ACC* to *CC* operating modes, *ACCtoDA* is used to model the system switching behavior from *ACC* to *DA* operating modes, *DAtoCC* is used to model the system switching behavior from *DA* to *CC* operating modes, *remainATCC* is used to formalise the system behaviors under the *CC* mode, *remainATDA* is used to formalise system behavior under the *DA* mode, *remainATACC* is used to formalise system behavior under the *ACC* mode, *updateGap_active_plus* is used to increase the gap value in *Active* state, *updateGap_active_plus_sat* is used to set the updated value, *updateGap_active_minus* is used to decrease the gap value in the *Active* state, *updateGap_active_minus_sat* is used to set the updated value, *coastIncrement* is used to increase the *Coast* value, *coastIncrement_sat* is used to set incremented coast value, *coastDecrement* is used to decrease the coast value and finally *coastDecrement_sat* is used to set the decremented coast value. We give an example of a refinement event *CCtoACC* to understand the other refinement of events of the system.

Another group of events is a refinement of the *remainATstandby*. There are four events, where *updateGap_standby_plus* is used to increase the value of a gap under *Standby* mode, *updateGap_standby_minus* is used to decrease the value of a gap under *Standby* mode, *updateGap_standby_plus_sat* is used to set the updated value of the incremented gap under *Standby* mode, and *updateGap_standby_minus_sat* is used to set updated value of the decremented gap under *Standby* mode. We have shown this refinement through an example using the event *updateGap_standby_plus*.

EVENT CCtoACC Refines remainAtActive
WHEN
grd1 $seqFl1 = TRUE$
grd2 $seqFl2 = FALSE$
grd3 $seqFl3 = FALSE$
grd4 $state = On$
grd5 $substate = Active$
grd6 $faultSensor = FALSE$
grd7 $speed \geq MINSPEED$
grd8 $switch \neq swOff$
grd9 $brPedal = FALSE \wedge thPedal = FALSE$
grd10 $fvSensor = TRUE$
grd11 $activeMode = CC$
grd12 $actGap \leq 500$
THEN
act1 $seqFl2 := TRUE$
act2 $activeMode := ACC$
act3 $setGapDist := setGapTime * speed$
END

EVENT updateGap_satndby_plus Refines remainATstandby
WHEN
grd1 : $seqFl1 = TRUE$
grd2 : $seqFl2 = FALSE$
grd3 : $seqFl3 = FALSE$
grd4 : $state = On$
grd5 : $switch \neq swOff$
grd6 : $substate = Standby$
grd7 : $faultSensor = FALSE$
grd8 : $\neg(setCC = TRUE \wedge speed \geq MINSPEED)$
grd9 : $gapPlus = TRUE$
grd10 : $setGapTime < 3$
THEN
act1 : $seqFl2 := TRUE$
act2 : $setGapTime := setGapTime + 1$
END

Two new events *throttle_by_ACC_control* and *throttle_by_CC_control* are introduced as the refinement of the *throttle_by_control*. This refinement is refining control function of

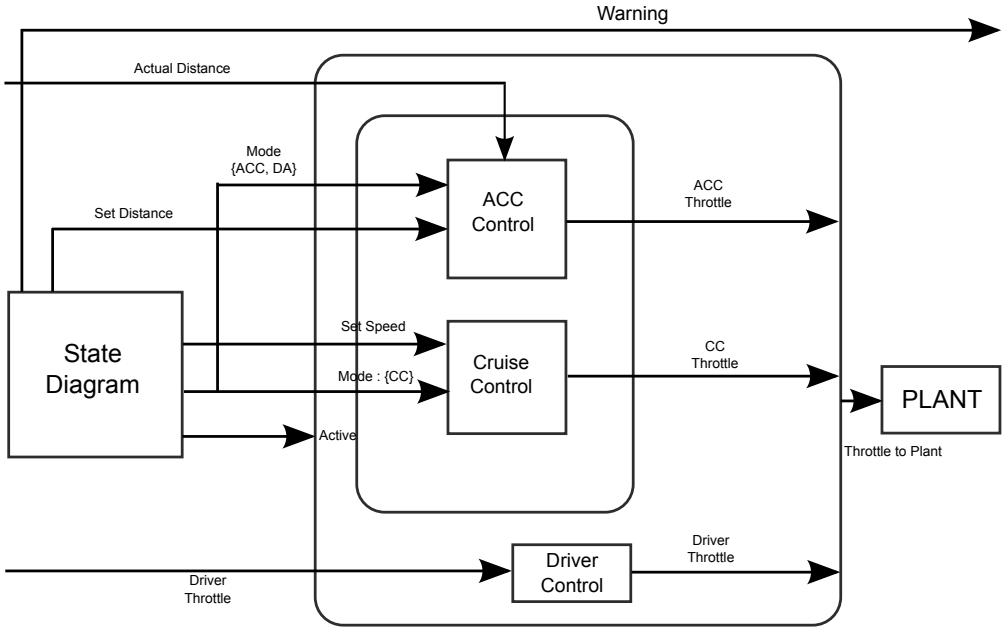


Figure 10.9: Basic Architecture of ACC system

the ACC system, which is represented in Fig. 10.9. This refinement is used to decompose the single event *throttle_by_control* using refinement approach and to model two different components (CC, ACC) for throttle output. These two throttle outputs are *throttle angle* for the ACC and CC controls.

EVENT throttle_by_ACC_control Refines throttle_by_controller
ANY cth
WHERE
grd1 : $seqFl1 = TRUE$
grd2 : $seqFl2 = TRUE$
grd3 : $seqFl3 = FALSE$
grd4 : $cth \in 0 \,..\, 90$
grd5 : $state = On \wedge substate = Active$
grd6 : $activeMode \in \{ACC, DA\}$
THEN
act1 : $seqFl3 := TRUE$
act2 : $throttle := cth$
END

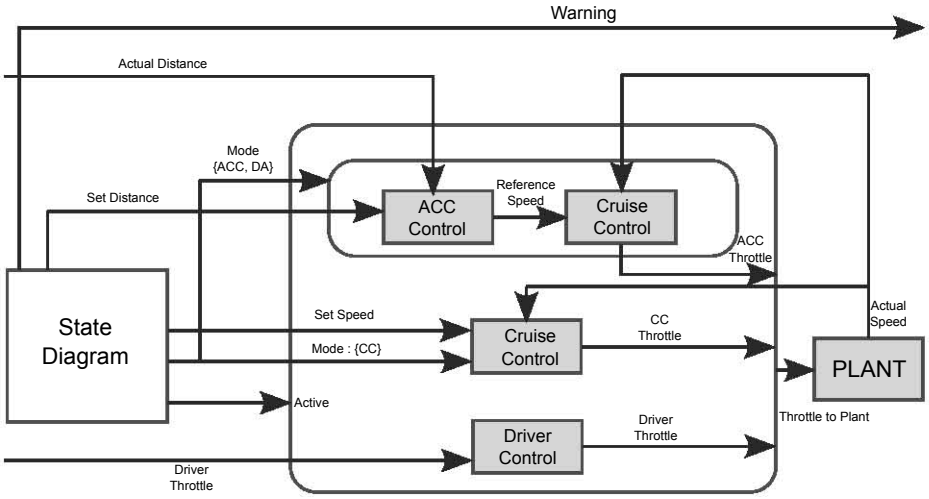


Figure 10.10: Detailed Architecture of CC and ACC Components

EVENT throttle_by_CC_control Refines throttle_by_controller
ANY cth
WHERE
grd1 : $seqFl1 = TRUE$
grd2 : $seqFl2 = TRUE$
grd3 : $seqFl3 = FALSE$
grd4 : $cth \in 0 \,..\, 90$
grd5 : $state = On \wedge substate = Active$
grd6 : $activeMode = CC$
THEN
act1 : $seqFl3 := TRUE$
act2 : $throttle := cth$
END

10.6.4 Third Refinement: Controller Definition

Final refinement of the ACC system is to specify the *uninterpreted functions* and to model the different components. Fig. 10.9 represents a more abstract model of the different components of the ACC, while this refinement formalises a detailed architecture of the ACC. Fig. 10.10 presents complete block architecture of the ACC, where it presents the cruise control throttle (CC), adaptive cruise control throttle (ACC) and driver throttle. All these throttle functions are defined as the *uninterpreted functions* including plant function. Input and output parameters of the *uninterpreted functions* are similar to the real controller functions.

Two new axioms ($axm1 - axm2$) are defined to represent as a set of controller functions (FACC, FCC) to calculate the *throttle angle*. These functions are defined as *uninterpreted functions* where inputs and outputs are known. In the final concrete model, these functions are replaced by the real functions which are provided by a control engineer. The input and output parameters of the *uninterpreted function* and real functions are same. The

variable throttle angle (*throttle*) represents the current throttle angle, which is updated by the *uninterpreted functions* FACC and FCC.

$$axm1 : FACC \in 0 \mathinner{..} 180 \times 0 \mathinner{..} 500 \rightarrow 0 \mathinner{..} 60$$
$$axm2 : FCC \in 0 \mathinner{..} 60 \times 0 \mathinner{..} 60 \rightarrow 0 \mathinner{..} 90$$

A new variable *refSpeedACC* is declared to obtain a reference value through the actual gap (*actGap*) and desired gap distance (*setGapDist*) using the ACC control function (FACC). Another sequence variable (*seqFl4*) is defined as a *boolean* type to define a proper sequencing for the ACC control and CC control functions according to the state-flow model. All these variables are declared using invariants ($inv1 - inv2$). Other invariants ($inv3 - inv6$) are introduced as the safety properties and to verify the desired behaviors of the complete system using the state-flow and controller functions according to the domain experts. Invariant ($inv3$) states that if *seqFl2* is *TRUE*, *state* is *On*, *substate* is *Active* and system mode (*activeMode*) is either in the ACC control (*ACC*) mode or in the driver alert (*DA*) mode, then the actual gap between the forward vehicle and the host vehicle should be less than or equal to 500. The next three invariants ($inv4 - inv6$) are most important invariants of the ACC system. Invariant ($inv4$) states that if the *state* is *on*, *substate* is *Active*, system operating mode (*activeMode*) is in the cruise control (*CC*) mode and the sequence flag *seqFl3* is *TRUE*, then the throttle angle (*throttle*) should be equivalent to *FCC(setSpeed* $\mapsto$ *speed)*. The *FCC* is a function which represents controller function blocks in Fig. 10.10. The next invariant ($inv5$) represents that if *state* is *On*, *substate* is *Active*, system operating mode (*activeMode*) is either in the ACC control (*ACC*) mode or in the driver alert (*DA*) mode and the sequence flags (*seqFl3*, *seqFl2*) are *TRUE*, then the throttle angle (*throttle*) should be equivalent to *FCC(FACC(setGapDist* $\mapsto$ *actGap)* $\mapsto$ *speed)*. The *FCC* and *FACC* are two functions, which are representing the two controller functions (*FACC*, *FCC*) (see Fig. 10.10). The Desired distance (*setGapDist*) and actual distance (*actGap*) are the input parameters of the function *FACC*. The output of function *FACC* is the desired speed *(FACC(setGapDist* $\mapsto$ *actGap)*) of the host vehicle and the actual speed of the host vehicle (*speed*) is the input parameter of the function *FCC*. The last invariant ($inv6$) represents that if the *state* is *On*, *substate* is *Active*, system operating mode (*activeMode*) is either in the ACC control (*ACC*) mode or in the driver alert (*DA*) mode and the sequence flags (*seqFl2*, *seqFl4*) are *TRUE*, then the referenced speed (*refSpeedACC*) should be equivalent to *(FACC(setGapDist* $\mapsto$ *actGap)*. The *FACC* is a function which represents a controller block (see Fig. 10.10). The desired distance (*setGapDist*) and actual distance (*actGap*) are the input parameters of the function *FACC*. All these invariants are related to the throttle angle calculation, which maintains speed of the host vehicle. These invariants verify all required properties of the controller functions, and their functional composition *FCC(FACC(...)...)* output is in form of the throttle angle (*throttle*) calculation.

$inv1 : refSpeedACC \in 0 \mathinner{..} 60$
$inv2 : seqFl4 \in BOOL$
$inv3 : seqFl2 = TRUE \wedge state = On \wedge substate = Active \wedge$
$activeMode \in \{ACC, DA\} \Rightarrow actGap \leq 500$
$inv4 : state = On \wedge substate = Active \wedge activeMode = CC \wedge$
$seqFl3 = TRUE \Rightarrow throttle = FCC(setSpeed \mapsto speed)$
$inv5 : state = On \wedge substate = Active \wedge activeMode \in \{ACC, DA\} \wedge$
$seqFl3 = TRUE \wedge seqFl2 = TRUE \Rightarrow$
$throttle = FCC(FACC(setGapDist \mapsto actGap) \mapsto speed)$
$inv6 : state = On \wedge substate = Active \wedge activeMode \in \{ACC, DA\} \wedge$
$seqFl2 = TRUE \wedge seqFl4 = TRUE \Rightarrow$
$refSpeedACC = FACC(setGapDist \mapsto actGap)$

In this refinement, we have introduced only one new event *ACC_control_1*, which is used to calculate the reference speed (*refSpeedACC*) using a control function FACC. The guards of this event state that when sequence flags (*seqFl1,seqFl2*) are TRUE, sequence flag (*seqFl3*) is *FALSE*, *state* is *On*, *substate* is *Active*, and system operating mode (*activeMode*) is either ACC or CC, then the actions state that sequence flag (*seqFl4*) becomes *TRUE* and the reference speed is calculated from the functional expression *FACC(setGapDist* $\mapsto$ *actGap)*.

EVENT ACC_control_1
WHEN
grd1 $seqFl1 = TRUE$
grd2 $seqFl2 = TRUE$
grd3 $seqFl3 = FALSE$
grd4 $state = On \wedge substate = Active$
grd5 $activeMode \in \{ACC, DA\}$
THEN
act1 : $seqFl4 := TRUE$
act2 : $refSpeedACC := FACC(setGapDist \mapsto actGap)$
END

Last two events *throttle_by_ACC_control* and *throttle_by_CC_control* are simple refinements, in which all guards and actions are similar to the previous refinement level. In the event *throttle_by_ACC_control* a new witness is defined to replace the local variable *cth* with a new functional expression *FCC(refSpeedACC* $\mapsto$ *speed)*, and a new action ($act2$) is added to set *FALSE* boolean value of the sequence flag ($seqFl4$) and finally action ($act3$) predicate is modified using new predicate expression *throttle* := *FCC(refSpeedACC* $\mapsto$ *speed)*. Similarly, in the event *throttle_by_CC_control* a new witness is introduced to replace the local variable *cth* with a new functional expression *FCC(setSpeed* $\mapsto$ *speed)* and finally, the action ($act2$) predicate is modified by a new predicate expression *throttle* := *FCC(setSpeed* $\mapsto$ *speed)*.

EVENT throttle_by_ACC_control Refines throttle_by_ACC_control
WHEN
grd1 $seqFl1 = TRUE$
grd2 $seqFl2 = TRUE$
grd3 $seqFl3 = FALSE$
grd4 $state = On \wedge substate = Active$
grd5 $activeMode \in \{ACC, DA\}$
grd6 $seqFl4 = TRUE$
WITNESS
st $cth = FCC(refSpeedACC \mapsto speed)$
THEN
act1 : $seqFl3 := TRUE$
act2 : $seqFl4 := FALSE$
act3 : $throttle := FCC(refSpeedACC \mapsto speed)$
END

EVENT throttle_by_CC_control Refines throttle_by_CC_control
WHEN
grd1 $seqFl1 = TRUE$
grd2 $seqFl2 = TRUE$
grd3 $seqFl3 = FALSE$
grd4 $state = On \wedge substate = Active$
grd5 $activeMode = CC$
WITNESS
st $cth = FCC(setSpeed \mapsto speed)$
THEN
act1 : $seqFl3 := TRUE$
act2 : $throttle := FCC(setSpeed \mapsto speed)$
END

10.7 Model Validation and Analysis

Table 10.1 is expressing proof statistics for the formal development of ACC using the Rodin platform. These statistics measure the size of the model, the proof obligations generated and discharged by the Rodin prover, and those are interactively proved. The complete development of the ACC system results in 529(100%) proof obligations, in which 525(99%) are proved automatically by the Rodin tool. The remaining 4(1%) proof obligations are proved interactively using Rodin tool. In Event-B models, many proof obligations are generated due to introduction of new functional behaviors of the ACC. In order to guarantee the correctness of these functional behaviors, we have established various invariants in the stepwise refinement. Interactive proof obligations are proved through simplification of the complex expression. The stepwise refinement of the ACC system helps to achieve a high degree of automatic proofs.

We have also used ProB [ProB] model checker to verify the system behavior, which helps for gaining confidence that the developed formal model complies with requirements

Model	Total number of POs	Automatic Proof	Interactive Proof
Abstract Model	17	17(100%)	0(0%)
First Refinement	224	224(100%)	0(0%)
Second Refinement	104	104(100%)	0(0%)
Third Refinement	184	180(98%)	4(2%)
Total	529	525(99%)	4(1%)

Table 10.1: Proof Statistics

document [Houser Amy 2005; Hrovat 1991]. We have used the ProB tool [ProB] that supports *automated consistency checking* of Event-B machines via model checking [Clarke 1999] and constraint-based checking [Jackson 2002]. ProB model checker helps to discover potential problems and helps for strengthening the invariants of Event-B model. We have validated state-flow model with all possible operating modes of the ACC system in each refinement of models. ProB was very useful in the development of ACC specification, and could animate all of our models and able to prove the absence of any error (no counter example exist).

10.8 Formal Specification into Simulink

Finally, we have transformed concrete level formal specification of the ACC system into equivalent *Simulink* [MATLAB] model. Here, we have done all transformation manually. Transformation process is based on semantics of the developed formal model. The developed formal model in Event-B modeling language and transformed *Simulink* model are comply with requirements document. Some parts of the formal model are represented by *State-flow* and remaining parts of the model is presented by the *Simulink* [MATLAB]. To distinguish between the *State-flow* and *Simulink* parts from the Event-B formal model is one of the major problems. Most of the events are represented by *Simulink* block diagrams or subsystems. Actions of an event are represented in the form of expressions. Guards are also represented in the form of expressions. The guards are also helped to connect different blocks of the *Simulink* blocks. Final *Simulink* model contains both *State-flow* as well as *Simulink* models. Fig. 10.11 represents only *State-flow* model of the ACC. Fig. 10.12 represents combined block diagrams of *State-flow* and *Simulink* models. We have also compiled our *Simulink* model and test the desired behavior of the ACC system.

Fig. 10.13 represents a relationship between Event-B formal model and the SL/SF model. We have assumed that covered regions by both eclipses are transformable from Event-B to *Simulink*. Some parts cannot be possible to transform directly from Event-B to Simulink model. Non-overlapping part of the Event-B in the figure presents formal specification part, which is not transformable or applicable to generate the Simulink models. The formal specification of this part is required to formalise the whole system. Similarly non-overlapping part of the Simulink in the figure presents some extra required components according to the Simulink modeling technique, which are not defined in the Event-B

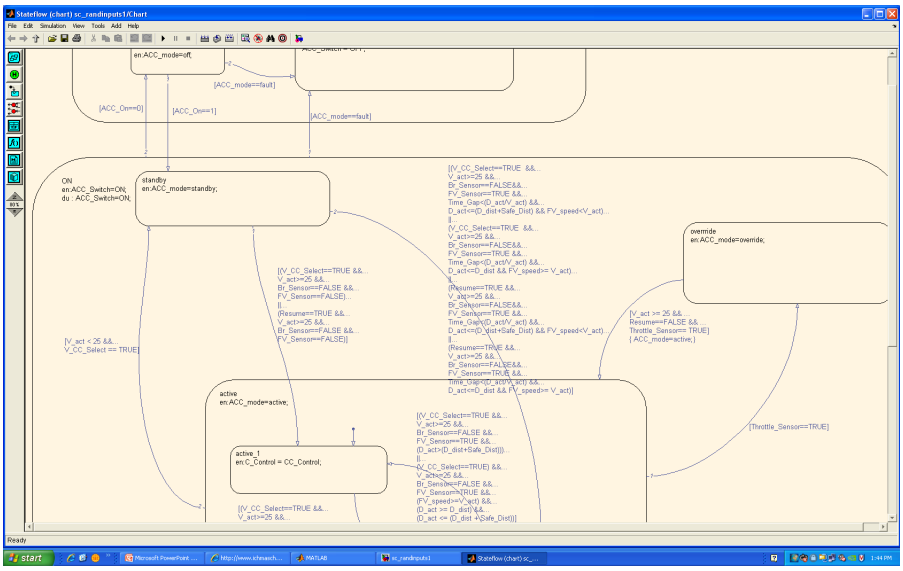


Figure 10.11: State-flow model generation from Event-B formal model

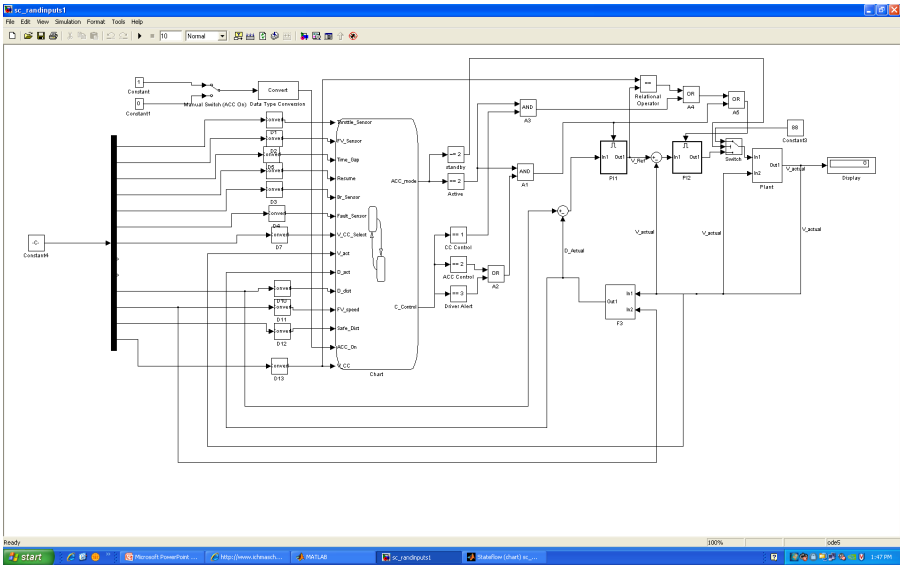


Figure 10.12: Simulink model generation from Event-B formal model

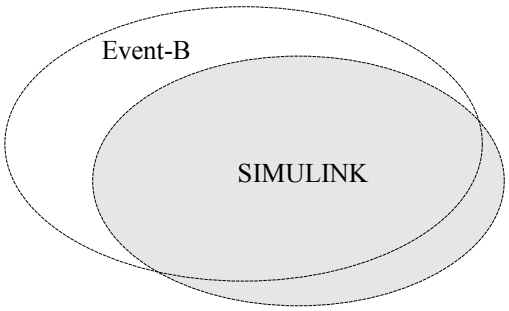


Figure 10.13: Relationship between Event-B and Simulink

models. During the Simulink model generation from the Event-B specification, there is required some depth analysis for removing some parts from the Event-B formal specification and to add some new components to produce the final Simulink model. For instance, in this ACC case study, we have modeled some events to input data for processing, while in the Simulink model by default each functional block has some inputs and outputs. Hence, the events which model the system for input and output arguments are not directly applicable to the Simulink transformation. Another example is that in the Simulink model each block has some functions with inputs and outputs, while single block of the Simulink model is represented by several events into Event-B formal specification. We have gained some relationship through our experience of transformation of Event-B formal model to Simulink model, which are as follows:

- Sensor input events in Event-B are equivalent to constant or inputs in the SL/SF model.
- Event-B model has a lack of sequencing.
- State-flow and Simulink blocks, which are difficult to distinguish in the Event-B models.
- Event-B model is loosely coupled while Simulink model is tightly coupled.

10.9 Code Generation for ACC System using EB2ALL

In this chapter, we have presented a proof-based an incremental formal development of the ACC system using our proposed techniques and tools (see Part-I). This section presents an automatic code generation from the developed and proved formal specification of the ACC system. We now illustrate the use of EB2C, EB2C++, EB2J and EB2C# tools [EB2ALL 2011; Méry 2011b] by means of the automatic generation of C, C++, Java and C# codes for the ACC system. This tool has a technique of automatic support of the safety assurance of a generated code. To achieve a verified source code of the ACC system, we have done further refinement of the concrete model of the ACC system using a new context, which has some data ranges (see Table 7.1) corresponding to the programming languages. This context file

provides a deterministic range for all kinds of data types. This refinement makes the model deterministic and generates some proof obligations due to defining the fixed data ranges of all the constants and variables of the ACC model. The generated proof obligations are discharged by automatic as well as manual, and all these proofs are necessary to verify the specification in order to guarantee the consistency and correctness of the system. We have discharged all generated proof obligations before generating the source code. This level of refinement complies system specification abstractly. Now, we move to the next level of code translation methodology as to pass the concrete model for continuing translation process.

The code translation from Event-B formal specification into any programming language using EB2ALL is straightforward using a set of plugins (EB2C, EB2C++, EB2J and EB2C#) [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b]. The main idea is to translate an Event-B model into any programming language code using the last concrete model. The EB2ALL tool generates programming language files corresponding to the concrete models. A generated source file using EB2ALL tool has a basic structure: a set of constants, variables and functions. A set of constants and variables are extracted from the context and machine sections of the Event-B model of the ACC system, respectively. Data type of a constant is defined as an axiom in Event-B model. Similarly, data type of a variable is extracted from the invariant section of the model. Initial value of the constants and variables are initialized, if their initial values are declared. A set of constants and variables are given as follows, which are excerpted from the translated 'C' codes of the ACC system model.

```
enum States {On,Off,Fault}; /* Enumerated definition */
enum SubStates {Standby,Active,Override}; /* Enumerated
definition */
enum SwStates {swOn,swOff}; /* Enumerated definition */
enum MODES {CC,ACC,DA}; /* Enumerated definition */
const int MINSPEED=8; /* Integer in range 7-8 */

/* Structure of function in C*/
 int FACC(int arg1,int arg2)
 {

//TODO: Add your Code
 return;
 }

/* Structure of function in C*/
 int FCC(int arg1,int arg2)
 {

//TODO: Add your Code
 return;
 }

enum MODES activeMode; /* Enumerated type variable */
```

```
enum SwStates Switch; /* Enumerated  type variable */
unsigned long int actGap; /* Integer in range undefined */
BOOL DA_warning;
BOOL brPedal;
BOOL coastMinus;
BOOL coastPlus;
int dri_th; /* Integer in range 0-90 */
BOOL faultSensor;
BOOL fvSensor;
BOOL gapMinus;
BOOL gapPlus;
int refSpeedACC; /* Integer in range 0-60 */
BOOL resume;
BOOL seqFl1;
BOOL seqFl2;
BOOL seqFl3;
BOOL seqFl4;
BOOL setCC;
int setGapDist; /* Integer in range 0-180 */
unsigned long int setGapTime; /* C data type declaration when
variable is given in type of predicate */
int setSpeed; /* Integer in range */
int speed; /* Integer in range 0-60 */
enum States state; /* Enumerated  type variable */
enum SubStates substate; /* Enumerated  type variable */
BOOL thPedal;
int throttle; /* Integer in range 0-90 */
```

A set of functions are extracted equivalent to a set of events of the ACC formal model. All the events of Event-B are translated into equivalent programming language functions. An event INITIALIZATION is a function, which initialize default values of all the variables. An event of Event-B model has fixed organization of the internal components: local variables, guards (pre-conditions) and actions. An event may contain some local variables. The global constants and variables are declared on the top of the programming language source file, while local variables are declared within the function body. All events of a formal model are translated as a set of programming language functions. This function has the similar structure as an event. During the translation of the events, the guards are translated into equivalent to 'if' statement using logical conjunction, disjunction, implication and equivalence. Each guard represents into a separate 'if' statement like nested 'if' structure. All these guards represent a set of preconditions, which are required to satisfy for executing the action predicates. All action predicates of a formal model event are directly translatable equivalent into programming language assignment expressions. The EB2ALL tool is capable of analyse the syntax of Event-B guards and actions predicate. In the ACC formal model, their predicates are simple, which are obtained through several refinements. All pre-conditions or guards are required to be TRUE for firing all the set of actions. However, despite being a complex system, the ACC system pre-conditions are fairly simple to calculate. If all the guards are true, then the action's predicates execute and return TRUE

for successful execution of the function. If any 'if' condition is false, then the function returns FALSE and action's part of the function does not execute.

```
...
BOOL throttle_by_ACC_control(void)
{
  /* Guards No. 1*/
  if(seqF11 == TRUE){
  /* Guards No. 2*/
  if(seqF12 == TRUE){
  /* Guards No. 3*/
  if(seqF13 == FALSE){
  /* Guards No. 4*/
  if( ( state == On ) && ( substate == Active ) ){
  /* Guards No. 5*/
  if( ( activeMode == ACC ) || ( activeMode == DA ) ){
  /* Guards No. 6*/
  if(seqF14 == TRUE){
  /* Actions */
  seqF13 = TRUE;
  seqF14 = FALSE;
  throttle= FCC(refSpeedACC,speed);
  return TRUE;
}}}}}}
  return FALSE;
}
...
```

To make the generated code executable, the EB2ALL tool generates an *Iterate* function that contains a list of all functions as in form of a function call. Another function is a main body of the program like *main()* in 'C', which calls *Iterate* function. These two extra functions are used to compile and execute the generated code.

The source code is automatically generated in any programming language (C, C++, Java and C#) from the verified specification in less than three seconds. The generated code resulted in over 2300 lines. Here, we have presented a brief overview of the translation from the Event-B specification of the ACC formal model into 'C' using EB2C tool. Based on this translation, we were able to automatically generate 'C' code and execute a simulation of the ACC system.

10.10 Discussion and Conclusion

10.10.1 Discussion

This chapter has focused on the formalization of hybrid control system using *"separation of concern"* based refinement approach. In the system development process, we have decomposed the whole system in two parts; first is *control function* and second is *control laws*. We have formalised all controller functions like *uninterpreted functional* blocks. The

uninterpreted function blocks have same input and output parameters as the original controller functions, and we have assumed that the original control functions are correct, which are provided by the control engineers. The control laws are represented by discrete control logic. New development methodology is successfully applied for developing the adaptive cruise control (ACC) from requirement analysis to code implementation. The whole system development life-cycle is based on formal techniques. The complete system is designed using different kinds of tools related to the formal techniques. Event-B modeling language is used for formalizing the ACC system using refinement techniques. Each level of refinements is validated through the ProB model checker for verifying the correctness of the system behaviors against requirements and according to the controller and automotive experts. If any error is discovered during verification, validation or domain experts reviews, then the ACC specification is modified and again follow the verification, validation and domain experts review. This process is continued applied in the loop until not find the correct proved formal specification of the ACC system. The verification, validation and domains experts reviews are applied on each refinement level for modeling the whole system. To handle the complexity of the system according to the refinements, we have used the refinement chart to model the ACC system. The refinement charts of the ACC present integration architecture of a system in form of various switching modes. This technique is not only for integration, but also it helps for analyzing the switching modes of the system and also in the code structuring. A tool EB2ALL is used for generating the source code into multiple languages (C, C++, C#, and Java) using developed formal specifications. Finally, we have transformed the final proved Event-B models into equivalent Simulink models. It should be noted that in this development process, we have not considered the safety assessment approach.

The current work intends to explore those problems related to the modeling of controller systems. Formal verification of the control system is very crucial issue due to erratic behavior of the control functions. The formal representation of the ACC system, we have applied a rather different approach than existing approach. Refinement based formal methods techniques are used for verification of the ACC system. We have decomposed the whole system in two parts; first is the *control function* and second is the *control laws*. Block diagram (Fig. 10.2) represents a relationship between control function and control laws. Finally, we have transformed concrete level formal specification of the ACC system into equivalent *Simulink* [MATLAB] model. The developed formal model in the Event-B modeling language and transformed *Simulink* model are complied with requirements document. Some parts of the formal model are represented by *State-flow* and remaining parts of the model is presented by the *Simulink* [MATLAB]. Finally, we have executed this *Simulink* model and test the desired behavior of the ACC system.

10.10.2 Conclusion

A complete development of the adaptive cruise control (ACC) is represented from formal verification to the automatically code generation. In this development, we have applied our proposed approaches (see Part-I) related to the formal development, modeling of refinement charts, automatic code generation and directly Simulink model generation from the

proved ACC model. Our approach for formalizing and reasoning about the ACC system, which is mainly based on the controller (PID). The ACC case study is another kind of case study related to the automotive domains, which suggests that such an approach is useful for validation against system-level properties at the early stage of the development process.

Refinement is a key concept for developing complex systems, since it starts with a very abstract model and incrementally adds new details to the set of requirements. We have outlined how an incremental refinements approach for formalising a controller based automotive system using the Rodin tool. The approach, we have taken is not specific to the Event-B. We believe that similar approach can be used by other state-based notations such as ASM, TLA$^+$ or Z tools. The Rodin proof tool is used to generate the hundreds of proof obligations and to discharge those obligations automatically and interactively. Another key role of this tool is for helping us to discover appropriate gluing invariants to prove the refinements. In summary, some key lessons are that the incremental development with small refinement steps; appropriate abstractions at each level and powerful tool support are all invaluable in this kind of formal development.

The main advantage of proposed development life-cycle methodology and a set of associated techniques and tools are the ability to develop formal techniques based the ACC system. Proposed methodology exploits the advance capabilities of the combined approach of formal verification and model validation using a model-checker, automatic source code generation, and finally *Simulink* model generation from the verified formal model in order to achieve considerable advantages for a critical system design.

System integration methodology using refinement charts (see Fig. 10.5) are also used for system development, which helps a code designer to improve the code structure, code optimization, code generation for synthesizing, and synchronizing the software codes of a critical system like ACC system. In the ACC case study, system has different kinds of functional requirements, and all the possible operating modes are decomposed in the refinement chart using multiple refinements related to the state flow models. A set of requirements are formally represented in specifications. The correctness of formal specifications is the amounts of proof obligations (see Table 10.1). Therefore use of the refinement chart, and formal specifications state the correctness of the system design and implementation.

In this chapter, we have shown that formal verification of the hybrid control systems. The formal model of the hybrid control systems is verified, and this verified model is not only feasible but also useful for improving the existing hybrid control system. We have fully formalised a real-world hybrid control system like ACC system using an incremental refinement formalisation process, and we have used proof tools to systematically analyse whether the formalisation complies with certain properties. Throughout this process, we have obtained the following concrete results:

- A novel approach to model hybrid control system using refinement approach.
- Use of *"Separation of Concern"* in refinement approach and to define all controller functions as in the form of *uninterpreted function*.
- Verification of complex properties likes composition of individual controllers.

- A formal specification language Event-B that is used for modeling the complex system, is also used to model control systems. Event-B is the general modeling language tool. The Event-B is used to present a formal specification for a real-life control system in the automotive domain; Adaptive Cruise Control (ACC).
- Adaptive cruise control is used in our study has been developed in an incremental way and finally transformed into a concrete formal representation. Each proved refinement level of the formal model of the protocol represents feasibility and correctness of the ACC.
- Transformation of the ACC Event-B model into Simulink and State-flow models.
- Requirements Verification before the development of the Simulink model.

For generating the source code into different kinds of programming languages (C, C++, Java and C#), we have used a tool EB2ALL [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b]. We have used EB2ALL tool to demonstrate the ability to generate automatically source code from EVENT B specification of the ACC system in various high level programming languages like C, C++, Java and C#. Generated codes are comparable to the hand-written codes in any particular programming language. In order to assess the overall utility of our approach, a selection of the results of the formalisation and verification steps have been developed under supervision of a group of researchers (General Motor, India Science Lab, Bangalore, India). Throughout our case study, we have shown formal specification and verification of the ACC system. The controller based ACC models must be validated to ensure that they meet requirements. Hence, validation must be carried out by both formal modeling and domain experts. Our main objective behind this work is that the hybrid control system development and verify compositional architecture of the control theory, and behavior of the system must be met with desired behavior of the domain experts. This is our first attempt to develop a formal model of any hybrid control system and encounter all required properties and behavior of the PID controllers.

CHAPTER 11

Electrocardiogram (ECG) Interpretation Protocol using Formal Methods

"The fact that your patient gets well does not prove that your diagnosis was correct."
(Samuel J. Meltzer)

Today, an evidence-based medicine has given number of medical practice clinical guidelines and protocols. Clinical guidelines systematically assist practitioners with providing appropriate health care for specific clinical circumstances. However, a significant number of guidelines and protocols are lacking in quality. Indeed, ambiguity and incompleteness are more likely anomalies in medical practices. From last few years, many researchers have tried to address the problem of protocol improvement in clinical guidelines, but results are not sufficient since they believe on informal processes and notations. Our objective is to find anomalies and to improve the quality of medical protocols using well known formal techniques, such as Event-B. In this chapter; we use a modeling language to capture the guidelines for their validation. We have established a classification of the possible properties to be verified in a guideline. Our approach is illustrated with a guideline which published by the National Guideline Clearing House (NGC) and AHA/ACC Society. Our main contribution is to evaluate the real-life medical protocols using refinement based formal methods for improving quality of the protocols. Refinement based formalisation is very easy to handle any complex medical protocols. For this evaluation, we have selected a real-life reference protocol (ECG Interpretation) which covers a wide variety of protocol characteristics related to the several heart diseases. We formalise the given reference protocol, verify a set of interesting properties of the protocol and finally determine anomalies. Our main results are: to formalise an ECG interpretation protocol for diagnosing the ECG signal in an optimal way; to discover a hierarchical structure for the ECG interpretation efficiently using incremental refinement approach; a set of properties which should be satisfied by the medical protocol; verification proofs for the protocol and properties according to the medical experts; and perspectives of the potentials of this approach. Finally, we have shown the feasibility of our approach for analysing the medical protocols.

11.1 Introduction

A promising and challenging application area for the application of formal methods is a clinical decision making, as it is vital that the clinical decisions are sound. In fact, ensuring safety is the primary preoccupation of medical regulatory agencies. Medical guidelines are "systematically developed statements to assist practitioners and patient decisions about appropriate health care for specific circumstances" [Lohr 1990; Ten Teije 2006]. Based on updated empirical evidence; the medical protocols to provide clinicians with health-care testimonial and facilitate the spreading of high-standard practices. In fact, this way represents that adherence to protocol may reduce the costs of care up to 25% [Ten Teije 2006]. In order to reach their potential benefits, protocols must fulfill strong quality requirements. Medical bodies worldwide have made efforts in this direction, e.g. elaborating appraisal documents that take into account a variety of aspects, of both protocols and their development process. However, these initiatives are not sufficient since they rely on informal methods and notations. The informal methods and notations have not any mathematical foundation.

We are concerned with a different approach, namely the quality improvement of medical protocols through formal methods. In this chapter, we report on our experiences in the formalisation and verification of a medical protocol for diagnosis of the Electrocardiogram (ECG) [Méry 2011h; Méry 2011j]. The ECG signals are too complex for diagnosis. All kinds of diseases related to the heart are predictable using 12-lead ECG signals. A high number of medical guidelines for the ECG interpretation has been published in the literature and on the internet, making them more accessible. Currently, protocols are described using a combination of different formats, e.g. text, flow diagrams and tables. These approaches are used in form of informal processes and notations for analysing the medical protocols, which are not sufficient for medical practices. As a result, the ECG interpretation guidelines and protocols[1] still contain ambiguous, incomplete or even inconsistent elements.

The idea of our work is translating the informal descriptions of the ECG interpretation into a more formal language, with the aim of analysing a set of properties of the ECG protocol. In addition to the advantages of such a kind of formal verification, making these descriptions more formal can serve to expose problematic parts in the protocols.

Formal methods have well structure representation language with clear and well-defined semantics, which can be used for taxonomy verification of clinical the guidelines and medical protocols. The representation language represents guidelines and protocols explicitly and in a non-ambiguous way. The process of verification using formal semantic representation of guidelines and protocols to allow the determination of consistency and correctness.

Formal modelling and verification of medical protocol to have been carried out as a case study to assess the feasibility of this approach. Throughout our case study, we have shown formal specification and verification of medical protocols. The ECG interpretation protocol is very complex, ambiguous, incomplete and inconsistent.

The contribution of this chapter is to give a complete idea of formal development

[1] Guideline and protocol are different terms. The term protocol is used to represent a specialised version of a guideline. In this paper, we use them indistinctively.

of the ECG interpretation protocol, and we have discovered a hierarchical structure for the ECG interpretation efficiently using incremental refinement approach [Méry 2011h; Méry 2011j]. Same approach can be also applied for developing a formal model of the protocol of any other disease. Our approach is based on the Event-B [Cansell 2007; Abrial 2010] modeling language which is supported by the Rodin platform integrating tools for proving models and refinements of the models. Here, we present an incremental proof-based development to model and verify such interdisciplinary requirements in the Event-B [Cansell 2007; Abrial 2010]. The ECG interpretation models must be validated to ensure that they meet requirements of the ECG protocols. Hence, validation must be carried out by both formal modeling and medical domain experts.

We have used a general formal modeling tool like Event-B [Abrial 2010] for modeling a complex medical protocol related to diagnoses of the ECG signal. To apply a refinement based technique to model a medical protocol is our main objective. The Event-B supports refinement technique. The refinement supported by the Rodin [RODIN 2004] platform guarantees the preservation of safety properties. The safety properties are detection of an actual disease under the certain conditions. The behavior of the final system is preserved by an abstract model as well as in the correctly refined models. This technique is used to model a medical protocol more rigorously based on formal mathematics, which helps to find the anomalies and provide the consistency and correctness of the medical protocol. The current work intends to explore those problems related to the modeling of the ECG protocols. The formalization of the ECG protocol is based on the original protocol, and all the safety properties and related assumptions are verified with the medical experts. Moreover, an incremental development of the ECG interpretation protocol model helps to discover the ambiguous, incomplete or even inconsistent elements in current the ECG interpretation protocol.

The outline of the remaining chapter is as follows. Section 2 contains related work. Section 3 presents selection of medical protocol for formalisation. We give a brief outline of the ECG in Section 4. In Section 5, we explore the incremental proof-based formal development of the ECG interpretation protocol. The verification results are analyzed by statistical proof and lessons learned in Section 6. Finally, in Section 7, we conclude the chapter.

11.2 Related Work

Section 2 currently presents ongoing research work related to computer-based medical guidelines and protocols for clinical purposes. From past few years many languages have been developed for representing medical guidelines and protocols using various levels of formality based on expert's requirements. Although we have used the Event-B modeling language for guidelines and protocol representation in our case study. Various kinds of protocol representation languages like Asbru [Shahar 1998; Ten Teije 2006], EON [Samson 1996], PROforma [Fox 1998] and others [Peleg 2003; Wang 2002] are used to represent a formal semantics of guidelines and medical protocols.

Clinical guidelines are useful tools to provide some standardization and helps for im-

proving the protocols. A survey paper [Isern 2008] presents benefits and comparison through an analysis of different kinds of systems, which are used by clinical guidelines. This paper covers a wide scope of clinical guidelines related literatures and tools, which are collected from the medical informatics area.

An approach for improving guidelines and protocols is by evaluating the physician. Evaluation process involves the scenario and evidence based testing, which compares the actions. The actions are performed by physicians to handle particular patient case using testimonials that are prescribed by the guidelines [Miller 1985]. When results of the actions deviate, evaluation process can be either focused on the explanation alternatively provide some valuable feedback for improving the guidelines and protocols [Marcos 2001]. An intention based evaluation process are deduced by the physicians from both the patient data and the performed actions. These are then verified against the intentions reported in the guidelines.

Automated quality assessment of clinical actions and patient outcomes is another area of related work, which is used to derive structured quality indicators from formal specifications of guidelines. This technique is used in decision support [Advani 2003]. Such kinds of indicators is used as formal properties in our work that guideline must comply with.

Decision-table based techniques for the verification and simplification of guidelines are presented by Shiffman et al. [Shiffman 1994; Shiffman 1997]. The basic idea behind this approach is to describe guidelines as condition/action statements: *If the antecedent circumstances exist, then one should perform the recommended actions* [Shiffman 1997]. Completeness and consistency are two main properties for verification, when guidelines and protocols are expressed in terms of decision-table. Again, these properties are internal coherence properties, whereas we are focused on domain-specific properties.

Formal development of the guidelines and protocols using clinical logic may be incomplete or inconsistent. This problem is tackled by Miller et al. [Miller 1999]. *If "if-then" rules are used as representation language for guidelines, incompleteness means that there are combinations of clinically meaningful conditions to which the system (guideline) is not able to respond* [Miller 1999]. The verification of rule-based clinical guidelines using semantic constraints is supported by the commander tool. This tool is able to identify clinical conditions where the rules are incomplete. Miller et al. [Miller 1999] were able to find a number of missing rules in various case studies of the guidelines and protocols.

Guidelines enhancement is represented through adoption of an advanced Artificial Intelligence techniques [Bottrighi 2010]. This paper has proposed an approach for verification of the guidelines, which is based on the integration of a computerized guidelines management system with a model-checker. They have used SPIN model checker [Holzmann 1997; Clarke 1999] for executing and verifying medical protocols or guidelines. A framework for authoring and verification of clinical guidelines is provided by Beatriz et al [Pérez 2010]. The verification process of guidelines is based on combined approach of Model Driven Development (MDD) and Model Checking [Clarke 1999] to verify guidelines against semantic errors and inconsistencies. UML [Rumbaugh 1999; Warmer 2003] tool is used for modeling the guidelines, and a generated formal model is used as the input model for a model checker.

Jonathan et. al [Schmitt 2006] have proposed a way to apply formal methods, namely

interactive verification to improve the quality of medical protocols or guidelines. They have applied this technique for the management of jaundice in newborns based on guidelines of American Academy of Pediatrics. This paper includes formalisation of the jaundice protocol and verifies some interesting properties. Simon et. al [Bäumler 2006] have used the same protocol for improvement purpose using a modeling language Asbru, temporal logic for expressing the quality requirements, and model checking for proof and error detection.

Applying a formal approach for improving medical protocol is one major area of research, which helps to the medical practitioners for improve the quality of patient care. A project Protocure [Michael Balser 2004] is a European project, which is carried out by five different institutions. The main objective of this project is for improving medical protocol through integration of formal methods. The main motivation of this project is to identify anomalies like ambiguity and incompleteness in the medical guidelines and protocols. Presently, all medical protocols and guidelines are in text, flow diagrams and tables formats, which are easily understandable by the medical practitioners. But these are incomplete and ambiguous due to lack of formal semantics. The idea of using formal methods is to uncover these ambiguous, incomplete or even inconsistent parts of the protocols, by defining all the different descriptions more precisely using a formal language and to enable verification. Mainly, the researchers have used Asbru [Ten Teije 2006] language for protocol description and KIV for interactive verification system [Balser 1999].

Asbru [Ten Teije 2006] is a main modeling language for describing medical protocol and formal proof of the medical protocol is possible through KIV interactive theorem prover [Balser 1999]. Guideline Markup Tool(GMT) [Kosara 2002] is an editor who helps translating guidelines into Asbru. An additional functionality of the tool is to define relations between the original protocol and its Asbru translation with a link macro [Kosara 2002]. Asbru language is used for protocol description and Asbru formalizations are translated into KIV. Asbru is considered as a semi-formal language to support the tasks necessary for protocol-based care. It is called a semi-formal language because of its semantics, although more precise than in other protocol representation languages, are not defined in a formal way. This semi-formal quality makes Asbru suitable for an initial analysis but not for systematic verification of protocols [Miksch 2005].

According to our literatures survey, existing medical protocol tools are based on semi-formal techniques. Existing techniques [Bottrighi 2010; Miller 1999; Ten Teije 2006] based on formal techniques are failed to scale the complexity of the protocol. They have not given any proper idea to model the medical protocols only using formal techniques due to complex nature of the medical protocol. To tackle the complexity of the protocol in formal methods is only a solution to use the refinement approach to model the whole protocol from abstract level to a final concrete model. In this chapter, we have provided sufficient detailed information about modeling a complex protocol using any formal method technique. In this study, we have tried to model a medical protocol, completely based on formal semantics and to check various anomalies. To overcome from the existing problems [Seyfang 2006; Miksch 2005] in the area of development of medical protocols, we have used the general formal modeling tool like Event-B [Abrial 2010] for specifying a complex medical protocol related to the diagnoses of ECG signal. The main objective to use Event-B modeling language is to model medical protocols using the refinement

approach. The medical protocols are very complex and to model a complex protocol, a refinement approach is very helpful, which introduces peculiarity of the protocols in an incremental way. This technique is used to model a medical protocol more rigorously based on formal mathematics, which helps to find the anomalies and provide the consistency and correctness of the medical protocol.

11.3 Selection of Medical Protocol

Concerning the protocols that is the object of our study, we have selected the ECG interpretation that covers a wide range of protocol characteristics related to the heart diseases. All kinds of medical guidelines and protocols to differ from each others along several dimensions, which can be referred to the contents of the protocols or to its form. General practitioners (GPs), nurses and a large group of people related to this domain[2] are the *most important target users* of the guidelines and protocols, and the main aspects of clinical practice are to cover *diagnosis* as well as help in treatments. The medical guidelines and protocols, which are used by general practitioners and nurses, are also characterized by time dimensions; short time-span protocols; long-time span protocols. The form of guidelines and protocols are related to the textual descriptions. Sometimes it is also represented in the textual form as well as the combination with *tables* and *flowcharts*.

The ECG interpretation protocol [Khan 2008; Barold 2004] aims at cardiologist as well as GPs and covers both diagnosis and treatment over a long period of time. The ECG interpretation protocol can be considered more precisely: one is in daily use by cardiologist, and the other is included in the repository of the National Guideline Clearinghouse(NGC), American College of Cardiology/American Heart Association (ACC/AHA). The basic standard for inclusion in the NGC and ACC/AHA are that the guidelines and protocols to contain well structured meaningful informations and systematically developed statements. The contents are produced under the supervision of medical specialty associations. It should be also based on literatures, reviewed and revised within the last 5 years. Furthermore, the ECG interpretation protocol has been published in a peer-reviewed scientific journal. In summary, the chosen protocol covers different aspects while fulfilling high-quality standards, which are the good criteria for selection of our case study.

In the following sections, we will use the ECG interpretation protocol as the main example in our explanations, and we therefore give a brief description of this protocol. The Electrocardiogram (ECG or EKG) interpretation is a common technique to trace abnormalities in the heart system and various levels of tracing help to find severe diseases. The guideline is more than 100 pages document, which contains knowledge in various notations: the main text; a list of factors to be considered when assessing an abnormality in the heart ECG signal and a flowchart describing the steps in the ECG interpretation protocol. The protocol consists of an evaluation (or diagnosis) part and a treatment part, to be performed in the successive way. During the application of guidelines and protocols, as soon as the possibility of a more serious disease is uncovered, the recommendation is to leave the protocol without any further actions.

[2] http://www.guideline.gov/

11.4 Basic overview of Electrocardiogram (ECG)

The electrocardiogram (ECG or EKG) [Khan 2008; Hesselson 2003] is a diagnostic tool that measures and records the electrical activity of the heart precisely in the form of signals. Clinicians can evaluate the conditions of a patient's heart from the ECG and perform further diagnosis. Analysis of these signals can be used for interpreting diagnosis of a wide range of the heart conditions and to predict the related diseases. The ECG records are obtained by sampling the bioelectric currents sensed by several electrodes, known as leads. A typical one-cycle ECG tracing is shown in Fig. 11.1. Electrocardiogram term is introduced by Willem Einthoven in 1893 at the meeting of Dutch Medical Society. In 1924, Einthoven received the Nobel Prize for his life's work in developing the ECG [Khan 2008; Writing Committee 2008; Barold 2004; Love 2006; Ellenbogen 2005; Hesselson 2003; Malmivuo 1995].

The normal electrocardiogram (ECG or EKG) is depicted in Fig. 11.1. All kinds of segments and intervals are represented in this ECG diagram. The depolarization and repolarization of the ventricular and atrial chambers are presented by deflection of the ECG signal. All these deflections are denoted by alphabetic order (P-QRS-T). Letter P indicates the atrial depolarization, and the ventricular depolarization is represented by QRS complex. The ventricular repolarization is represented by T-wave. The atrial repolarization appears during the QRS complex and generates a very low amplitude signal which cannot be uncovered from a normal ECG signal.

11.4.1 Differentiating the P-, QRS- and T-waves

Sequential activation, depolarization, and repolarization are deflected distinctly in the ECG due to anatomical difference of the atria and ventricles. Even all sequences are easily distinguishable when they are not in a correct sequence: P-QRS-T. The QRS-complex is easily identifiable between P- and T-waves because it has characteristic waveform and dominating amplitude. This amplitude is about 1000 μm in a normal heart and can be much greater in the ventricular hypertrophy. Normal duration of the QRS-complex is 80-90 ms. In case of non-existence of the atrial hypertrophy; an amplitude and duration of the P-wave are about 100 μm and 100 ms, respectively. The T-wave has about twice of the amplitude and duration of the P-wave. The T-wave can be differentiated from the P-wave by observing that the T-wave follows the QRS-complex after about 200 ms. In the ECG signal several parameters are used to evaluate the conditions of a patient's heart from the ECG. The parameters are: PR-interval, P-wave, QRS duration, Q-wave, R-wave, ST-segment, T-wave, Axis, QT-interval. All these parameters have several different characteristics that are used for diagnosis.

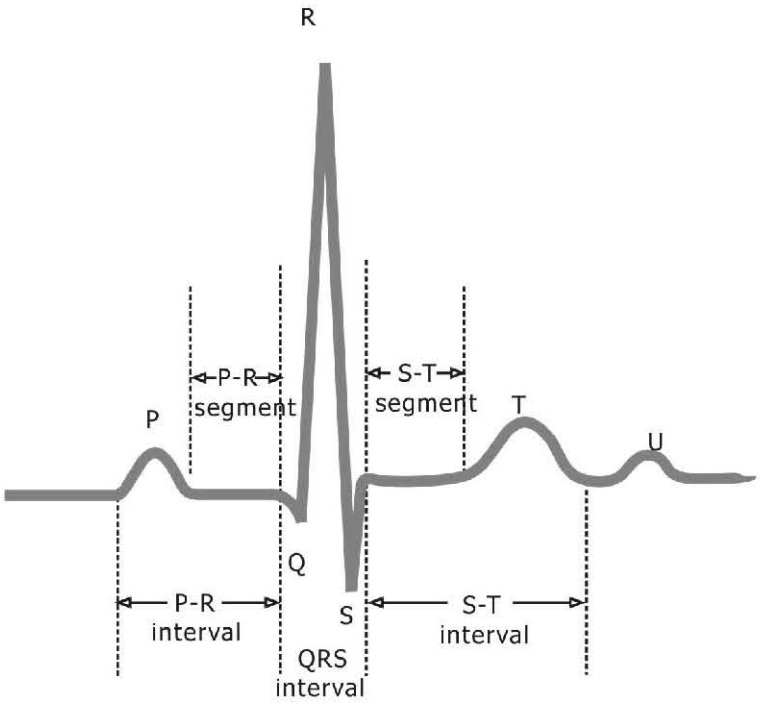


Figure 11.1: A typical one-cycle ECG tracing [Malmivuo 1995]

11.5 Formal Development of the ECG Interpretation

11.5.1 Abstract Model : Assessing Rhythm and Rate

We begin by defining the Event-B context. The context uses sets and constants to define axioms and theorems. Axioms and theorems represent the logical theory of a system. The logical theory is the static properties and properties of the target system. In the context, we define constants *LEADS*, *HState* and *YesNoState* that are related to an enumerated set of the ECG leads, normal and abnormal states of the heart and yes-no states, respectively. These constants are extracted from the ECG interpretation protocol [Khan 2008; Writing Committee 2008; Ellenbogen 2005; Hesselson 2003]. The standard 12-lead electrocardiogram is a representation of the heart's electrical activity recorded from electrodes on the body surface. A set of leads is represented as *LEADS = {I, II, III, aVR, aVL, aVF, V1, V2, V3, V4, V5, V6}*. Normal and abnormal states of the heart are represented by *HState = {OK, KO}* and yes-no states are represented by *YesNoState = { Yes, No }*. Fig. 11.2 depicts an incremental formal development of the ECG interpretation protocol. In our development process, some refinements are decomposed into several refinements for the simplicity. Every refinement level introduces a *diagnosis* criteria for different components of the ECG signal, and each new criteria helps to analyse a particular set of diseases. A particular set of diseases is introduced in the multiple context related to each refinement.

Fig. 11.3 shows an abstract representation of a *diagnostic-based* system development,

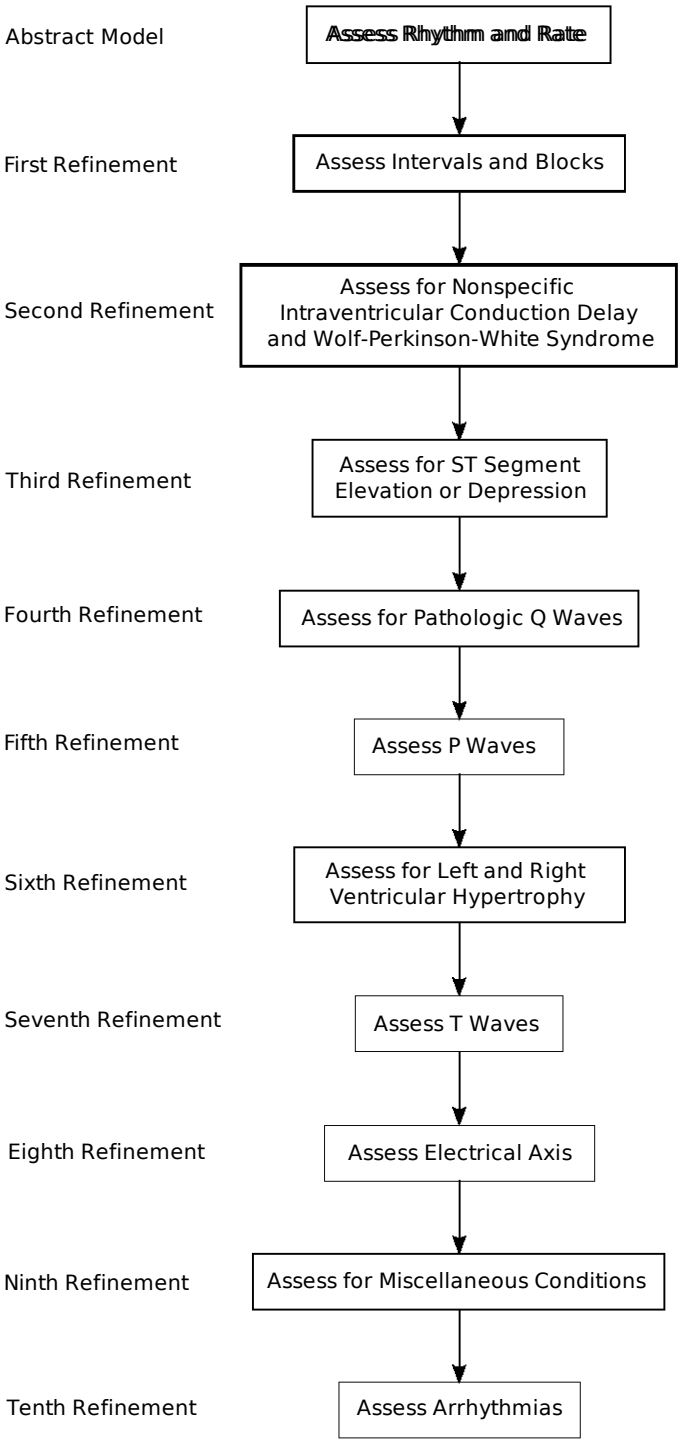


Figure 11.2: ECG interpretation protocols refinements

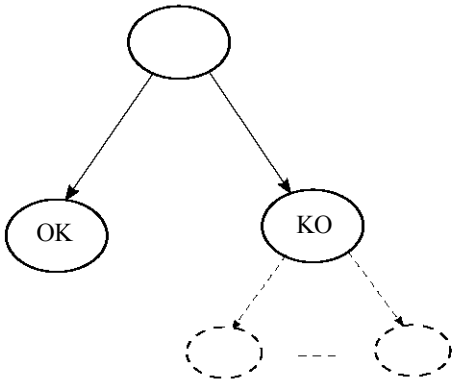


Figure 11.3: Abstract Representation

where a root node (top circle in Fig. 11.3) represents a set of conditions for testing any particular disease abstractly. The possible abstract outcomes of a diagnosis criterion are in form of *OK* and *KO*, which are represented by two branches. The *KO* represents that the diagnosis criteria have found some conditions for further testing, while the *OK* represents the absence of any disease. The dash line of circles and arrows represent the next level of refinement for further analysing of any particular diseases according to the guidelines and protocol.

Our abstract Event-B model of the ECG interpretation protocol assesses the *rhythm* and *heart rate* to distinguish between normal and abnormal heart. Fig. 11.4 presents a basic diagram of the ECG analysis at an abstract level according to the standard procedure of the ECG protocol analysis. The specification consists of just three-state variables ($inv1 - int3$) *Sinus*, *Heart_Rate* and *Heart_State*. The *Sinus* variable is represented by *YesNoState* as enumerated sets. The last two variables *Heart_Rate* and *Heart_State* are introduced as to show the heart rate limit and heart states. One possible approach is to introduce a set of variables (*RR_Int_equidistant*, *PP_Int_equidistant*, *P_Positive*, *PP_Interval* and *RR_Interval*) representing total functions mapping leads (LEADS) to a standard data type (*BOOL*, $\mathbb{N}$) in invariants ($inv4 - inv8$). The RR and PP equidistant intervals in the ECG signal are represented by variables *RR_Int_equidistant* and *PP_Int_equidistant* as the total functions from *LEADS* to *BOOL*. The *RR_Int_equidistant* and *PP_Int_equidistant* are functions, which represent RR and PP equidistant interval's states in a boolean form. A variable *P_Positive* represents a positive wave of the signal also as a total function from *LEADS* to BOOL. The *P_Positive* function is used to show the positive visualization of the P-waves. The next variables PP and RR intervals in the ECG signal are represented by the variables *PP_Interval* and *RR_Interval* as the total functions from *LEADS* to $\mathbb{N}$. The *PP_Interval* and *RR_Interval* functions are used to calculate the PP and RR-intervals.

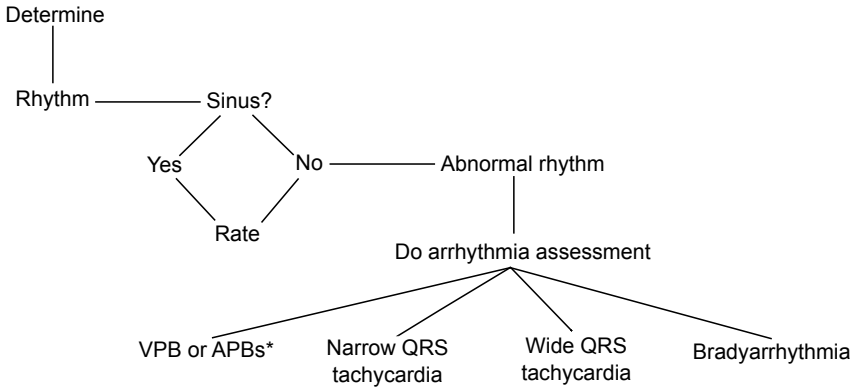


Figure 11.4: Basic Diagram of Assessing Rhythm and Rate [Khan 2008]

$$
\begin{array}{l}
inv1 : Sinus \in YesNoState \\
inv2 : Heart\_Rate \in 1 \mathinner{..} 300 \\
inv3 : Heart\_State \in HState \\
inv4 : RR\_Int\_equidistant \in LEADS \rightarrow BOOL \\
inv5 : PP\_Int\_equidistant \in LEADS \rightarrow BOOL \\
inv6 : P\_Positive \in LEADS \rightarrow BOOL \\
inv7 : PP\_Interval \in LEADS \rightarrow \mathbb{N} \\
inv8 : RR\_Interval \in LEADS \rightarrow \mathbb{N} \\
inv9 : P\_Positive(II) = FALSE \Rightarrow Sinus = No \\
inv10 : ((\forall l \cdot l \in \{II, V1, V2\} \\
\qquad \Rightarrow \\
\qquad PP\_Int\_equidistant(l) = FALSE \lor \\
\qquad RR\_Int\_equidistant(l) = FALSE \lor \\
\qquad RR\_Interval(l) \neq PP\_Interval(l)) \\
\qquad \lor \\
\qquad P\_Positive(II) = FALSE) \Rightarrow Sinus = No \\
inv11 : Sinus = Yes \Rightarrow ((\exists l \cdot l \in \{II, V1, V2\} \land \\
\qquad PP\_Int\_equidistant(l) = TRUE \land \\
\qquad RR\_Int\_equidistant(l) = TRUE \land \\
\qquad RR\_Interval(l) = PP\_Interval(l)) \\
\qquad \land \\
\qquad P\_Positive(II) = TRUE) \\
inv12 : Heart\_Rate \in 60 \mathinner{..} 100 \land Sinus = Yes \\
\qquad \Rightarrow \\
\qquad Heart\_State = OK \\
inv13 : Heart\_Rate \in 1 \mathinner{..} 300 \setminus 60 \mathinner{..} 100 \land Sinus = Yes \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv14 : Heart\_Rate \in 60 \mathinner{..} 100 \land Sinus = No \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO
\end{array}
$$

A set of invariants ($inv9 - inv14$) represents the safety properties, and these are used to verify the required conditions for the ECG interpretation protocol using all possible be-

havior of the heart system and analysis of the signal features, which are collected from the ECG signals. All these safety properties are designed under the supervision of cardiologist experts to verify the correctness of the formal model. These invariants in form of safety properties are extracted from the original protocol.

The invariant ($inv9$) states that if positive visualization of the P-wave is *FALSE*, then there is no sinus rhythm. According to the clinical document, lead II is best for visualization of the P-waves to determine the presence of sinus rhythm. The next invariant ($inv10$) is stronger invariant to identify the non-existence of the sinus rhythm. This invariant states that if the PP intervals (*PP_Int_equidistant*) or RR intervals (*RR_Int_equidistant*) is not equidistant (*FALSE*), or the RR intervals (*RR_Interval*) and PP intervals (*PP_Interval*) are not equivalent, in all leads (II,V1,V2), or positive visualization of the P-wave in lead II is *FALSE*, then there is no sinus rhythm. Similarly, next invariant ($inv11$) confirms, if the rhythm is sinus, then the PP intervals (*PP_Int_equidistant*) and RR intervals (*RR_Int_-equidistant*) are equidistant, and the RR intervals (*RR_Interval*) and PP intervals (*PP_-Interval*) are equal, exist in any leads (II,V1,V2), and the P-wave is positive in lead II. The invariant ($inv12$) represents that if the heart rate (*Heart_Rate*) is belonging between 60-100 bpm and the sinus rhythm is *Yes*, then the *Heart_State* is *OK*. The next two invariants ($inv13 - inv14$) represent *KO* state of the Heart, mean the heart has any disease. The invariant ($inv13$) states that if the heart rate (*Heart_Rate*) is belonging between less than 60 bpm and greater than 100 bpm but less than 300 bpm, and the sinus rhythm is *Yes*, then the heart state (*Heart_State*) is KO. Similarly, the last invariant ($inv14$) represents that if the heart rate (*Heart_Rate*) is in between 60-100 bpm and the sinus rhythm is *No*, then the *Heart_State* is *KO*, means heart has any disease.

Three significant events *Rhythm_test_TRUE*, *Rhythm_test_FALSE* and *Rhythm_test_-TRUE_abRate* are introduced in the abstract model. The *Rhythm_test_TRUE* represents successful ECG testing and found the sinus rhythm *Yes* and the heart state is *OK* . The next event *Rhythm_test_FALSE* represents successful ECG testing and found the sinus rhythm is *No* and the heart state is *KO*. Third event *Rhythm_test_TRUE_abRate* represents successful ECG testing and found the sinus rhythm is *Yes* and the heart state is *KO* due to abnormal heart rate. These events are the abstract events, which are equivalent to the first step of diagnosis of the ECG signal of the original protocol. We have taken some assumptions for modeling the medical protocol. These assumptions are extracted from the original protocol. In our formal model, all invariants and assumptions are verified with the medical experts. Our developed formal models are always complied with existing original protocols.

Mostly, events are used to test the criteria of possible disease using the ECG features. The criteria for testing the sinus rhythm is to focus on leads V1, V2, and II. The leads V1 and II are best for visualization of the P-waves to determine the presence of the sinus rhythm or an arrhythmia, and the V1 and V2 are best to observe for the bundle branch block. If the P-waves are not clearly visible in V1, assess them in lead II, which usually shows well-formed P-waves [Khan 2008]. Identification of the P-wave and then the RR intervals allow the interpreter to discover immediately whether the rhythm is sinus or other and to take the following steps:

- Confirm, if the rhythm is sinus, that the RR intervals are equidistant, that the P-wave

is positive in lead II, and that the PP intervals are equidistant and equal to the RR interval.

- Do an arrhythmia assessment if the rhythm is abnormal.
- Determine the heart rate.

EVENT Rhythm_test_TRUE
ANY $rate$
WHEN
grd1 : $(\exists l \cdot l \in \{II, V1, V2\} \land PP\_Int\_equidistant(l) = TRUE \land$
$RR\_Int\_equidistant(l) = TRUE \land$
$RR\_Interval(l) = PP\_Interval(l)) \land$
$P\_Positive(II) = TRUE$
grd2 : $rate \in 60 \,..\, 100$
THEN
act1 : $Sinus := Yes$
act2 : $Heart\_Rate := rate$
act3 : $Heart\_State := OK$
END

EVENT Rhythm_test_FALSE
ANY $rate$
WHEN
grd1 : $(\forall l \cdot l \in \{II, V1, V2\} \Rightarrow PP\_Int\_equidistant(l) = FALSE$
$\lor RR\_Int\_equidistant(l) = FALSE \lor$
$RR\_Interval(l) \neq PP\_Interval(l)) \lor$
$P\_Positive(II) = FALSE$
grd2 : $rate \in 1 \,..\, 300$
THEN
act1 : $Sinus := No$
act2 : $Heart\_Rate := rate$
act3 : $Heart\_State := KO$
END

EVENT Rhythm_test_TRUE_abRate
ANY $rate$
WHEN
grd1 : $(\exists l \cdot l \in \{II, V1, V2\} \land PP\_Int\_equidistant(l) = TRUE \land$
$RR\_Int\_equidistant(l) = TRUE \land$
$RR\_Interval(l) = PP\_Interval(l)) \land$
$P\_Positive(II) = TRUE$
grd2 : $rate \in 1 \,..\, 300 \setminus 60 \,..\, 100$
THEN
act1 : $Sinus := Yes$
act2 : $Heart\_Rate := rate$
act3 : $Heart\_State := KO$
END

In the abstract model, we have seen that the sinus rhythm and heart rate are introduced for the ECG interpretation in a single atomic step. This provides for a clear and simple specification of the essence of the basic ECG interpretation protocol and predicts the heart state (OK or KO). However, in the real protocol, the ECG interpretation and heart state prediction is not atomic. Instead, the ECG interpretation and prediction are also encounter lots of diagnosis to find the various kinds of heart diseases.

11.5.2 Overview of the Full Refinement Chain

So far, we have described our abstract model of the ECG interpretation protocol. Every level of refinement introduces new context file for adding static properties of the system and list of heart diseases after introducing certain protocol of the ECG interpretation. Every refinement level is used to introduce a new set of diagnosis criteria to test the ECG signals. Rather than presenting the other chain of refinement stages in similar detail as an abstract model, we will just present a sufficient overview of the remaining refinement stages helping a reader to understand the rational of each refinement stage for formalising the ECG interpretation protocol.

11.5.2.1 First Refinement : Assess Intervals and Blocks

In an abnormal ECG signal, all ECG features are varying according to symptoms of the heart diseases. We will formalise the ECG interpretation protocol using an incremental approach, where we determine all features of the ECG signal. This level of refinement determines the PR- and QRS-intervals for the ECG interpretation. These intervals classify different kinds of the heart disease.

Invariants ($inv1 - inv3$) represent a set of new introduced variables in the refinement for expressing formalisation of the ECG interpretation protocol. These variables are *PR_Int*, *Disease_step2*, *QRS_Int*. Other variables (*M_Shape_Complex*, *Slurred_S*, *Notched_R*, *Small_R_QS* and *Slurred_S_duration*) are introduced as total functions in invariants ($inv4 - inv8$) where total functions are mapping from leads (LEADS) to *BOOL* and $\mathbb{N}_1$, respectively. The function *M_Shape_Complex* returns existence of M-shape complex from the ECG signals in form of *TRUE* or *FALSE*. The function *Slurred_S* represents Slurred S-wave, the function *Notched_R* represents notched R-wave and the function *Small_R_QS* represents small R or QS waves, in boolean type. The function *Slurred_S_duration* is used to calculate Slurred-S duration.

A set of invariants ($inv9 - inv14$) represent safety properties to validate formal representation of the ECG interpretation protocol. All these properties are derived from the original protocol to verify the correctness and consistency of the system. These properties are formulated through logic experts as well as cardiologist experts according to the original protocol. The main advantage of this technique is that if any property is not holding by the model, then it helps to find anomalies or to find missing parts of the model such as required conditions and parameters.

Invariants ($inv9 - inv13$) represent an abnormal state of the heart (*KO*) due to finding any disease and unsatisfiable condition for features of the ECG signal, in the formal

diagnosis process. While the last invariant ($inv14$) represents all required properties for a normal heart. It states that if the heart rate is in between 60 to 100 bpm, the sinus rhythm is *Yes*, the PR interval is less than or equal to 200 ms and the QRS interval is less than 120 ms, then the heart state is *OK*.

$$
\begin{array}{l}
inv1 : PR\_Int \in 120 \mathinner{..} 250 \\
inv2 : Disease\_step2 \in Disease\_Codes\_Step2 \\
inv3 : QRS\_Int \in 50 \mathinner{..} 150 \\
inv4 : M\_Shape\_Complex \in LEADS \rightarrow BOOL \\
inv5 : Slurred\_S \in LEADS \rightarrow BOOL \\
inv6 : Notched\_R \in LEADS \rightarrow BOOL \\
inv7 : Small\_R\_QS \in LEADS \rightarrow BOOL \\
inv8 : Slurred\_S\_duration \in LEADS \rightarrow \mathbb{N}_1 \\
inv9 : Sinus = Yes \wedge PR\_Int > 200 \wedge Disease\_step2 = First\_degree\_AV\_Block \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv10 : Sinus = Yes \wedge QRS\_Int \geq 120 \wedge Disease\_step2 \in \{LBBB, RBBB\} \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv11 : Sinus = Yes \wedge Disease\_step2 = First\_degree\_AV\_Block \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv12 : Sinus = Yes \wedge Disease\_step2 = LBBB \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv13 : Sinus = Yes \wedge Disease\_step2 = RBBB \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv14 : Heart\_Rate \in 60 \mathinner{..} 100 \wedge Sinus = Yes \wedge PR\_Int \leq 200 \wedge QRS\_Int < 120 \\
\qquad \Rightarrow \\
\qquad Heart\_State = OK
\end{array}
$$

To express formal logic for a new set of diagnoses for the ECG signal, we have introduced three events *PR_Test*, *QRS_Test_LBBB* and *QRS_Test_RBBB*. The *PR_Test* interval represents, if the PR intervals are abnormal (>200 ms), then consider the first-degree atrioventricular (AV) block. The next two events *QRS_Test_LBBB* and *QRS_Test_RBBB* are used to assess the QRS duration for the bundle branch block and states that, if the QRS interval is ≥120 ms, the bundle branch block is present. Understanding the genesis of the QRS complex is an essential step and clarifies the ECG manifestations of bundle branch blocks [Khan 2008]. We have formalised the basic criteria to distinguish between RBBB and LBBB in the diagnosis process. The basic description of RBBB and LBBB are given as follows:

Right Bundle Branch Block (RBBB)

- QRS duration ≥120 ms.
- M-shaped complex in V1 and V2.

- Slurred S-wave in leads 1, V5, V6; and an S-wave that is of greater amplitude (length) than the preceding R-wave.

Left Bundle Branch Block (LBBB)

- QRS duration $\geq$120 ms.
- A small R- or QS-wave in V1 and V2.
- A notched R-wave in leads 1, V5, and V6.

Due to limited space, we will not show the formal representation of introduced events. For the complete details with formal specifications see [Méry 2011j].

11.5.2.2 Second Refinement : Assess for Nonspecific Intraventricular Conduction Delay and Wolff-parkinson-white Syndrome

This level of refinement of the ECG interpretation assesses for nonspecific intraventricular conduction delay (IVCD) and wolff-parkinson-white (WPW) syndrome. The WPW syndrome may mimic an inferior MI (see in further refinements). If The WPW syndrome, RBBB, or LBBB is not present, interpret as nonspecific intraventricular conduction delay (IVCD) and assess for the presence of electronic pacing [Khan 2008]. Some new variables (*Delta_Wave* and *Disease_step3*) are introduced in this refinement to assess atypical right bundle branch block using ECG signal. Two invariants ($inv3 - inv4$) are used to declare new variables in form of the total functions mapping leads (LEADS) to *BOOL*. These functions are used to calculate the ST-segment elevation and epsilon wave, respectively. Invariants ($inv5 - inv8$) represent an abnormal state of the heart (*KO*) when the sinus rhythm is *Yes* and any new particular disease is found in this refinement. All these properties are derived from the original protocol to verify the correctness and consistency of the system according to the cardiologist.

$$
\begin{array}{l}
inv1 : Delta\_Wave \in \mathbb{N} \\
inv2 : Disease\_step3 \in Disease\_Codes\_Step3 \\
inv3 : ST\_elevation \in LEADS \rightarrow BOOL \\
inv4 : Epsilon\_Wave \in LEADS \rightarrow BOOL \\
inv5 : Sinus = Yes \wedge Disease\_step3 = WPW\_Syndrome \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv6 : Sinus = Yes \wedge Disease\_step3 = Brugada\_Syndrome \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv7 : Sinus = Yes \wedge Disease\_step3 = RV\_Dysplasia \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv8 : Sinus = Yes \wedge Disease\_step3 = IVCD \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO
\end{array}
$$

We have introduced four events *QRS_Test_Atypical_RLBBB_WPW_Syndrome*, *QRS_-Test_At-ypical_RBBB_Brugada_Syndrome*, *QRS_Test_Atypical_RBBB_RV_Dysplasia* and *QRS_Test_Aty-pical_RBBB_IVCD* to interpret atypical right bundle branch block using QRS interval. The following criteria to assess are as follows:

- If the QRS duration is prolonged $\geq$110 ms and bundle branch block appears to be present but is atypical, consider WPW syndrome, particularly if there is a tall R wave in leads V1 and V2.
- Assess for a short PR interval $\leq$120 ms and for a delta wave.

11.5.2.3 Third Refinement : Assess for ST-segment Elevation or Depression

This refinement provides a criteria for the ST-segments assessment by introducing some new variables (*ST_seg_ele* and *ST_depression*) in form of total functions mapping leads (LEADS) to $\mathbb{N}$ in invariants ($inv2-inv3$). The ST-segment for elevation and ST depression features are calculated by the *ST_seg_ele* and *ST_depression* functions. Invariants ($inv4-inv8$) are introduced for representing the safety properties to confirm an abnormal state of the heart (*KO*) when sinus rhythm is *Yes* and a new disease is found in this refinement.

$$
\begin{array}{l}
inv1 : Disease\_step4 \in Disease\_Codes\_Step4 \\
inv2 : ST\_seg\_ele \in LEADS \rightarrow \mathbb{N} \\
inv3 : ST\_depression \in LEADS \rightarrow \mathbb{N} \\
inv4 : Sinus = Yes \wedge Disease\_step4 \in \{Acute\_inferior\_MI, Acute\_anterior\_MI\} \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv5 : Sinus = Yes \wedge Disease\_step4 = STEMI \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv6 : Sinus = Yes \wedge Disease\_step4 \in \{Troponin, CK\_MB\} \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv7 : Sinus = Yes \wedge Disease\_step4 = Non\_STEMI \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv8 : Sinus = Yes \wedge Disease\_step4 = Ischemia \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO
\end{array}
$$

Four new events *ST_seg_elevation_YES*, *ST_seg_elevation_NOTCKMB_Yes*, *ST_seg_-elevation_-NO_TCKMB_No* and *Acute_IA_MI* are defined to cover *diagnosis* related to the ECG signals. All these events are used to interpret about the ECG signal using ST-segment elevation or depression features [Khan 2008]. To assess the ST-segments elevation or depression; we have formalised the following textual criteria:

- Focus on the ST-segment for elevation or depression. ST-elevation $\geq$1000 μm (0.1 mV) in two or more contiguous ECG leads in a patient with chest pain indicates ST

elevation MI (STEMI). The diagnosis is strengthened if there is reciprocal depression.

- ST-elevation in leads II, III, and aVF, with marked reciprocal depression in leads I and aVL, diagnostic of acute inferior MI.
- ST-segment elevation in V1 through V5, caused by extensive acute anterior MI.
- The ECG of a patient with a subtotal occlusion of the left main coronary artery. Note the ST elevation in aVR is greater than the ST elevation in V1, a recently identified marker of left main coronary disease.
- Features of non-ST-elevation MI (non-Q-wave MI).
- Elevation of the ST-segment may occur as a normal variant and ST-segment abnormalities and MI.

These textual sentences are formulated in the incremental development of our ECG protocol. This refinement advises scrutiny of the ST-segment before assessment of the T-waves, electrical axis, QT interval, and hypertrophy because the diagnosis of acute MI or ischemia is vital and depends on careful assessment of the ST-segment. Above given criteria are more complex and too ambiguous to represent. Therefore, we have formalised this part through careful cross reading of many reliable sources such as literature and encounter suggestions of the medical experts.

11.5.2.4 Fourth Refinement : Assess for Pathologic Q-wave

This refinement only introduces new guidelines to interpret Q-wave feature of the ECG signal and assessment-related diseases to the Q-wave and R-wave [Khan 2008]. Some new variables are represented by a set of invariants ($inv1 - inv2$) to handle the required features of the Q-wave and R-wave to diagnose the ECG signal. The functions *Q_Normal_Status* and *R_Normal_Status* represent the normal state of the Q and R-waves in a boolean type. The next three invariants ($inv3 - inv5$) are used to declare new variables in form of total functions mapping leads (LEADS) to $\mathbb{N}$, and an invariant ($inv6$) is also total function mapping leads (LEAD) to *BOOL*. The functions *Q_Width*, *Q_Depth* and *R_Depth* calculate the Q-wave width, Q-wave depth and R-wave depth, respectively. The last function *Q_Wave_State* represents the boolean state of the Q-wave for all leads. Two other new variables *Age_of_Inf* and *Mice_State* represent infarction age and miscellaneous states. An enumerated set of infarction age and miscellaneous states define as *Age_of_Infarct = {recent, indeterminate, old}* and *Mice_State5 = {Exclude_Mimics_MI, late_transition, normal_variant, borderline_Qs,NMS}*, respectively in the context. The variable *Disease_step5* represents a group of diseases of this refinement level as analysis of the Q-wave from the ECG signals. Some invariants ($inv10 - inv13$) are introduced as representing the safety properties to confirm an abnormal state of the heart (*KO*). All invariants have similar form for checking the heart state under the various disease conditions. These invariants state that if the sinus rhythm is *Yes* and a new disease is found, then the heart must be in the abnormal (*KO*) state.

$$
\begin{array}{l}
inv1 : Q\_Normal\_Status \in BOOL \\
inv2 : R\_Normal\_Status \in BOOL \\
inv3 : Q\_Width \in LEADS \rightarrow \mathbb{N} \\
inv4 : Q\_Depth \in LEADS \rightarrow \mathbb{N} \\
inv5 : R\_Depth \in LEADS \rightarrow \mathbb{N} \\
inv6 : Q\_Wave\_State \in LEADS \rightarrow BOOL \\
inv7 : Age\_of\_Inf \in Age\_of\_Infarct \\
inv8 : Mice\_State \in Mice\_State5 \\
inv9 : Disease\_step5 \in Disease\_Codes\_Step5 \\
inv10 : Sinus = Yes \wedge Disease\_step4 = Acute\_anterior\_MI \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv11 : Sinus = Yes \wedge Disease\_step4 = Acute\_inferior\_MI \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv12 : Sinus = Yes \wedge Disease\_step5 = Hypertrophic\_cardiomyopathy \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv13 : Sinus = Yes \wedge Disease\_step5 \in \\
\qquad \{anterior\_MI, LVH, emphysema, lateral\_MI\} \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO
\end{array}
$$

In this level of refinement, we have introduced nine events (*Q_Assessment_Normal*, *Q_Assess-ment_Abnormal_AMI*, *Q_Assessment_Abnormal_IMI*, *Determine_Age_of_Infarct*, *Exclude_Mimics*, *R_Assessment_Normal*, *R_Assessment_Abnormal*, *R_Q_Assessment_R_-Abnormal_V1234* and *R_Q_Assessment_R_Abnormal_V56*) for assessing the Q-wave and R-wave in all leads of the ECG signals. We have represented the formal notation of following guidelines, which are used to assess the Q-wave and the R-wave:

- Assess for the loss of R waves-pathologic Q-waves in leads I, II, III, aVL, and aVF.
- Assess for R wave progression in V2 through V4. The variation in the normal QRS configuration that occurs with rotation. The R wave amplitude should measure from 1 mm to at least 20000 μm in V3 and V4. Loss of R waves in V1 through V4 with ST-segment elevation indicates acute anterior MI.
- Loss of R wave in leads V1 through V3 with the ST-segment isoelectric and the T-wave inverted may be interpreted as anteroseptal MI age indeterminate (i.e., infarction in the recent or distant past). Features are given of old anterior MI and lateral infarction in this refinement.

Sometimes, R-wave progression in leads V2 through V4 are very poor, may be caused by the following reasons: improper lead placement, late transition, anteroseptal or anteroapical MI, LVH Severe chronic obstructive pulmonary disease, particularly emphysema may cause QS complexes in leads V1 through V4, which may mimic MI; a repeat ECG with recording electrodes placed one intercostal space below the routine locations should

cause R waves to be observed in leads V2 through V4, Hypertrophic cardiomyopathy, LBBB [Khan 2008].

11.5.2.5 Fifth Refinement : P-wave

This refinement level introduces a criterion to assess the P-wave for abnormalities, including the atrial hypertrophy in the ECG signal [Khan 2008]. A new variable *Disease_step6* is introduced in this refinement to introduce a set of diseases related to the P-wave. Some new variables are also introduced to assess the P-wave from 12-leads ECG signals, which are represented by $inv2-inv4$. The first two invariants introduce new variables in form of total functions mapping from leads (LEADS) to $\mathbb{N}$. These functions return height and broadness of the P-waves. The next invariant ($inv4$) represents total function mapping leads (LEADS) to *BOOL*. It returns diphasic state in a boolen type. A set of invariants ($inv5 - inv7$) are representing the confirmation of an abnormal state of the heart (*KO*). These invariants state that if the sinus rhythm is *Yes* and a new disease is found, then the heart will be in an abnormal state. The invariant ($inv5$) is checking for existence of multiple diseases during the P-wave diagnosis. Five new events *P_Wave_assessment_Peaked_Broad_No*, *P_Wave_assessment_Peaked_Yes*, *P_Wave_assessment_Peaked_Yes_Check_RAE*, *P_Wave_assessment_Broad_Yes* and *P_Wave_asses-sment_Broad_Yes_Check_LAE* are introduced to assess the P-wave.

$$
\begin{array}{l}
inv1 : Disease\_step6 \in Disease\_Codes\_Step6 \\
inv2 : P\_Wave\_Peak \in LEADS \rightarrow \mathbb{N} \\
inv3 : P\_Wave\_Broad \in LEADS \rightarrow \mathbb{N} \\
inv4 : Diphasic \in LEADS \rightarrow BOOL \\
inv5 : Sinus = Yes \wedge Disease\_step6 \in \\
\qquad \{RVH, RV\_strain, pulmonary\_embolism, \\
\qquad RAE, mitral\_stenosis, mitral\_regurgitation, LV\_failure, \\
\qquad LAE, dilated\_cardiomyopathy, LVH\_cause\} \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv6 : Sinus = Yes \wedge Disease\_step6 = LAE \Rightarrow Heart\_State = KO \\
inv7 : Sinus = Yes \wedge Disease\_step6 = RAE \Rightarrow Heart\_State = KO
\end{array}
$$

The textual representation of formal notation of the P-wave assessment is given in [Khan 2008]. We have formalised all textual guidelines.

11.5.2.6 Sixth Refinement : Assess for left and right ventricular hypertrophy

The Left Ventricular Hypertrophy (LVH) and Right Ventricular Hypertrophy (RVH) are assessed by this refinement. The criteria for LVH and RVH are not applicable if the bundle branch block is present [Khan 2008]. Thus, it is essential to exclude the LBBB and RBBB early in the interpretive sequences as delineated previously in refinement 2 and refinement 3. This refinement introduces two new variables *S_Depth* and *R_S_Ratio* in form of total functions mapping leads (LEADS) to $\mathbb{N}$. These functions are used to calculate the S-wave depth and ratio of R-wave and S-wave from the 12-leads ECG signal.

Invariants ($inv3-inv4$) are used to verify an abnormal state (*KO*) of the heart in case of detecting any disease. Two new events (*LVH_Assessment* and *RVH_Assessment*) are introduced to assess the LVH and RVH from the 12-leads ECG. Detailed textual representation of assessment of the LVH and RVH is given in [Khan 2008].

$$
\begin{array}{l}
inv1 : S\_Depth \in LEADS \rightarrow \mathbb{N} \\
inv2 : R\_S\_Ratio \in LEADS \rightarrow \mathbb{N} \\
inv3 : Sinus = Yes \wedge Disease\_step6 = RVH \Rightarrow Heart\_State = KO \\
inv4 : Sinus = Yes \wedge Disease\_step6 = LVH\_cause \Rightarrow Heart\_State = KO
\end{array}
$$

11.5.2.7 Seventh Refinement : Assess T-wave

This refinement is used to assess the pattern of T-wave changes in the 12-leads ECG signals. The T-wave changes are usually nonspecific [Khan 2008]. The T-wave inversion associated with the ST-segment depression or elevation indicates myocardial ischemia. A new variable *T_Normal_Status* represents as a boolean state like *TRUE* is for normal state, and *FALSE* is for abnormal state. A variable *Disease_step8* is introduced in this refinement to assess a set of diseases related to T-wave from the ECG signals. Invariants ($inv3 - inv8$) represent variables in form of total functions mapping leads (LEADS) to possible other attributes (*T_State*,*T_State_B* , *BOOL*, $\mathbb{N}$ and *T_State_l_d*).

The function *T_Wave_State* represents the T-wave states like peaked or flat, or inverted. Similarly, the function *T_Wave_State_B* also represents the T-wave states like upright or inverted, or variable using second method of diagnosis of the T-wave. The function *Abnormal_Shaped_ST* and *Asy_T_Inversion_strain* returns boolean state of the abnormal ST-shape and asymmetric T-wave inversion strain pattern, respectively. The Function *T_inversion* calculates deep the T-wave inversion and the last function *T_inversion_l_d* represents the localized and diffuse T-inversion.

From $inv9$ to $inv15$ represent an abnormal state of the heart due to finding some diseases. All these invariants are similar to the previous level of refinements. This refinement is very complex, and we have formalised two alternate diagnosis for the ECG signal. We have introduced many events to assess the T-wave from the ECG signals and to predict the various diseases related to the T-wave. Events are *T_Wave_Assessment_Peaked_V123456*, *T_Wave_Assessment_Peaked_V12*, *T_Wave_Assessment_Peaked_V12_MI*, *T_Wave_Assessment_Flat*, *T_Wave_Assessment_Inverted_Yes*, *T_Wave_Assessment_Inverted_No*, *T_Wave_Assess-ment_Inverted_Yes_PM*, *T_Wave_Assessment_B*, *T_Wave-_Assessment_B_DI*, *T_Inversion_Likely_Ischemia*, *T_Inver-sion_Diffuse_B*. All these events estimate a different kinds of properties from the T-wave signal for obtaining the correct heart disease. A long textual representation for analysing the T-wave is given in [Khan 2008].

$$
\begin{array}{l}
inv1 : T\_Normal\_Status \in BOOL \\
inv2 : Disease\_step8 \in Disease\_Codes\_Step8 \\
inv3 : T\_Wave\_State \in LEADS \rightarrow T\_State \\
inv4 : T\_Wave\_State\_B \in LEADS \rightarrow T\_State\_B \\
inv5 : Abnormal\_Shaped\_ST \in LEADS \rightarrow BOOL \\
inv6 : Asy\_T\_Inversion\_strain \in LEADS \rightarrow BOOL \\
inv7 : T\_inversion \in LEADS \rightarrow \mathbb{N} \\
inv8 : T\_inversion\_l\_d \in LEADS \rightarrow T\_State\_l\_d \\
inv9 : Sinus = Yes \wedge Disease\_step8 = Nonspecific \Rightarrow Heart\_State = KO \\
inv10 : Sinus = Yes \wedge Disease\_step8 = Nonspecific\_ST\_T\_changes \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv11 : Sinus = Yes \wedge Disease\_step8 = posterior\_MI \Rightarrow Heart\_State = KO \\
inv12 : Sinus = Yes \wedge Disease\_step8 \in \{Definite\_ischemia, \\
\qquad Probable\_ischemia, Digitalis\_effect\} \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv13 : Sinus = Yes \wedge Disease\_step8 = Definite\_ischemia \Rightarrow Heart\_State = KO \\
inv14 : Sinus = Yes \wedge Disease\_step8 = Probable\_ischemia \Rightarrow Heart\_State = KO \\
inv15 : Sinus = Yes \wedge Disease\_step8\_B \in \{Cardiomyopathy, other\_nonspecific\} \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO
\end{array}
$$

11.5.2.8 Eighth Refinement : Assess Electrical Axis

After finding all kinds of information about abnormal ECG, it is also essential to check the electrical axis (see Table 11.1) using two simple clues:

- If leads I and aVF are upright; the axis is normal.
- The axis is perpendicular to the lead with the most equiphasic or smallest QRS deflection. Left-axis deviation and the commonly associated left anterior fascicular block are visible in ECG signal.

This refinement is very essential refinement for the ECG interpretation because of the different angle of the ECG signal gives different output and angle based prediction can be changed [Khan 2008]. So, for accuracy of the ECG interpretation electrical axis must be included. New variables *minAngle*, *maxAngle*, *Axis_Devi* and *Disease_step9* have been defined here for assessment of the electrical axis. A new variable *QRS_Axis_State* is defined as a total function mapping from leads (LEADS) to *QRS_directions*. This function represents the QRS-axis direction of the leads. Two invariants ($inv6 - inv7$) represent the safety properties in assessment of the correct axis. These invariants are verifying an abnormal state of the heart (KO) using axis position.

Most equiphasic lead	Lead perpendicular	Axis
		Lead I and aVF positive = normal axis
III	aVR	Normal = +30 degrees
aVL	II	Normal = +60 degrees Lead I positive and aVF negative = Left axis
II	aVL (QRS positive)	Left = -30 degrees
aVR	III (QRS negative)	Left = -60 degrees
I	aVF (QRS negative)	Left = -90 degrees Lead I negative and aVF positive = right axis
aVR	III (QRS positive)	Right = +120 degrees
II	aVL (QRS negative)	Right = +150 degrees

Table 11.1: Electrical Axis

$$
\begin{array}{l}
inv1 : minAngle \in -90 \mathinner{..} 180 \\
inv2 : maxAngle \in -90 \mathinner{..} 180 \\
inv3 : Axis\_Devi \in Axis\_deviation \\
inv4 : Disease\_step9 \in Disease\_Codes\_Step9 \\
inv5 : QRS\_Axis\_State \in LEADS \rightarrow QRS\_directions \\
inv6 : Disease\_step9 \in \{LPFB, Dextrocardia, NV\_MSEC\} \wedge \\
\qquad maxAngle = 180 \wedge minAngle = 110 \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv7 : Disease\_step9 \in \{LAFB, MSCHD, Some\_Form\_VT, ED\_OC\} \\
\qquad \wedge maxAngle = -90 \wedge minAngle = -30 \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO
\end{array}
$$

In this refinement level, we introduce various events for assessing different kinds of features from 12 leads ECG signal corresponding to the angle. Following events are introduced in this refinement: *Axis_Assessment_QRS_upright_Yes_Age_less_40*, *Axis_Assessment_QRS_upright_Yes_Age_gre_40*, *Axis_Assessment_QRS_upright_No_QRS_positive*, *Axis_Assessment_QRS_upright_No_QRS_negative*, *Misc_Disease_Step9_LAD*, *Misc_Disease_Step9_RAD*, *R_Q_Assessment_R_Abno-rmal_V56_axis_deviation*.

11.5.2.9 Ninth Refinement: Assess for Miscellaneous Conditions

There are lots of heart diseases, and it is very difficult to predict everything. A lot of conditions make it more and more ambiguous. This refinement level keeps multiple miscellaneous conditions about the ECG interpretation [Khan 2008]. Following conditions are given for miscellaneous conditions as follows:

- Artificial pacemakers: If electronic pacing is confirmed, usually no other diagnosis can be made from the ECG.

- Prolonged QT syndrome: See normal QT parameters listed in Table 11.2. No complicated formula is required for assessment of the QT intervals.

Heart rate (bpm)	Male	Female
45−65	<470	<480
66−100	<410	<430
>100	<360	<370

Table 11.2: Clinically useful approximation of upper limit of QT interval (ms.)

A variable *MC_Step10_Test_Needed* is declared to represent miscellaneous condition tests as a boolean type *TRUE* or *FALSE*. Variable *Disease_step10* is introduced in this refinement to assess a set of diseases of miscellaneous conditions from the ECG signal. The next two invariants ($inv2 - inv3$) represent the abnormality of the heart state (*KO*) in case of discovery of new miscellaneous diseases. In this refinement, we introduce only two events (*Miscellaneous_Conditions_Step10* and *Misc_Disease_Step10_Dextrcardia_-Test*) to discover miscellaneous conditions from the ECG signal.

$$
\begin{array}{l}
inv1 : MC\_Step10\_Test\_Needed \in BOOL \\
inv2 : Disease\_step10 \in Misc_Disease\_Codes\_Step10 \\
inv3 : Sinus = Yes \wedge Disease\_step10 \in \{Incomplete\_RBBB, \\
\qquad Long\_QT, Hypokalemia, Digitalis\_toxicity, Hypothermia, \\
\qquad Electronic\_pacing, Pericarditis, , Hypercalcemia\} \\
\qquad Electrical\_alternans \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO \\
inv4 : Sinus = Yes \wedge Disease\_step9 = Dextrocardia \\
\qquad \Rightarrow \\
\qquad Heart\_State = KO
\end{array}
$$

11.5.2.10 Tenth Refinement: Assess Arrhythmias

This is the final refinement of the ECG interpretation of the system. In this refinement, we introduce different kinds of tachyarrhythmias and give the protocols for assessment as follows:

- Narrow complex tachycardia: Gives the differential diagnosis of narrow QRS complex tachycardia.

- Wide complex tachycardia: Gives the differential diagnosis of wide QRS complex tachycardia.

$inv1 : NW\_QRS\_Tachycardia\_RT\_State \in$
$\quad NW\_QRS\_Tachycardia\_RI$
$inv2 : Disease\_step11 \in Misc\_Disease\_Codes\_Step11$
$inv3 : Sinus = Yes \wedge Disease\_step11 \in$
$\quad \{Ventricular\_Premature\_Beats, Nodal\_Premature\_Beats,$
$\quad Bradyarrhythmias, Narrow\_QRS\_Tachycardias,$
$\quad Wide\_QRS\_Tachycardias, Atrial\_Premature\_Beats\}$
$\quad \Rightarrow Heart\_State = KO$

$inv4 : Sinus = Yes \wedge Distease\_step11\_NW\_QRST \in$
$\quad \{Sinus\_Tachycardia, Supraventricular\_Tachycardia,$
$\quad WPW\_Syndrome\_Orthodromic, Torsades\_de\_pointes,$
$\quad Atrial\_Tachycardia, AF\_Fixed\_AV\_Conduction, AVNRT,$
$\quad Ventricular\_Tachycardia, WPW\_Syndrome\_Antidromic,$
$\quad AF\_Variable\_AV\_Conduction\_BBB\_WPW\_Synd\_Anti,$
$\quad AF\_BBB\_WPW\_Synd\_Antidromic\}$
$\quad \Rightarrow Heart\_State = KO$

$inv5 : Sinus = Yes \wedge Distease\_step11\_NW\_QRST \in$
$\quad \{AF\_Variable\_AV\_Conduction, AVNRT,$
$\quad AT\_Paroxysmal\_NParoxysmal, AT\_Variable\_AV\_Block,$
$\quad AF\_Fixed\_AV\_Conduction, WPW\_Syndrome\_OCMT,$
$\quad Sinus\_Tachycardia, Multifocal\_Atrial\_Tachycardia,$
$\quad Atrail\_Fibrillation\}$
$\quad \Rightarrow Heart\_State = KO$

$inv6 : NW\_QRS\_Tachycardia\_RT\_State = Regular \wedge$
$\quad Distease\_step11\_NW\_QRST \in \{Sinus\_Tachycardia,$
$\quad WPW\_Syndrome\_OCMT, AF\_Fixed\_AV\_Conduction,$
$\quad AVNRT, AT\_Paroxysmal\_NParoxysmal\}$
$\quad \Rightarrow Heart\_State = KO$

$inv7 : NW\_QRS\_Tachycardia\_RT\_State = Irregular \wedge$
$\quad Distease\_step11\_NW\_QRST \in \{Atrail\_Fibrillation,$
$\quad AT\_Variable\_AV\_Block, AF\_Variable\_AV\_Conduction,$
$\quad Multifocal\_Atrial\_Tachycardia\}$
$\quad \Rightarrow Heart\_State = KO$

$inv8 : NW\_QRS\_Tachycardia\_RT\_State = Regular \wedge$
$\quad Distease\_step11\_NW\_QRST \in \{Ventricular\_Tachycardia,$
$\quad Sinus\_Tachycardia, AF\_Fixed\_AV\_Conduction,$
$\quad Supraventricular\_Tachycardia, Atrial\_Tachycardia,$
$\quad AVNRT, WPW\_Syndrome\_Antidromic,$
$\quad WPW\_Syndrome\_Orthodromic\}$
$\quad \Rightarrow Heart\_State = KO$

$inv9 : NW\_QRS\_Tachycardia\_RT\_State = Irregular \wedge$
$\quad Distease\_step11\_NW\_QRST \in$
$\quad \{AF\_Variable\_AV\_Conduction\_BBB\_WPW\_Synd\_Anti,$
$\quad Torsades\_de\_pointes, AF\_BBB\_WPW\_Synd\_Antidromic\}$
$\quad \Rightarrow Heart\_State = KO$

A new variable *NW_QRS_Tachycardia_RT_State* is defined to express the QRS tachycardia regular or irregular state using $inv1$. A variable *Disease_step11* is introduced in this refinement to assess arrhythmias from the ECG signals. All rest of the invariants ($inv3 - inv9$) represents an abnormal state (*KO*) of the heart after analysing the arrhythmia and related disease. All invariants have similar kinds of properties. We introduce five new events to assess tachyarrhythmias from the 12-leads ECG signals in case of abnormal rhythm. Five events are *Rhythm_test_FALSE_Step11*, *Step11_N_QRS_Tachycardia_Regular*, *Step11_N_QRS_Tachy-cardia_Irregular*, *Step11_W_QRS_Ta-chycardia_Regular* and *Step11_W_QRS_Tachycardia_Irregular*.

In this paper we have given only required safety properties in form invariants in all refinements. All these properties are derived from the original protocol to verify the correctness and consistency of the system. These properties are formulated through logic experts as well as cardiologist experts according to the original protocol. The main advantage of this technique is that if any property is not holding by the model, then it helps to find anomalies or to find missing parts of the model such as required conditions and parameters.

We have described here only summary information about each refinement in form of the basic description and required invariants of the ECG interpretation protocol using incremental refinement-based approach and omit detailed formalisation of the events and proof details. A technical report [Méry 2011h; Méry 2011j] contains the complete formal representation of the ECG interpretation protocol.

11.6 Statistical Analysis and Lesson Learned

11.6.1 Statistical Analysis

All the proof obligations for all ten refinements are generated and proved using the Rodin prover [RODIN 2004]. Table 11.3 shows statistics of the ECG interpretation protocol using refinement approach. In the table, the POs column represents the total number of proof obligations generated for each level. The interactive POs column represents the number of those proof obligations that have to be proved interactively. Those proof obligations that are not proved interactively are proved completely automatically by the prover. The complete development of the ECG interpretation protocol system results in 599 (100%) proof obligations, in which 343 (58%) are proved automatically by the Rodin tool. The remaining 256 (42%) proof obligations are proved interactively using Rodin tool. In seventh refinement, numbers of POs are higher than other refinements because significantly in this level; number of variables and events are higher than another level of refinements. All the proofs are discharged completely automatic as well as interactive for all refinement levels. All these proofs are involved either by the complexity of the formal expression that proved by *do case* or finiteness constraints on a set of leads. The main interactive steps involved instantiating for total function of the different features of the ECG interpretation in every level of refinement. In order to guarantee the correctness of the system, we have established various invariants in the stepwise refinement. All these invariants are derived from the original protocol to verify the correctness and consistency of the system under the guidance of the cardiologist expert. Most of the invariants are introduced for checking the

abnormality of the features of the ECG signal. Detection of an abnormal criteria, the heart shows surety of a particular disease or a set of diseases. A set of diseases are distinguished in next level of refinements.

Model	Total number of POs	Automatic Proof	Interactive Proof
Abstract Model	41	33(80%)	8(20%)
First Refinement	61	54(88%)	7(12%)
Second Refinement	41	38(92%)	3(8%)
Third Refinement	51	36(70%)	15(30%)
Fourth Refinement	60	35(58%)	25(42%)
Fifth Refinement	43	22(51%)	21(49%)
Sixth Refinement	38	14(36%)	24(64%)
Seventh Refinement	124	29(23%)	95(77%)
Eighth Refinement	52	30(57%)	22(43%)
Ninth Refinement	21	9(42%)	12(52%)
Tenth Refinement	67	43(64%)	24(36%)
Total	599	343(58%)	256(42%)

Table 11.3: Proof Statistics

11.6.2 Lesson learned

The task of modelling of the ECG interpretation protocol in the Event-B has required a significant effort. It is a typical knowledge engineering task, where the knowledge is the original document, is transformed into the Event-B formal notation, which provides a significant hierarchical structure for analysing the ECG interpretation protocol and to diagnose different kinds of heart diseases. As the result, the Event-B ECG interpretation protocol specification is much more lengthy than the original text: the original ECG interpretation protocol. The complete formal specification of the ECG interpretation protocol in the Event-B is more than 200 pages.

We consider that logic-based modeling approach is very difficult to model a complex medical protocol. This approach has required a good understanding of logic as well as knowledge of the medical protocol. We have spent a lot of time with medical experts to understand the structure of the medical protocols for formalising purpose. For modelling the ECG protocol, we have consulted with cardiologist and medical experts. The formal model of ECG protocol is based on original protocol and checked by medical experts [Méry 2011h; Méry 2011j].

We cannot strictly say that the formal representation of the ECG interpretation protocol in the Event-B modeling language has contributed to the improvement of the original protocol. Most important contribution is refinements-based formal development of the ECG interpretation protocol and to generate a new optimal way of the ECG interpretation protocol for diagnosing the ECG signal. The developed formal model is proved and verifying according to the given protocol properties as discussed in the formal development. Fur-

thermore, the Event-B formalisation has served to disambiguate unclarities in the original document that resulted from the modelling stage: a number of ambiguity and repetition diagnosis problems with original document are uncovered and resolved by refining the formal specification of the ECG interpretation protocol in the Event-B. The formal model can help to restructure the original document of guidelines and protocols.

The verification attempts have served to clarify any remaining problems in the original ECG interpretation protocol document. More importantly, we have shown that it is possible in practice to systematically analyse whether a protocol formalised in the Event-B complies with certain medically relevant properties. Various properties of the ECG interpretation protocol have been the object of formal verification using the Event-B system, with different type of results. Mostly, the given properties of the ECG interpretation protocol have been confirmed by the formal representation of the ECG interpretation protocol. However, in other cases, verification is not simple and lots of ambiguous informations, i.e. it is not possible to complete the proof or further development of the model due to ambiguity. We have introduced some additional assumptions with the help of cardiologist experts for describing the conditions needed to make the property true and added more conditions to remove the ambiguity. These assumptions are missing piece of information in the medical protocol, which helps to improve the medical protocol. We have applied a pragmatic approach to collect lots of informations through literature survey and medical experts advises for finding the exact facts to introduce new assumptions and conditions for discharging all generated proof obligations.

For example, pieces of informations missing from the original ECG interpretation protocol like it is not given that how many leads should hold particular property during diagnosis. As per our solution, we have applied test for particular properties in all leads. This results in a characterization of the circumstances under which the property holds. The obtained characterization is analyzed by the medical experts under all the possible conditions, and it can be used either to redefine the property or to improve the original ECG interpretation protocol text by documenting the cases under which the property does (or does not) hold.

More importantly, numerous anomalies became apparent during the Event-B modelling of the ECG interpretation protocol. Here, we have used term anomaly to refer to any issues that are not able to represent satisfactory of the original ECG interpretation protocol. Some set of anomalies, which have found during the development of the system are described below. We have grouped all anomalies in three well known general categories: ambiguity, inconsistency and incompleteness.

11.6.2.1 Ambiguous

Ambiguous is a well-known anomaly in the area of formal representation, and it is very hard to interpret. For instance, a problem we encountered while modelling the ECG interpretation protocol is determining whether the terms “ST-depression” and “ST-elevation” had the same meaning or not. These are terms that are used in the ECG interpretation original protocol, but not defined elsewhere. Similarly, what is the difference between “ischemia”, “Definite ischemia”, “probable ischemia” and “likely ischemia”.

In the ECG interpretation, there are 12 leads ECG signals, which are used for interpretation, but a lot of places in the original document not clarify in which lead the particular property should hold. Such kinds of information are very ambiguous and give lots of confusions to model the system.

11.6.2.2 Inconsistencies

Inconsistencies are other kinds of anomalies which are always given conflicting results or different decisions on same patient data. The problems derived from inconsistent elements are very serious and as such must be avoided during development. The ECG interpretation protocol presents several inconsistencies. For instance, we found an inconsistency in form of applicable conditions in the ECG protocol. It expresses that the conditions are applicable to both "male" and "female" under some certain circumstances. However, elsewhere in the protocol an action is advised that these conditions of the protocol are not applicable to "female".

11.6.2.3 Incompleteness

Either missing pieces of information or insufficient information in the original document are always related to the incompleteness anomaly. In either case, incompleteness hinders a correct interpretation of the guidelines and protocols. For example, the original protocol contains "normal variant" factors to be considered when assessing the T-wave. However, what "normal variant" exactly means is missing in the protocol. As an example of insufficient information for "normal variant", we provide the class of disease for further analysis the system.

11.7 Conclusion

Refinement is a key concept for developing the complex systems, since it starts with a very abstract model and incrementally adds new details to the set of requirements. We have outlined an incremental refinement-based approach for formalising medical protocols using the Rodin tool. The approach we have taken is not specific to the Event-B. We believe a similar approach could be taken using others state-based notations such as ASM, TLA$^+$ and Z etcetera. The Rodin proof tool is used to generate the hundreds of proof obligations and to discharge those obligations automatically and interactively. Another key role of the tool is in helping us to discover appropriate gluing invariants to prove the refinements. In summary, some key lessons are that incremental development with small refinement steps; appropriate abstractions at each level and powerful tool support are all invaluable in such a kind of formal development.

In this chapter, we have shown the formal representation of medical protocol. The formal model of medical protocol is verified, and this verified model is not only feasible but also useful for improving the existing medical protocol. We have fully formalised a real-world medical protocols (ECG interpretation) in an incremental refinement-based formalisation process, and we have used proof tools to systematically analyse whether the

formalisation complies with certain medically relevant protocol properties [Méry 2011h; Méry 2011j]. The formal verification process has discovered a number of anomalies which all are discussed in the previous section. Throughout this process, we have obtained the following concrete results:

- A formal specification language like Event-B is used for modeling a complex system, is used to model the medical practice protocols. The Event-B is a general modeling language tool. The Event-B is used to present a formal specification for a real-life medical protocols; ECG interpretation.

- The ECG interpretation protocol is formalised in the Event-B modeling language. The medical protocol ECG interpretation is used in our study has been developed in incremental way and finally transformed into a concrete formal representation. Each proved refinement level of the formal model of the protocol represents feasibility and correctness.

- In our formal verification process of the ECG interpretation, we have obtained a list of anomalies.

- Verification proofs for the ECG interpretation protocol, and properties have proved using the Rodin proof tool. Generated proof obligations and proofs show that formal verification of the ECG interpretation protocols is feasible.

- Original protocol of the ECG is also based on some hierarchy, but in that hierarchy, some diagnosis is repeating in multiple branches (see in [Khan 2008]). We have also discovered an optimized hierarchical structure for the ECG interpretation efficiently using incremental refinement approach, which can help to diagnose more efficiently then old techniques, and this obtained hierarchical structure is verified through medical experts.

The ECG interpretation protocol [Méry 2011h; Méry 2011j] is very complex, and it interprets various kinds of heart diseases. Improving quality of medical protocol using the formal verification tools like highly mathematical based modeling languages; Event-B, is the main contribution of our work. We have also discovered a hierarchical structure for the ECG interpretation efficiently that helps to discover a set of conditions that can be very helpful to diagnose particular disease an early stage of the diagnosis without using multiple diagnosis. Our hierarchical tree structure provides more concrete solutions for the ECG interpretation protocol and helps to improve the original ECG interpretation protocol. Our objective behind this work is that if any medical protocol is developed under particular circumstances to handle a set of specific properties according to the medical experts, formal verification can also meet whether the protocol actually complies with them. This has been the first attempt ever in verifying medical protocols with mathematical rigour with the generalized formal modeling tool Event-B. The main objective of this approach to test correctness and consistency of the medical protocol using refinement based incremental development. This approach is not only for diagnosis purpose, but it may be applicable to

covering a large group of other categories (i.e treatment, management, prevention, counseling, evaluation etc.)[3] related to the medical protocols.

[3] http://www.guideline.gov/

CHAPTER 12

Conclusion and Outlook

"The best way to predict the future is to invent it."
(Alan Kay)

Highly critical systems, such as medical, avionic, and automotive systems require high integrity, software reliability, and proof based development for complying with certification standards [FDA ; IEEE-SA ; CC ; ISO], which evaluate the systems before their usage. In this framework, adaptation of the formal method has become the state-of the-art tech to meet the high demands on safety and reliability by certification bodies [NITRD 2009]. However, adaptation of formal methods significantly complicates the development process of a system due to complexity of the modeling as well as a system itself. Refinement based modeling techniques reduce verification effort significantly by designing the whole system using the stepwise development process. The complete system is verified with the help of theorem prover, model checker and animation tools. Moreover, critical systems can be analysed already at the early stages of their development, which allow to explore conceptual errors, ambiguities, requirement correctness and design flaws before implementation of the actual system, and this approach helps to correct errors more easily and with less cost

12.1 Contributions

This thesis presents a new development life-cycle methodology, which is an extension of the waterfall model for developing the critical systems, where each phase has used different kinds of techniques based on formal methods. In current system development process, formal methods are used only at the early stage of the system development for verifying the requirements. We have proposed new development methodology, which supports formal methods at every stage of the system development process. We have not only used existing development life-cycle steps, but also introduced some new steps in the life-cycle methodology. The proposed new recursive approach is based on refinement techniques to build the whole system from requirement analysis to code generation. New introduced techniques and tools based on formal methods support refinement based formal development, essential verification, and validation steps and automatic code generation in the process of critical system development. The proposed techniques and tools are development methodology, a framework for real-time animator [Méry 2010b], refinement chart [Méry 2011e],

formal logic based heart model for closed-loop modeling [Méry 2011f; Méry 2011i] and automatic code generation tools [Méry 2010a; EB2ALL 2011; Méry 2011c; Méry 2011b].

In Chapter 1, we have given a list of objectives, which all are covered in this thesis through giving a new development life-cycle methodology and a set of associated techniques and tools for developing the critical systems. Assessment of the development methodology and a set of techniques and tools are given through well known case studies related to the medical and automotive domains. This work has established a unified theory for the critical system development, which is our first main objective. A set of techniques and tools related to the new development life-cycle methodology confirm the proposed unified theory, which is the second objective. Our third objective is related to environment modeling for developing the closed-loop systems. We have developed a heart model, which has been used for building a closed-loop model of a cardiac pacemaker. Refinement based modeling approach used in our methodology, helps to design the error-free system, satisfying the fourth objective. The next three objectives are focused on modeling of different components, subsystems, and finally, integration to the final system. In this context, we have proposed the refinement chart, which supports components based modeling as well as helps for integrating into the final system. Another objective, evidence-based certification, also met through our real-time animation technique. The real-time animator can also be used to rectify some conceptual level problems through animation, which are hidden using traditional techniques. The final objective is related to the safety assurance certification. When a system is developed using our proposed approach then it could be certifiable, because certification standards have some similar requirements for validating the system processes. We have given a rigorous approach for the system development rather than the traditional development of critical systems. In traditional development, formal methods are used to provide safety assurances and to meet the requirements of the standard of the certification bodies. Our new approach based completely on formal techniques, develops the whole system rigorously from requirement analysis to code implementation, satisfying all requirements of the standard certification bodies. In addition, the methodology provides a safety assessment approach to analyse the whole development life cycle of the critical system, which meets requirements of the certification standard bodies.

Complexity of the critical system makes, it is hard to understand and to verify. Several approaches exist for the verification of critical systems, including model checking, theorem proving, or simulation based validation. There exists a vast variety of problems related to the critical systems and several solutions for each problem. Rigorous reasoning about the system behavior is required to ensure that a desired behavior is achieved. Event-B is a modeling language, which describes a system abstractly, and introduces system details through refinement steps to obtain the final concrete system. The Rodin [RODIN 2004; Abrial 2010] tools provide significant automated proof support for generating the proof obligations and discharging them. Generated proof obligations help to understand the complexity of the problem and to ensure the correctness of the system. In this thesis, we have made the following contributions towards an integration of refinement based development using Event-B with formal specification, verification and code implementation for the critical systems:

A major step forward is the new life-cycle methodology, exploits the mathematical

base to carry out a complete rigorous proof based system development using formal techniques in every step from requirement analysis to automatic code generation. This life-cycle methodology is used for developing the critical systems for obtaining the certificate standards, such as IEC-62304 [IEC62304 2006] and the Common Criteria [CC ; Farn 2004; Mead 2003]. This development methodology combines the refinement approach with verification tool, model checker tool, real-time animator, and finally generates the source code using the automatic tools. System development process is concurrently assessed by safety assessment approach [Leveson 1991] to comply with certification standards. Applying these new approaches for highly critical systems have many benefits, i.e. the exposure of errors, which might not have been detected without formal methods. The guidance of NITRD [NITRD 2009] allows adoption of formal methods into an established set of processes for development and verification of a high confidence medical device to be an evolutionary refinement rather than an abrupt change of methodology.

Thesis work also advances the development of new techniques and tools for supporting the new life-cycle methodology, and is explained in subsequent phases:

Real time animator is used to validate the formal model with real-time data set at an early stage of system development without generating the source code [Méry 2010b], and to bridge the gap between software engineers and stakeholders to build quality system and discover all ambiguous informations from the requirements. The combined approach of formal verification and real-time animation allows the systematic development of a clear, concise, precise and unambiguous specification of a software system and enables software engineers to animate the formal specification at an early stage of the development. Moreover, there are scientific and legal applications as well, where the formal model based animation can be used to simulate (or emulate) certain scenarios to glean more information or better understandings of the system to improve the final given system.

Another significant contribution towards improving techniques and tools section is the "refinement chart", which is used to present the whole system using layering approach in graphical block diagrams, where functional blocks are divided into multiple simpler blocks in a new refinement level, without changing the original behavior of the system. The refinement chart offers a clear view of assistance in "system" integration. This approach also gives a clear view about the system assembling based on operating modes and different kinds of features. This is an important issue not only for being able to derive system-level performance and correctness guarantees, but also for being able to assemble components in a cost-effective manner. The complexity of design is reduced by structuring systems using modes and by detailing this design using refinement.

Automatic code generation from a proved formal model to the target programming language is an essential step for system implementation, which is an equally important contribution. We have developed the main principles, rules, and implementation solutions for the translation tool, and also code verification techniques for generating target programming language (C, C++, Java and C#) code satisfying Event-B specifications [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b]. The syntax adopted is restrictive, but with many salient & essential characteristics for the most numeric applications, supports powerful static-analysis methods and generates fast and safe source code in the target programming languages. The benefits of developing and enhancing the translation tool [EB2ALL 2011;

Méry 2010a; Méry 2011c; Méry 2011b] presented stem primarily from their increased support for automated translation between the two components of a formal model and target programming language. The adaptations of the translation rules require more complete experiments, especially with large formal models for checking the impact on the execution time for some specific platforms. The gains rely then on the guarantees provided using a formal method and on the certification level which can be obtained by this way. As far as we know, only few formal methods support code generation, which is as time & space efficient as handwritten code.

Development of an environment for closed-loop modeling using formal techniques is our another remarkable contribution of this thesis. We have presented a methodology for modeling a mathematical heart model based on logico-mathematical theory. The most important goal is that this formal model helps to obtain a certification for the medical devices related to the heart system such as cardiac pacemaker and ICDs. It can be also used as a diagnostic tool to identify a critical state of the patient using a patient environment model. The heart model is based on electrocardiography analysis, which models the heart circulatory system at the cellular level. This has been one of the most challenging problems to validate and verify the correct behavior of the developed system model (a cardiac pacemaker or ICDs) under biological environment (i.e. heart). This approach for formalizing and reasoning about impulse propagation into the heart system through the conduction network. The heart model suggests that such an approach can yield a viable model that can be subjected to useful validation against medical device softwares at an early stage in the development process (i.e. cardiac pacemaker). The heart model is verified with the help of physiologist and cardiologist experts.

Assessment of proposed development life-cycle methodology and a set of associated techniques and tools are made through the development of the industrial-scale case studies, which cover two different domains related to medical, instrumentation and automotive. The rational is to select two different kinds of case studies to show the generalization of proposed approach, and hence validate its applicability to any other domains. These two well-knows case studies are the cardiac pacemaker and cruise control [Méry 2009; Méry 2011g; Méry 2010b]. We have applied development methodology and associated techniques and tools for system implementation. The combined approaches of the formal verification, and validation, refinement chart, real-time animator, and automatic code generation, cover enumerated claims like certifiable assurance and safety, error-free system development and system integration. Refinement chart specially covers component-based design frameworks and decomposition, integration of critical infrastructure and device integration. Our case study on cardiac pacemaker illustrates the potential value of a formal specification, and its subsequent animation can bring to the comprehension and clarification of the informal requirements. The case study has shown that requirement specifications could be used directly in real-time environment without modifications for automatic test result evaluation using our approach. We can see from our pacemaker and ACC case studies that all these claims help to design error-free system and different phases of the system have been shown by refinements in form of formal development as well as refinement charts. We have presented evidence that such an analysis is fruitful for both formal and non-formal group of people. The second observation from our experiments is that develop-

ment of multiple models helped us not only find errors in the requirements documents but also gave us an opportunity to better understand intricate requirements such as the control algorithm of a critical system. Moreover, we believe that the effort needed is commensurate with the benefits we derive from developing the multiple models.

In order to assess the overall utility of our approach, a selection of the results of the formalisation and verification steps have been presented to a group of pacemaker developers (French-Italian based pacemaker development company). The developers are satisfied by the result of pacemaker development using this methodology in sense of incremental development as well as integration of hardware and softwares. They really agreed on the refinement charts for showing operating mode relation and their mode transitions. Similarly, for another case study related to the ACC system, the results of the formalisation and verification steps have been developed under supervision of a group of researchers (General Motor, India Science Lab, Bangalore, India). Throughout our case study, we have shown formal specification and verification of the ACC system. The controller based ACC models must be validated to ensure that they meet requirements. Hence, validation must be carried out by both formal modeling and domain experts. Based on the experiment described above and our conclusions we are convinced of the usefulness on certain areas, and therefore, we are considering to use this methodology for designing the highly critical systems. The proposed framework and developed techniques and tools offer system development from formal verification to code generation, which offer to obtain that challenge of complying with FDA's QSR, ISO/IEC and IEEE standards quality system directives [Keatley 1999; IEEE-SA ; FDA ; ISO ; CC] and help to get certification for the highly complex critical systems.

This thesis also contributes in the area of formal representation of the medical protocol. The formal model of medical protocol is verified, and this verified model is not only feasible but also useful for improving the existing medical protocol. We have fully formalised a real-world medical protocol (ECG interpretation) in an incremental refinement-based formalisation process, and we have used proof tools to systematically analyse whether the formalisation complies with certain medically relevant protocol properties [Méry 2011h; Méry 2011j]. The formal verification process has discovered a number of anomalies. We have also discovered a hierarchical structure for the ECG interpretation efficiently that helps to discover a set of conditions that can be very helpful to diagnose particular disease at early stage of the diagnosis without using multiple diagnosis. Our hierarchical tree structure provides more concrete solutions for the ECG interpretation protocol and helps to improve the original ECG interpretation protocol. The main objective of this approach is to test correctness and consistency of the medical protocol. This approach is not only for diagnosis purpose, but it may be applicable to covering a large group of other categories (i.e treatment, management, prevention, counseling, evaluation etc.)[1] related to the medical protocols.

[1] http://www.guideline.gov/

12.2 Consequences and Future Challenges

This thesis provides a huge amount of works related to the development of life-cycle methodology and a set of associated techniques and tools for developing the critical systems and presents assessment through comprehensive treatment of the cardiac pacemaker and the adaptive cruise control case studies. In this section, we outline the possible extensions to our work presented in this thesis.

In Chapter 4, we have presented the development life-cycle methodology for the critical system development using formal methods. In this thesis, we have reported two challenging case studies related to the medical and automotive domains to show the effectiveness of the proposed approach, without applying the safety assessment techniques. Safety assessment approach is also main important part of this proposed development life-cycle. It is our future work to apply safety assessment approaches at each level of the development life-cycle methodology for developing any critical system.

In Chapter 5, we have presented an architecture of real-time animator and develop a prototype model for testing the effectiveness in the cardiac pacemaker case study. The proposed architecture is not complete yet due to certain limitations like acquisition devices, features extraction algorithms, etc. It is an essential, and expected to use same architecture in multi domain to validate any formal model specification in the early stage of development. Manual application of this architecture to apply real-time data set in the formal model is tedious, cumbersome, and may be error prone. Therefore, there is a need to develop an application programming interface (API) which can interface automatically with different components of the architecture.

In Chapter 6, we have given the idea of the refinement chart, that is useful to design the modal system according to the operating modes through refinement approach. As a result, we have used manual development of the refinement chart in our cardiac pacemaker and adaptive cruise control (ACC) case studies. As a future work, we plan to develop an integrated development environment (IDE) for designing a critical system using refinement chart and automatic formalisation of the critical system.

In Chapter 7, we have presented a tool EB2ALL [EB2ALL 2011; Méry 2011b], which generates source codes into C, C++, Java and C# programming languages automatically. This tool has been successfully implemented except for generating code verification step. We are working on our translation tool, and attempting to implement last step of our tool chain; generated code verification. This is the important and challenging task in the development of a tool. Because of preservation at the code level of the properties proved at the architectural level is guaranteed only if the underlying platform is correct, and correctness of the final system when filling in the stubs for internal actions into the automatic generated code. It is our ongoing project, so we are continuing to extend this tool to support all other remaining Event-B formal symbols, which provides freedom for a developer to formalise a critical system and automatic source code generation of a developed model into any target language. In future, we plan to extend this translation tool for other kinds of the target programming languages such as *PLC, ADA, Boogie* so that this translation tool [EB2ALL 2011; Méry 2010a; Méry 2011c; Méry 2011b] can be used by all industrial areas, whereas formal verification, and validation are primary techniques to develop a

system.

In Chapter 8, we present an environment model for the heart system, which we use to integrate cardiac pacemaker formal specification [Méry 2011g] and the heart formal specification to model the closed-loop system for verifying the desired behavior of the cardiac pacemaker for certification purpose. As future work, we require to test such a kind of environment model to test for other medical systems related the heart system.

In Chapter 9, the most important goal is that the developed formal model helps to obtain a certification for the cardiac pacemaker, and our aim is to create the simulator of the cardiac pacemaker based on formal models, that can be used by the doctor to analyze the real-time heart signal and predict the operating modes. It can be also used as a diagnostic tool to diagnose the patient and help to take the better decision for implanting a pacemaker [Writing Committee 2008].

In Chapter 10, we have demonstrated a new modeling approach for modeling the hybrid system. We have shown that successful formal method application for developing the ACC system. We have completed the entire process of modeling and formalisation of the ACC system and properties, and discharged all generated proof obligations. The same approach of formal development of the highly complex hybrid control systems in multiple domains wherever hybrid controllers are used, would be a challenging progressive step of this work. We have identified some areas where our approach would help in creating better and reliable critical systems:

- Use of various kinds of PID control (P, PI, PD, PID)
- Verification of complex properties likes composition of individual controllers
- Parallel composition, series composition and combined parallel and series composition of controllers
- Identification and use of patterns in control design
- Automatic transformation from Event-B model to Simulink Model

In Chapter 11, we focus on some future challenges related to the application of formal techniques to improve medical protocol. We have shown that successful implementation for medical protocol improvement using formal methods, where, we have completed the entire process of modelling and formalisation of the ECG interpretation protocol and properties, and discharged all generated proof obligations. A hierarchical structure for ECG diagnosis is discovered, which can help for improving the ECG protocol. As a future work, It is possible to use the same technique to investigate other kinds of medical protocols (e.g EEG analysis) using our refinement-based modeling approach.

APPENDIX A

Certification Standards

"A hard beginning maketh a good ending."
(John Heywood)

A.1 What is Standards?

Standards are documented agreements containing technical specifications, which produce precise criteria, consistent rules, procedures to ensure reliability, software processes, methods, products, services and use of products are fit for their purpose in this world. Standards include a set of issues corresponding to the product functionality and compatibility, facilitate interoperability, including designing, developing, enhancing, and maintaining. A set of protocols and guidelines, which are produced by the standards, are consistent and universally acceptable for product development. Standards allow to understand the quality of different products for competing with them and provides a way to verify the credibility of new products [ISO ; IEEE-SA]. A basic definition of standards is defined by ISO [ISO] as follows:

Standards are documented agreements containing technical specifications or other precise criteria to be used consistently as rules, guidelines, or definitions of characteristics, to ensure that materials, products, processes and services are fit for their purpose.

Different nations tend to have different views of what a standard is and what standardization is for. Standards are varied from nation to nation. For instance, UK and Europe standards define a product. Implementation dependencies should be reduced, and rigorous testing of products should satisfy the standards. The standard is a description of an artifact that is to be built precisely according to the provisions of the standard.

Since software plays an increasingly important role in software-based products related to medical, automotive and avionic systems. Because of the uncertainty of the reliability and compatibility of these software-based products, different kinds of national and international standards related to certification bodies (FDA's QSR and ISO's 13485, etc.) need effective means for ensuring that the developed software-based system is safe and reliable.

There is a wide variety of standards bodies. More than 300 software standards and 50 organizations are developing software standards [Fries C. 2011]. Standards come in many different flavours, for example, de-facto standards, local, national and international standards. Some of the standards are more specific related to the defense, financial, medical,

nuclear, and transportation etcetera. Some of the major software standards are given in Table A.1 [Fries C. 2011].

AIAA	American Institute of Aeronautics and Astronauts
ANS	American Nuclear Society
ANSI	American National Standards Institute
ASTM	American Society for Testing and Materials
BSI	British Standards Institution
CCITT	Telecommunication Standardization Bureau
CEN	European Committee for Standardization
CSA	Canadian Standards Association
CSE	Communications Security Establishment
DEF	British Defense Standards
DIN	Drug Information Association
DIN	Deutsches Institute für Normung
DoD	U.S. Department of Defense
ISO	International Organization for Standardization
IEC	International Electrotechnical Committee
IEEE	Institute of Electronic a d Electrical Engineers
CC	Common Criteria
FDA	The Food & Drug Administration

Table A.1: Standards Organizations

In the next sections, we describe here only international standards related to the information technology by ISO/IEC (the International Organization for Standardization/ International Electrotechnical Commission), IEEE (Institute of Electronic and Electrical Engineers), FDA (Food and Drug Administration) and CC (Common Criteria).

A.2 ISO/IEC Standards

IEC (International Electrotechnical Commission) is established in 1906 and ISO (International Organization for Standardization) is a non-governmental organization is established in 1947 [Duce 1997; ISO ; Fries C. 2011]. The ISO is a worldwide federation of national standards bodies from more than 140 countries, one from each country, which facilitate the international coordination and unification of the industrial standards [ISO]. The primary preoccupation of international standards is to eliminate *technical barriers to trade*; the view is that world-wide standards help rationalize the international trading process [Duce 1997; Huhn 2010].

There are number of standards addressing safety and security of a system related to the software development. For example, avionics RTCA-Do-178B [RTCA 1992] or the IEC 61508 [IEC61508 2008; Gall 2008] as the fundamental standard for functional safety of the E/E/EP systems [IEC61508 2008; Gall 2008]. The IEC 62304 [IEC62304 2006] standard is for software life-cycles of medical device development, which addresses to

achieve more specific goals through standard process activity. The process standard IEC 62304 [IEC62304 2006] is a collection of two other standards ISO 14791 and ISO 13485, where ISO 14791 standards are for quality, and ISO 13485 is for risk management. Here, we have presented a brief introduction about IEC 61508 and IEC 62304 standards, which may be achieved using our proposed methodologies(see Part-I).

A.2.1 IEC 61508 - Software Safety in E/E/EP Systems

Systems constitute of electrical and/or electronic elements, which can be used to perform safety functions in many application sectors. ISO/IEC 61508 [IEC61508 2008; Gall 2008] constitutes a generic approach for all safety life cycle activities of the electrical and/or electronic and/or programmable electronic (E/E/PE) systems to perform safety functions. It provides a generic development approach for achieving a rational and consistent technical policy for all kinds of electrical systems to the safety-related system. This standard provides some frameworks to consider safe and reliable for the safety-related systems that are developed in other technologies. It covers a wide variety of complexity, hazard and risk potentials related to the E/E/PE systems. Main objective of this standard is to define a life-cycle for safety-critical software considering best practices and recommendations from early phases of requirements and development to operation, maintenance and disposal. A complete detail description about *Software Architecture Design* related to the properties for systematic integrity, software design and development are given in a tabular form [IEC61508 2008; Gall 2008; Huhn 2010]. The main objective of the IEC 61508 is to provide software architecture design, including design activity of the system, which are defined as follows:

- Selection of techniques and verify the satisfiable level according to the safety requirements
- Partitioning of the system
- Software/hardware interaction
- Unambiguous representation of the architecture
- Treatment of safety integrity of data
- Specification of architecture integration tests

Besides generic quality goals, the IEC 61508 also covers process dependencies and concrete characteristics of the architecture related to the completeness and correctness according to the requirements, no design faults, simple modular and structure-able, satisfiable desired behaviour, verifiable and testable design, and fault tolerance against system failure due to common cause [IEC61508 2008; Huhn 2010].

A.2.2 IEC 62304 - Process Requirements for Medical Device Software

The IEC 62304 [IEC62304 2006] standard specifies a framework of the life cycle processes for medical devices, which helps to design a safe system. All necessary requirements for each life cycle process are provided by the IEC 62304. Life cycle process is divided into a subset of activities and is controlled by the risk management and quality management. The risk-management process is defined by the ISO 14971 standard and quality management is defined by the ISO 13485 standards.

The ISO 14971 and ISO 13485 standards [ISO] provide risk-based quality management that determines the required rigor of software quality assurance from the risk, which appears from a medical device in form of undesired behavior of the system. Software can be an important part of a medical device providing safety and effectiveness of the software-based a medical device requires to fulfill requirements and to use of software without any risk. When a software is contributing to a hazard, which is determined by hazard identification activity of the risk-management process. Hazards could be indirectly caused by software, which can be considered that software is a contributing factor. The use of software to control risk is made during the risk control activity and risk management process under consideration of the ISO 14971 and ISO 13485 standards, respectively. The software-development process consists of a number of activities related to the service or maintenance of a medical device system, including software updates. All these activities are also considered as an important task of the software-development process. The IEC 62304 mentions six sub-activities for the architectural design step, which are as follows:

- Realization of the requirements
- Interface design
- Specification of functional and non-functional software components
- Specification of the environment of software components
- Partitioning due to the risk mitigation strategy
- Verification of the architecture

The software safety classification ranges from A - no harm or injury - to C - death or severe injury is possible. The classification defines the principle level of rigor, and consequently, the efforts to be undertaken, is required for all software development and maintenance activities [Huhn 2010]. The IEC 62304 standards provide assurance for medical software system and guarantees that the software does not contribute to hazardous failure of the system due to its systematic safety-oriented process and implementation of the functional requirements are performed carefully for the required activities. The requirement analysis, architecture, design, implementation and integration are main phases of the development process for handling the complexity of a system. Each phase of the development process is controlled by the IEC 62304, which recommends activities to plan, track, control and communicate possible problems to prevent the risk of systematic errors. The complete process development with risk management for a medical device is described in [IEC62304 2006; Fries C. 2011].

A.3 IEEE Standards Association

The Institute of Electrical and Electronics Engineers (IEEE) standard [IEEE-SA] provides the safety assurance level for industries, including: power and energy, biomedical and health care, information technology, transportation, nanotechnology, telecommunication, information assurance, and many more. The IEEE standard is internationally recognized and technical experts from all over the world participate in the development of it [IEEE-SA ; IEEE Std. 1074-1997]. The IEEE standards documents are prepared within the IEEE societies. The IEEE standards are developed by both groups of experts related to the subject within the IEEE societies and outside from the IEEE societies. The IEEE societies built a group from the broad range of individuals and organizations from worldwide with different kinds of expert to assist in standards development and standards-related collaboration. This group helps for innovation and expansion of technologies on the criteria of international market demand related to the safety systems. The IEEE standard is approved by authority and considers the users recommendations before apply into the development process. All these standards are reviewed at least every five years to qualify the new amendments in the systems. IEEE societies provide standards for almost every area of engineering, which are enumerated as follows:

- Aerospace Electronics
- Antennas & Propagation
- Batteries
- Communications
- Computer Technology
- Consumer Electronics
- Electromagnetic Compatibility
- Green & Clean Technology
- Healthcare IT
- Industry Applications
- Instrumentation & Measurement
- National Electrical Safety Code
- Nuclear Power
- Power & Energy
- Power Electronics
- Smart Grid

- Software & Systems Engineering
- Transportation
- Wired & Wireless

We have given some basic details about related to the Software & Systems Engineering, which are defined as follows:

A.3.1 IEEE 1012-yyyy

The IEEE 1012-yyyy standard is particularly used for both critical and non-critical software related to the Software Verification and Validation Plans (SVVP). Critical software is *software in which a failure could have an impact on safety or could cause large, financial or social losses* [IEEE Std 1990]. The SVVP mainly used for first, verification of the software product according to the previously defined life-cycle phases, and second validation of the final product according to the existing software and system requirements [IEEE Std. 1012-1998].

A.3.2 IEEE 730-yyyy

The IEEE 730-yyyy standard [IEEE Std. 730-1998] is related to the Software Quality Assurance Plans (SQAP), which provides minimum acceptable requirements for the software. This standard helps specially for development and maintenance of the critical software. A subset of requirements of this standard is applicable to non-critical and already developed softwares.

A.3.3 IEEE 1074-yyyy

The IEEE 1074-yyyy standard is used for developing a process of a software project life cycle. This standard mainly controls the architecture of the process, which is particularly useful for organization that is responsible to complete development process of the software projects [IEEE Std. 1074-1997]. The development life cycle of the IEEE standards is depicted in Fig. A.1, which describes a process for developing the IEEE standards using six stages life cycle under the fixed time frame along with the effective and trusted process. A detailed description about each level of the development life cycle of the IEEE standards is available in [IEEE-SA].

A.4 FDA

The Food and Drug Administration (FDA) [Keatley 1999] is established by US Department of Health and Human Services (HHS) in 1930 for regulating the various kinds of product like food, cosmetics, medical devices etcetera. The FDA is now using standards in the regulatory review process to provide safety to the public before using any product. The FDA allows manufacturers to submit the declaration of conformity to satisfy premarket review requirements. The FDA provides some guidelines on the recognition use of and

Figure A.1: The Standards Development Lifecycle of IEEE

consensus standards. The FDA is interested in standards because they can help to serve as a common yardstick to assist with mutual recognition, based on the signed Mutual Recognition Agreement between the European Union and United States. More than ever before, standards will have the more prominent role for the review of medical devices. The FDA also recognizes ISO/IEC and IEEE standards [IEEE-SA ; IEEE Std 1990]. Basic goals of the FDA standard are:

- To promote health by reviewing research and approving new products
- To ensure foods and drugs are safe and properly labeled
- To work with other nations to "reduce the burden of regulation"
- To cooperate with scientific experts and consumers to effectively carry out these obligations

The FDA standard classifies the medical devices based on risk and use of medical devices. The FDA provides some standard guidelines for medical devices, and medical devices have required to meet these standards. Time to time lots of amendments have been done in the FDA standards [FDA ; Keatley 1999] according to the use of medical devices to provide safety.

The Center for Devices and Radiological Health (CDRH) [CDRH 2006] is the branch of the FDA, which is responsible for ensuring the safety of medical devices and eliminating unnecessary radiation from the medical products [Keatley 1999; FDA ; Jetley 2006]. It provides standards for medical products from the simple toothbrush to complex devices such as pacemakers. The CDRH [CDRH 2006] also checks the safety performance of non-medical devices, which emit certain types of electromagnetic radiation like cellular phones, screening equipment, microwave ovens etcetera. The CDRH has some standards, which are used to describe many aspects of a medical device related to premarket and post-market issues. Here, we briefly mention some basic concepts [Keatley 1999; FDA] involved in FDA regulation for medical devices:

- **Class I** devices are defined as non-life sustaining. These products are the least complicated and their failure poses little risk.
- **Class II** devices are more complicated and present more risk than Class I, though are also non-life sustaining. They are also subject to any specific performance standards.
- **Class III** devices sustain or support life, so that their failure is life threatening.

A.5 Common Criteria

The Common Criteria (CC) [CC] is an international standard that allows evaluation of security for IT products and technology. The CC is an international standard (ISO/IEC 15408) [ISO] for computer security certification. The CC is a collection of existing criteria: European (Information Technology Security Evaluation Criteria)(ITSEC)), US (Trusted Computer Security Evaluation Criteria (TCSEC)) and Canadian (Canadian Trusted Computer Product Evaluation Criteria (CTCPEC)) [CC 2009a; CC 2009b; CC 2009c]. The CC [CC] contributes for developing an international standard and provides a way to worldwide mutual recognition and evaluation results.

The Common Criteria enable an objective evaluation to validate that a particular product or system satisfies a defined set of security requirements. The CC provides a framework for computer users, vendors and testing organizations for fulfill their requirements and ensure that the process of specification, implementation and testing of the product has been conducted in a rigorous and standard manner. The CC has mainly three parts, which has been described in Table A.2 to show the interest of three different kinds of users (Consumers, Developers and Evaluators) [CC 2009a].

CC objectives [CC 2009a; CC 2009b; CC 2009c] are described as follows:

- To ensure that evaluations of Information Technology (IT) products and protection profiles are performed to high and consistent standards and are seen to contribute significantly to confidence in the security of those products and profiles
- To improve the availability of evaluated, security-enhanced IT products and protection profiles
- To eliminate the burden of duplicating evaluations of IT products and protection profiles
- To continuously improve the efficiency and cost-effectiveness of the evaluation and certification/validation process for IT products and protection profiles.

A.5.1 CC Evaluation Assurance Level (EAL)

The Common Criteria (CC) certification provided insurance coverage by measuring the level of security based the likelihood of threats and their impact. The Common Criteria defines two classes of security requirements: functional and assurance. The objectives of these two classes vary depending upon the security classification level. There are seven

	Consumers	Developers	Evaluators
Part 1: Introduction and General Model	For background information and reference purposes	For background information and reference for the development of requirements and formulating security specifications for TOEs	For background information and reference purposes. Guidance structure for PPs and STs
Part 2: Security Functional Requirements	For guidance and reference when formulating statements of requirements for security functions	For reference when interpreting statements of requirements and formulating functional specifications of TOEs	Mandatory statement of evaluation criteria when determining whether TOE effectively meets claimed security functions
Part 3: Security Assurance Requirements	For guidance when determining required levels of assurance	For reference when interpreting statements of assurance requirements and determining assurance approaches of TOEs	Mandatory statement of evaluation criteria when determining the assurance of TOEs and when evaluating PPs and STs

Table A.2: CC user groups (Consumers, Developers and Evaluators)

levels of assurance that is known as Evaluation Assurance Levels (EALs). The numerical rating of the EAL [CC 2009c] describes development and presentation of the product's evaluation. Each EAL corresponds to the Security Assurance Requirements (SARs), which cover the product development within the level of strictness. The assurance level from EAL1 to EAL7 represents an increasing order of evaluation assurance level. In the EALs, the first level being when the threat and impact are very low and the seventh is when the threat and impact are very strong, means the higher level provides more confidence and assurance safety. The last level of EAL involves verification of the developed software based on logical reasoning and theorem proving techniques. Higher level of EALs do not necessarily imply " better security", they only mean that the security claimed is extensively verified. All the Evaluation Assurance Levels (EALs) [Mead 2003] of safety are described as follows:

EAL1: Functionally Tested. It applies when you require confidence in a product's correct operation, but do not view threats to security as serious. An evaluation at this level should provide evidence that the target of evaluation functions in a manner consistent with its documentation, and that it provides useful protection against identified threats.

EAL2: Structurally Tested. It applies when developers or users require low to moderate independently assured security, but the complete development record is not readily available. This situation may arise when there is limited developer access or when there is an effort to secure legacy systems.

EAL3: Methodically Tested and Checked. It applies when developers or users require a moderate level of independently assured security and require a thorough investigation of

the target of evaluation and its development, without substantial re-engineering.

EAL4: Methodically Designed, Tested, and Reviewed. It applies when developers or users require moderate to high independently assured security in conventional commodity products and are prepared to incur additional security-specific engineering costs.

EAL5: Semi-Formally Designed and Tested. It applies when developers or users require high, independently assured security in a planned development and require a rigorous development approach that does not incur unreasonable costs from specialist security engineering techniques.

EAL6: Semi-Formally Verified Design and Tested. It applies when developing security targets of evaluation for application in high-risk situations where the value of the protected assets justifies the additional costs.

EAL7: Formally Verified Design and Tested. It applies to the development of security targets of evaluation for application in extremely high-risk situations, as well as when the high value of the assets justifies the higher costs.

Bibliography

[Ábrahám-Mumm 2001] Erika Ábrahám-Mumm, Martin Steffen and Ulrich Hannemann. *Verification of Hybrid Systems: Formalization and Proof Rules in PVS*. In International Conference on Engineering of Complex Computer Systems (ICECCS), pages 48–57, 2001. (Cited on page 218.)

[Abrial 1996a] J.-R. Abrial. The B book - Assigning Programs to Meanings. Cambridge University Press, 1996. (Cited on pages 4, 24, 41, 49, 50, 52, 53, 54, 55, 57, 87, 98, 117 and 216.)

[Abrial 1996b] J.-R. Abrial. *Extending B without changing it (for developing distributed systems)*. In H. Habrias, Editor, 1^{st} Conference on the B method, pages 169–190, November 1996. (Cited on page 51.)

[Abrial 1996c] Jean-Raymond Abrial, Egon Börger and Hans Langmaack, Editors. Formal methods for industrial applications, specifying and programming the steam boiler control, volume 1165 of *Lecture Notes in Computer Science*. Springer, 1996. (Cited on pages 64, 87 and 88.)

[Abrial 1998] Jean-Raymond Abrial and Louis Mussat. *Introducing Dynamic Constraints in B.* In B98, pages 83–128, 1998. (Cited on page 51.)

[Abrial 2003a] J.-R. Abrial. *$B^{\#}$: Toward a Synthesis Between Z and B*. In D. Bert and M. Walden, Editors, 3nd International Conference of B and Z Users - ZB 2003, Turku, Finland, Lectures Notes in Computer Science. Springer, June 2003. (Cited on pages 50 and 51.)

[Abrial 2003b] J.-R. Abrial, D. Cansell and D. Méry. *A Mechanically Proved and Incremental Development of IEEE 1394 Tree Identify Protocol.* Formal Aspects of Computing, vol. 14, no. 3, pages 215–227, 2003. (Cited on page 51.)

[Abrial 2010] J.-R. Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010. (Cited on pages 4, 6, 14, 24, 26, 45, 49, 52, 53, 57, 74, 83, 87, 96, 97, 98, 101, 102, 103, 117, 125, 128, 141, 155, 162, 167, 202, 216, 251, 253 and 282.)

[Acuña 2005] S.T. Acuña and N. Juristo. Software process modeling. International series in software engineering. Springer Science, 2005. (Cited on pages 4, 23, 35, 65 and 71.)

[Ada 1981] The programming language ada reference manual, proposed standard document, united states department of defense, volume 106 of *Lecture Notes in Computer Science*. Springer, 1981. (Cited on page 97.)

[Adam 1991] D.R. Adam. *Propagation of depolarization and repolarization processes in the myocardium-an anisotropic model*. Biomedical Engineering, IEEE Transactions on, vol. 38, no. 2, pages 133 –141, feb. 1991. (Cited on page 130.)

[Advani 2003] Aneel Advani, Mary Goldstein, Yuval Shahar and Mark A Musen. *Developing Quality Indicators and Auditing Protocols from Formal Guideline Models: Knowledge Representation and Transformations*. AMIA Annual Symposium proceedings AMIA Symposium AMIA Symposium, pages 11–15, 2003. (Cited on page 252.)

[Aizenbud-Reshef 2006] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin and Y. Shaham-Gafni. *Model traceability*. IBM Systems Journal, vol. 45, no. 3, pages 515 –526, 2006. (Cited on page 77.)

[Arnold 2005] Ken Arnold, James Gosling and David Holmes. The java programming language (fourth edition). Addison-Wesley Professional, 2005. (Cited on page 95.)

[Artigou 2007] J.Y. Artigou, Société française de cardiologie and J.J. Monsuez. Cardiologie et maladies vasculaires. Elsevier Masson, 2007. (Cited on pages 47, 129, 130, 132, 153 and 202.)

[Back 1979] R. Back. *On correct refinement of programs*. Journal of Computer and System Sciences, vol. 23, no. 1, pages 49–68, 1979. (Cited on pages 4, 24, 49, 50, 87, 102, 103 and 141.)

[Back 1998a] R. Back. *A Calculus of Refinements for Program Derivations*. Acta Informatica, vol. 25, pages 593–624, 1998. (Cited on pages 49 and 51.)

[Back 1998b] R. Back and J. von Wright. Refinement Calculus A Systematic Introduction. Graduate Texts in Computer Science. Springer, 1998. (Cited on page 50.)

[Badeau 2005] Frédéric Badeau and Arnaud Amelot. *Using B as a High Level Programming Language in an Industrial Project: Roissy VAL*. In Helen Treharne, Steve King, Martin Henson and Steve Schneider, Editors, ZB 2005: Formal Specification and Development in Z and B, volume 3455 of *Lecture Notes in Computer Science*, pages 15–25. Springer Berlin / Heidelberg, 2005. (Cited on page 64.)

[Bagaria 2010] Sankalp Bagaria. *An Algorithm for Designing Controllers*. vol. 90, pages 235–242, 2010. (Cited on page 217.)

[Baier 2008] Christel Baier and Joost-Pieter Katoen. Principles of Model Checking (Representation and Mind Series). The MIT Press, 2008. (Cited on page 163.)

[Balser 1999] Michael Balser, Wolfgang Reif, Gerhard Schellhorn and Kurt Stenzel. *KIV 3.0 for Provably Correct Systems*. In Dieter Hutter, Werner Stephan, Paolo Traverso and Markus Ullmann, Editors, Applied Formal Methods â FM-Trends 98, volume 1641 of *Lecture Notes in Computer Science*, pages 330–337. Springer Berlin / Heidelberg, 1999. (Cited on page 253.)

[Balzer 1982] Robert M. Balzer, Neil M. Goldman and David S. Wile. *Operational specification as the basis for rapid prototyping*. SIGSOFT Softw. Eng. Notes, vol. 7, pages 3–16, April 1982. (Cited on page 78.)

[Barnes 2003] John Barnes. High integrity software: The spark approach to safety and security. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. (Cited on page 97.)

[Barold 2004] S. Serge Barold, Roland X. Stroobandt and Alfons F. Sinnaeve. Cardiac pacemakers step by step. Futura Publishing, 2004. ISBN 1-4051-1647-1. (Cited on pages 131, 165, 254 and 255.)

[Basili 1975] Victor R. Basili and Albert J. Turner. *Iterative enhancement: A practical technique for software development*. IEEE Transactions on Software Engineering, vol. 4, pages 390–396, 1975. (Cited on page 63.)

[Bäumler 2006] Simon Bäumler, Michael Balser, Andriy Dunets, Wolfgang Reif and Jonathan Schmitt. *Verification of Medical Guidelines by Model Checking – A Case Study*. In Antti Valmari, Editor, Model Checking Software, volume 3925 of *Lecture Notes in Computer Science*, pages 219–233. Springer Berlin / Heidelberg, 2006. 10.1007/11691617_13. (Cited on page 253.)

[Bayes 2006] Batcharov V. N. Bayes de Luna A. and M. Malik. The morphology of the Electrocardiogram in The ESC Textbook of Cardiovascular Medicine, pages 1–36. Blackwell Publishing Ltd., 2006. (Cited on pages 47, 129, 130, 132, 153 and 202.)

[Behm 1999] P. Behm, P. Benoit, A. Faivre and J.-M.Meynadier. *METEOR : A successful application of B in a large project*. In Proceedings of FM'99: World Congress on Formal Methods, Lecture Notes in Computer Science, pages 369–387, 1999. (Cited on page 51.)

[Bell 1993] R. Bell and D. Reinert. *Risk and system integrity concepts for safety-related control systems*. Microprocess. Microsyst., vol. 17, pages 3–15, January 1993. (Cited on pages 4, 23, 35, 37, 38, 46, 65 and 71.)

[Bengtsson 1996] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson and Wang Yi. *UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems*, 1996. (Cited on page 130.)

[Berenfeld 1996] Omer Berenfeld and Shimon Abboud. *Simulation of cardiac activity and the ECG using a heart model with a reaction-diffusion action potential*. Medical Engineering & Physics, vol. 18, no. 8, pages 615 – 625, 1996. (Cited on page 130.)

[Bernardo 1998] Marco Bernardo, Rance Cleaveland, Steve Sims and W. Stewart. *TwoTowers: A Tool Integrating Functional and Performance Analysis of Concurrent Systems*. In Proceedings of the FIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XI) and Protocol Specification, Testing and Verification (PSTV

XVIII), FORTE XI / PSTV XVIII '98, pages 457–467, Deventer, The Netherlands, The Netherlands, 1998. Kluwer, B.V. (Cited on page 97.)

[Bernardo 2004] M. Bernardo and E. Bonta. *Generating well-synchronized multithreaded programs from software architecture descriptions*. In Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on, pages 167 – 176, june 2004. (Cited on page 124.)

[Bernardo 2005] Marco Bernardo and Edoardo Bontà. *Preserving Architectural Properties in Multithreaded Code Generation*. In Jean-Marie Jacquet and Gian Pietro Picco, Editors, Coordination Models and Languages, volume 3454 of *Lecture Notes in Computer Science*, pages 188–203. Springer Berlin / Heidelberg, 2005. (Cited on page 124.)

[Berry 2000] Gérard Berry, Amar Bouali, Xavier Fornari, Emmanuel Ledinot, Eric Nassor and Robert de Simone. *ESTEREL: a formal method applied to avionic software development*. Science of Computer Programming, vol. 36, no. 1, pages 5 – 25, 2000. (Cited on page 32.)

[Berry 2007] Gérard Berry. *Synchronous Design and Verification of Critical Embedded Systems Using SCADE and Esterel*. In Formal Methods for Industrial Critical Systems (FMICS), page 2, 2007. (Cited on page 69.)

[Bert 2003] Didier Bert, Sylvain Boulmé, Marie-Laure Potet, Antoine Requet and Laurent Voisin. *Adaptable Translator of B Specifications to Embedded C Programs*. In International Symposium of Formal Methods Europe, volume 2805 of *Lecture Notes in Computer Science*, pages 94–113, 2003. (Cited on pages 98 and 126.)

[Bertolino 1997] Antonia Bertolino and Lorenzo Strigini. *Acceptance criteria for critical software based on testability estimates and test results*. In SAFECOMP 96, Proc. of the 15th International Conference on Computer Safety, Reliability and Security, pages 83–94. Springer, 1997. (Cited on page 70.)

[Berzins 1993] V. Berzins, Luqi and A. Yehudai. *Using transformations in specification-based prototyping*. Software Engineering, IEEE Transactions on, vol. 19, no. 5, pages 436 –452, May 1993. (Cited on page 78.)

[Beyer 2007] Dirk Beyer, Thomas Henzinger, Ranjit Jhala and Rupak Majumdar. *The software model checker BLAST, Applications to software engineering*. International Journal on Software Tools for Technology Transfer (STTT), vol. 9, pages 505–525, 2007. (Cited on pages 96 and 124.)

[Bjørner 1978] Dines Bjørner and Cliff B. Jones, Editors. The Vienna Development Method: The Meta-Language, London, UK, 1978. Springer-Verlag. (Cited on pages 40, 78, 97 and 163.)

[Bjørner 2007] Dines Bjørner and Martin C. Henson, Editors. Logics of specification languages. EATCS Textbook in Computer Science. Springer, 2007. (Cited on pages 305, 307, 310, 318 and 321.)

[Blaauboer 2007] Floris Blaauboer, Klaas Sikkel and Mehmet N. Aydin. *Deciding to adopt requirements traceability in practice*. In Proceedings of the 19th international conference on Advanced information systems engineering, CAiSE'07, pages 294–308, Berlin, Heidelberg, 2007. Springer-Verlag. (Cited on page 77.)

[Blanc 2000] Lionel Blanc and Sylvan Dissoubray. *Esterel methodology for complex system design*. Microelectronic Engineering, vol. 54, no. 1-2, pages 163 – 170, 2000. (Cited on page 32.)

[Blanc 2007] Nicolas Blanc, Alex Groce and Daniel Kroening. *Verifying C++ with STL containers via predicate abstraction*. In Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, ASE '07, pages 521–524, New York, NY, USA, 2007. ACM. (Cited on page 103.)

[Blanchard 2006] B.S. Blanchard and W.J. Fabrycky. Systems engineering and analysis. Prentice-Hall international series in industrial and systems engineering. Pearson Prentice Hall, 2006. (Cited on page 35.)

[Bloomfield 1986] R E Bloomfield and Froo P K D. *The application of formal methods to the assessment of high integrity software*. IEEE Trans. Softw. Eng., vol. 12, pages 988–993, September 1986. (Cited on page 78.)

[Boehm 1976] B. W. Boehm. *Software Engineering*. IEEE Trans. Comput., vol. 25, pages 1226–1241, December 1976. (Cited on page 63.)

[Boehm 1981] Barry W. Boehm. Software Engineering Economics. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1981. (Cited on page 76.)

[Bonta 2009] E. Bonta and M. Bernardo. *PADL2Java: A Java code generator for process algebraic architectural descriptions*. In Software Architecture, 2009 European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on, pages 161 –170, 2009. (Cited on page 97.)

[Börger 2003] E. Börger and R. Stärk. Abstract state machines: A method for high-level system design and analysis. Springer, 2003. (Cited on page 52.)

[Bottrighi 2010] Alessio Bottrighi, Laura Giordano, Gianpaolo Molino, Stefania Montani, Paolo Terenziani and Mauro Torchio. *Adopting model checking techniques for clinical guidelines verification*. Artif. Intell. Med., vol. 48, pages 1–19, January 2010. (Cited on pages 252 and 253.)

[Bowen 1993] J. Bowen and V. Stavridou. *Safety-critical systems, formal methods and standards*. Software Engineering Journal, vol. 8, no. 4, pages 189–209, Jul 1993. (Cited on pages 31, 32, 40, 43, 64, 128 and 216.)

[Bozzano 2010] Marco Bozzano and Adolfo Villafiorita. Design and safety assessment of critical systems. Auerbach Publications, Boston, MA, USA, 1st edition, 2010. (Cited on pages 31, 40 and 43.)

[Britt 1994] J.J. Britt. *Case study: Applying formal methods to the Traffic Alert and Collision Avoidance System (TCAS) II*. In Computer Assurance, 1994. COMPASS '94 Safety, Reliability, Fault Tolerance, Concurrency and Real Time, Security. Proceedings of the Ninth Annual Conference on, pages 39 –51, jun-1 jul 1994. (Cited on page 41.)

[Bubenko 1995] Jr. Bubenko J.A. *Challenges in requirements engineering*. In Requirements Engineering, 1995., Proceedings of the Second IEEE International Symposium on, pages 160 – 162, March 1995. (Cited on page 76.)

[Burns 2004] Alan Burns, Brian Dobbing and Tullio Vardanega. *Guide for the use of the Ada Ravenscar Profile in high integrity systems*. Ada Lett., vol. XXIV, pages 1–74, June 2004. (Cited on page 102.)

[Butler 1996a] M. Butler. *Stepwise Refinement of Communicating Systems*. Science of Computer Programming, vol. 27, pages 139–173, 1996. (Cited on page 51.)

[Butler 1996b] Ricky W. Butler. *An Introduction to Requirements Capture Using PVS: Specification of a Simple Autopilot*. NASA Technical Memorandum 110255, NASA Langley Research Center, Hampton, VA, May 1996. (Cited on pages 64, 87 and 88.)

[Butler 1998] M. Butler and M. Walden. *Parallel Programming with the B Method*. In Program Development by Refinement Cases Studies Using the B Method, volume [Sekerinski 1998] of *FACIT*, pages 183–195. Springer, 1998. (Cited on page 51.)

[Butler 2000] M. Butler. *CSP2B: A Practical Approach To Combining CSP and B*. Formal Aspects of Computing, vol. 12, pages 182–196, 2000. (Cited on page 51.)

[Butler 2008] Michael Butler and Divakar Yadav. *An incremental development of the Mondex system in Event-B*. Formal Asp. Comput., vol. 20, no. 1, pages 61–77, 2008. (Cited on page 42.)

[Cansell 2002] Dominique Cansell, Ganesh Gopalakrishnan, Michael Jones, Dominique Méry and Airy Weinzoepflen. *Incremental Proof of the Producer/Consumer Property for the PCI Protocol*. In Proceedings of the 2nd International Conference of B and Z Users on Formal Specification and Development in Z and B, ZB '02, pages 22–41, London, UK, UK, 2002. Springer-Verlag. (Cited on page 51.)

[Cansell 2006] Dominique Cansell, Dominique Méry and Joris Rehm. Formal specification and development in b, chapitre Time Constraint Patterns for Event B Development, pages 140–154. Lecture Notes in Computer Science. Springer US, 2006. ISSN 0302-9743. (Cited on page 168.)

[Cansell 2007] Dominique Cansell and Dominique Méry. The Event-B Modelling Method: Concepts and Case Studies, pages 33–140. Springer, 2007. See [Bjørner 2007]. (Cited on pages 6, 26, 52, 53, 57, 83, 162, 216 and 251.)

[CC] CC. *Common Criteria*. http://www.commoncriteriaportal.org/. (Cited on pages 2, 3, 7, 12, 13, 15, 17, 22, 23, 26, 32, 34, 62, 72, 207, 281, 283, 285 and 296.)

[CC 2009a] CC. *Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model*. http://www.iec.ch/, 2009. (Cited on pages 34 and 296.)

[CC 2009b] CC. *Common Criteria for Information Technology Security Evaluation, Part 2: Security Functional Requirements*. http://www.iec.ch/, 2009. (Cited on pages 34 and 296.)

[CC 2009c] CC. *Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance components*. http://www.iec.ch/, 2009. (Cited on pages 34, 296 and 297.)

[CDRH 2006] CDRH. *Center for Devices and Radiological Health, Safety of Marketed Medical Devices,US FDA*, 2006. (Cited on pages 62, 128 and 295.)

[72] J.L. Lions (chairman). *Ariane 5 Flight 501 Failure: Report by the Inquiry Board*. Rapport technique, European Space Agency, Paris, 1996. (Cited on page 32.)

[Chandy 1988] K. M. Chandy and J. Misra. Parallel program design a foundation. Addison-Wesley Publishing Company, 1988. ISBN 0-201-05866-9. (Cited on pages 50 and 51.)

[Clarke 1996] Edmund M. Clarke and Jeannette M. Wing. *Formal Methods: State of the Art and Future Directions*. ACM Computing Surveys, vol. 28, pages 626–643, 1996. (Cited on pages 31, 40, 43 and 45.)

[Clarke 1999] E. M. Clarke, O. Grumberg and D. Peled. Model checking. MIT Press, 1999. ISBN 978-0262032704. (Cited on pages 68, 153, 200, 239 and 252.)

[Clarke 2000] E. M. Clarke, O. Grumberg and D. A. Peled. Model checking. The MIT Press, 2000. (Cited on page 50.)

[Cle 2004] ClearSy, Aix-en-Provence (F). *B4FREE*, 2004. http://www.b4free.com. (Cited on page 50.)

[ClearSy] ClearSy. *Atelier B*. http://www.clearsy.com. (Cited on pages 49, 50 and 57.)

[Clocksin 1987] W. F. Clocksin and C. S. Mellish. Programming in prolog. Springer-Verlag New York, Inc., New York, NY, USA, 1987. (Cited on page 79.)

[Coleman 1979] D. Coleman and J. W. Hughes. *The clean termination of Pascal programs*. Acta Informatica, vol. 11, pages 195–210, 1979. 10.1007/BF00289066. (Cited on page 100.)

[Commission 1994] Main Commission. *Report on the accident to Airbus A320-211 Aircraft in Warsaw on 14 September 1993*. Rapport technique, Aircraft Accident Investigation Warsaw, Poland, 1994. (Cited on page 32.)

[Cousot 2007] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux and Xavier Rival. *Combination of abstractions in the ASTRÉE static analyzer*. In Proceedings of the 11th Asian computing science conference on Advances in computer science: secure software and related issues, Lecture Notes in Computer Science, pages 272–300, Berlin, Heidelberg, 2007. Springer-Verlag. (Cited on pages 32, 64 and 100.)

[Crocker 2003] David Crocker. *Perfect Developer: A tool for Object-Oriented Formal Specification and Refinement. Tools Exhibition Notes at Formal Methods Europe*. In Tools Exhibition Notes at Formal Methods Europe, page 2003, 2003. (Cited on page 163.)

[Crow 1995] Judy Crow, Sam Owre, John Rushby, Natarajan Shankar and Ayam Srivas. *A Tutorial Introduction to PVS*. In Computer Science Laboratory, SRI International, 1995. (Cited on pages 41, 43, 44 and 62.)

[Cullyer 1989] W.J. Cullyer. *Implementing high integrity systems: the VIPER microprocessor*. Aerospace and Electronic Systems Magazine, IEEE, vol. 4, no. 6, pages 5 –13, jun 1989. (Cited on page 41.)

[Cullyer 1991] W. J. Cullyer, S. J. Goodenough and B. A. Wichmann. *The choice of computer languages for use in safety critical systems*. Softw. Eng. J., vol. 6, pages 51–58, March 1991. (Cited on page 35.)

[Davis 1992] Alan M. Davis. *Operational Prototyping: A New Development Approach*. IEEE Softw., vol. 9, pages 70–78, September 1992. (Cited on page 78.)

[Dis] (Cited on page 63.)

[Dorticós 2006] F. Dorticós, M. A. Quiones, F. Tornes, Y. Fayad, R. Zayas, J. Castro, A. Barbetta and F. Gregorio. *Rate-Responsive Pacing Controlled by the TVI Sensor in the Treatment of Sick Sinus Syndrome*. In Antonio Raviele, Editor, Cardiac Arrhythmias 2005, pages 581–590. Springer Milan, 2006. (Cited on page 81.)

[Dotti 2009] Fernando Dotti, Alexei Iliasov, Leila Ribeiro and Alexander Romanovsky. *Modal Systems: Specification, Refinement and Realisation*. In Karin Breitman and Ana Cavalcanti, Editors, Formal Methods and Software Engineering, volume 5885 of *Lecture Notes in Computer Science*, pages 601–619. Springer Berlin / Heidelberg, 2009. (Cited on page 88.)

[Duce 1997] D. A. Duce. *Formal Methods and Standards - An Idiosyncratic View, in 2nd BCS-FACS Northern Formal Methods Workshop, Ilkley*. Sci. Comput. Program., 1997. (Cited on page 290.)

[EB2ALL 2011] EB2ALL. *Automatic code generation from Event-B to many Programming Languages*. http://eb2all.loria.fr/, 2011. (Cited on pages 3, 5, 6, 7, 8, 14, 16, 19, 23, 24, 25, 26, 27, 95, 96, 98, 124, 126, 162, 164, 207, 208, 213, 217, 241, 242, 247, 282, 283 and 286.)

[Edmunds 2010] Andrew Edmunds and Michael Butler. *Tool Support for Event-B Code Generation*. In WS-TBFM2010, February 2010. (Cited on page 98.)

[Edmunds 2011] Andrew Edmunds and Michael Butler. *Tasking Event-B: An Extension to Event-B for Generating Concurrent Code*. In PLACES 2011, February 2011. (Cited on page 98.)

[Ellenbogen 2005] Kenneth A. Ellenbogen and Mark A. Wood. Cardiac Pacing and ICDs. 4th Edition, Blackwell, 2005. ISBN-10 1-4051-0447-3. (Cited on pages 131, 165, 255 and 256.)

[Farn 2004] Kwo-Jean Farn, Shu-Kuo Lin and Andrew Ren-Wei Fung. *A study on information security management system evaluation–assets, threat and vulnerability*. Computer Standards & Interfaces, vol. 26, no. 6, pages 501 – 513, 2004. (Cited on pages 15, 72 and 283.)

[FDA] FDA. *Food and Drug Administration*. http://www.fda.gov/. (Cited on pages 2, 3, 7, 12, 13, 17, 22, 23, 26, 32, 34, 62, 65, 207, 281, 285 and 295.)

[Fitzgerald 2007] John Fitzgerald. The Typed Logic of Partial Functions and the Vienna Development Method, pages 431–465. Springer, 2007. See [Bjørner 2007]. (Cited on pages 51 and 128.)

[Fitzgerald 2008] John S Fitzgerald, Jeremy W Bryans, David Greathead, Clifff B Jones and Richard Payne, Editors. Animation-based validation of a formal model of dynamic virtual organisations. Electronic Workshops in Computing, The British Computer Society, 2008. (Cited on page 79.)

[Fitzgerald 2010] John Fitzgerald, Peter Gorm Larsen, Ken Pierce, Marcel Verhoef and Sune Wolff. *Collaborative modelling and co-simulation in the development of dependable embedded systems*. In Proceedings of the 8th international conference on Integrated formal methods, Lecture Notes in Computer Science, pages 12–26, Berlin, Heidelberg, 2010. Springer-Verlag. (Cited on page 128.)

[Fohler 1992] Gerhard Fohler. *Realizing Changes of Operational Modes with a Pre Run-Time Scheduled Hard Real-Time System*. In Proceedings of the Second International Workshop on Responsive Computer Systems, pages 287–300. Springer Verlag, 1992. (Cited on page 88.)

[Forder 1993] Higgins C. McDermid J. Forder J. and G. Storrs. *SAM-A Tool to Support the Construction, Review and Evolution of Safety Arguments*. In Directions in Safety-Critical Systems, pages 195–216. Springer-Verlag: London, 1993. (Cited on page 39.)

[Fox 1998] John Fox, Nicky Johns and Ali Rahmanzadeh. *Disseminating medical knowledge: the PROforma approach*. Artificial Intelligence in Medicine, vol. 14, no. 1-2, pages 157–182, 1998. (Cited on page 251.)

[Frehse 2008] Goran Frehse. *PHAVer: algorithmic verification of hybrid systems past HyTech*. Int. J. Softw. Tools Technol. Transf., vol. 10, pages 263–279, May 2008. (Cited on page 218.)

[Fries C. 2011] Richard Fries C. Handbook of Medical Device Design. Marcel Dekker, 2011. (Cited on pages 33, 289, 290 and 292.)

[Fuchs 1992] N.E. Fuchs. *Specifications are (preferably) executable*. Software Engineering Journal, vol. 7, no. 5, pages 323 –334, September 1992. (Cited on page 78.)

[Gall 2008] Heinz Gall. *Functional safety IEC 61508 / IEC 61511 the impact to certification and the user*. In Proceedings of the 2008 IEEE/ACS International Conference on Computer Systems and Applications, AICCSA '08, pages 1027–1031, Washington, DC, USA, 2008. IEEE Computer Society. (Cited on pages 33, 290 and 291.)

[Galvao 2007] Ismenia Galvao and Arda Goknil. *Survey of Traceability Approaches in Model-Driven Engineering*. In Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, pages 313–, Washington, DC, USA, 2007. IEEE Computer Society. (Cited on page 77.)

[Gamma 1994] E. Gamma, R. Helm, R. Johnson, R. Vlissides and P. Gamma. Design Patterns : Elements of Reusable Object-Oriented Software design Patterns. Addison-Wesley Professional Computing, 1994. (Cited on page 167.)

[Gaudel 1996] M.-C. Gaudel and J. Woodcock, Editors. FME '96: Industrial Benefit and Advances in Formal Methods, Third International Symposium of Formal Methods Europe, Co-Sponsored by IFIP WG 14.3, Oxford, UK, March 18-22, 1996, Proceedings, volume 1051 of *Lecture Notes in Computer Science*. Springer, 1996. (Cited on pages 2 and 22.)

[Georg 2001] Geri Georg, Jores Bieman and Robert B. France. *Using Alloy and UML/OCL to Specify Run-Time Configuration Management: A Case Study*. In Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists, volume 7, pages 128–141. German Informatics, 2001. (Cited on page 97.)

[Gibbs 1994] Wayt W. Gibbs. Software's Chronic Crisis. Scientific American. September 1994. (Cited on pages 2, 21 and 32.)

[Goguen 1982] Joseph Goguen and Jose Meseguer. *Rapid prototyping: in the OBJ executable specification language*. SIGSOFT Softw. Eng. Notes, vol. 7, pages 75–84, April 1982. (Cited on page 78.)

[Goldman 1974] B. S. Goldman, E. J. Noble, J. G. Heller and D. Covvey. *The Pacemaker Challenge*. CMAJ, vol. 110, no. 1, pages 28–31, 1974. (Cited on pages 84 and 162.)

[Gomes 2009] Artur Gomes and Marcel Oliveira. *Formal Specification of a Cardiac Pacing System*. In Ana Cavalcanti and Dennis Dams, Editors, FM 2009: Formal Methods, volume 5850 of *Lecture Notes in Computer Science*, pages 692–707. Springer Berlin / Heidelberg, 2009. (Cited on pages 64, 163 and 164.)

[Gomes 2010] Artur Oliveira Gomes and Marcel Vinicius Medeiros Oliveira. *Formal Development of a Cardiac Pacemaker: from Specification to Code*. In SBFM 2010, Lecture Notes in Computer Science, pages 213–228, 2010. (Cited on page 163.)

[Gordon 1983] M.J.C. Gordon. LCF-LSM: a system for specifying and verifying hardware. University of Cambridge Computer Laboratory, 1983. (Cited on page 42.)

[Gosling 2005] James Gosling, Bill Joy, Guy Steele and Gilad Bracha. Java(TM) Language Specification, The (3rd Edition) (Java (Addison-Wesley)). Addison-Wesley Professional, 2005. (Cited on page 95.)

[Gotel 1994] Orlena C. Z. Gotel and Anthony C. W. Finkelstein. *An Analysis of the Requirements Traceability Problem*. pages 94–101, 1994. (Cited on page 77.)

[guo 2009] Ying guo, Meihong Yang, Jun Wang, Ping Yang and Feng Li. *An Ontology Based Improved Software Requirement Traceability Matrix*. In Proceedings of the 2009 Second International Symposium on Knowledge Acquisition and Modeling - Volume 01, KAM '09, pages 160–163, Washington, DC, USA, 2009. IEEE Computer Society. (Cited on page 77.)

[Gurevitch 1995] Y. Gurevitch. Specification and validation methods, chapitre "Evolving Algebras 1993: Lipari Guide", pages 9–36. Oxford University Press, 1995. (Cited on page 52.)

[Halbwachs 1991] N. Halbwachs, P. Caspi, P. Raymond and D. Pilaud. *The synchronous dataflow programming language LUSTRE*. In Proceedings of the IEEE, pages 1305–1320, 1991. (Cited on pages 32 and 64.)

[Hall 1996] A. Hall. *Using formal methods to develop an ATC information system*. Software, IEEE, vol. 13, no. 2, pages 66 –76, mar 1996. (Cited on page 40.)

[Harel 1987a] David Harel. Algorithmics: the spirit of computing. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987. (Cited on page 41.)

[Harel 1987b] David Harel. *Statecharts: A visual formalism for complex systems*. Sci. Comput. Program., vol. 8, no. 3, pages 231–274, 1987. (Cited on page 90.)

[Harrild 2000] David M. Harrild, Craig S. Henriquez, The Human Atria, David M. Harrild and Craig S. Henriquez. *A Computer Model of Normal Conduction*. In in the Human Atria, Circ Res, vol 87, pages 25–36, 2000. (Cited on pages 47, 128, 129, 139 and 202.)

[Hayes 1989] Ian Hayes and C. B. Jones. *Specifications are not (necessarily) executable*. Softw. Eng. J., vol. 4, pages 330–338, November 1989. (Cited on page 79.)

[Heimdahl 1996] Mats P. E. Heimdahl and Nancy G. Leveson. *Completeness and Consistency in Hierarchical State-Based Requirements*. IEEE Trans. Softw. Eng., vol. 22, pages 363–377, June 1996. (Cited on page 41.)

[Heitmeyer 2008] Constance L. Heitmeyer, Myla Archer, Elizabeth I. Leonard and John McLean. *Applying Formal Methods to a Certifiably Secure Software System*. IEEE Trans. Software Eng., vol. 34, no. 1, pages 82–98, 2008. (Cited on page 64.)

[Heitmeyer 2009] Constance L. Heitmeyer. *On the Role of Formal Methods in Software Certification: An Experience Report*. Electr. Notes Theor. Comput. Sci., vol. 238, no. 4, pages 3–9, 2009. (Cited on page 64.)

[Henderson 1986a] P Henderson. *Functional programming, formal specification, and rapid prototyping*. IEEE Trans. Softw. Eng., vol. 12, pages 241–250, February 1986. (Cited on page 78.)

[Henderson 1986b] P. Henderson and C. J. Minkowitz. *The me too Method of Software Design*. ICL Journal, vol. 5, no. 1, 1986. (Cited on page 78.)

[Hennebert 1993] C. Hennebert and G. Guiho. *SACEM: A fault tolerant system for train speed control*. In Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on, pages 624 –628, jun 1993. (Cited on page 41.)

[Henson 2007] Martin C. Henson, Moshe Deutsch and Steve Reeves. Z logic and its applications, pages 467–569. Springer, 2007. See [Bjørner 2007]. (Cited on page 51.)

[Hesselson 2003] Aaron Hesselson. Simplified Interpretations of Pacemaker ECGs. Blackwell Publishers, 2003. ISBN 978-1-4051-0372-5. (Cited on pages 131, 165, 178, 182, 189, 190, 191, 197, 204, 255 and 256.)

[Hinchey 1995] M.G. Hinchey and J.P. Bowen. Applications of formal methods. Prentice-Hall international series in computer science. Prentice Hall, 1995. (Cited on page 43.)

[Hoare 1996] J. Hoare, J. Dick, D. Neilson and I. Holm Sørensen. *Applying the B Technologies on CICS*. In FME 96, pages 74–84. Springer, 1996. (Cited on page 51.)

[Hoare 2003] Tony Hoare. *The verifying compiler: A grand challenge for computing research*. J. ACM, vol. 50, no. 1, pages 63–69, 2003. (Cited on page 162.)

[Hoare 2009] C.A.R. Hoare, Jayadev Misra, Gary T. Leavens and Natarajan Shankar. *The verified software initiative: A manifesto*. ACM Comput. Surv., vol. 41, no. 4, pages 1–8, 2009. (Cited on pages 7, 26, 84, 162 and 211.)

[Holzmann 1997] Gerard J. Holzmann. *The Model Checker SPIN*. IEEE Trans. Softw. Eng., vol. 23, pages 279–295, May 1997. (Cited on page 252.)

[Hosier 1961] W. A. Hosier. *Pitfalls and Safeguards in Real-Time Digital Systems with Emphasis on Programming*. Engineering Management, IRE Transactions on, vol. EM-8, no. 2, pages 99 –115, june 1961. (Cited on page 63.)

[Houser Amy 2005] John Pierowicz Houser Amy and Roger McClellan. *Concept of Operations and Voluntary Operational Requirements for Forward Collision Warning Systems (CWS) and Adaptive Cruise Control (ACC) Systems On-board Commercial Motor Vehicles*. FMCSA, Federal Motor Carrier Safety Administration Office of Research and Analysis, Virginia, VA, 2005. (Cited on pages 225 and 239.)

[Houston 1991] Iain Houston and Steve King. *CICS project report experiences and results from the use of Z in IBM*. In S. Prehn and W. Toetenel, Editors, VDM'91 Formal Software Development Methods, volume 551 of *Lecture Notes in Computer Science*, pages 588–596. Springer Berlin / Heidelberg, 1991. (Cited on page 40.)

[Hrovat 1991] D. Hrovat and W.J. Johnson. *Automotive control systems: past, present, future*. In International Conference on Industrial Electronics, Control and Instrumentation (IECON), 1991. (Cited on pages 216 and 239.)

[Huhn 2010] Michaela Huhn and Axel Zechner. *Arguing for Software Quality in an IEC 62304 Compliant Development Process*. In Tiziana Margaria and Bernhard Steffen, Editors, Leveraging Applications of Formal Methods, Verification, and Validation, volume 6416 of *Lecture Notes in Computer Science*, pages 296–311. Springer Berlin / Heidelberg, 2010. (Cited on pages 64, 290, 291 and 292.)

[Hull 2005] M. Elizabeth C. Hull, Ken Jackson and Jeremy Dick. Requirements engineering, second edition. Springer, 2005. (Cited on page 77.)

[IEC61508 2008] IEC61508. *IEC functional safety and IEC 61508: Working draft on functional safety of electrical/electronic/programmable electronic safety-related systems*. http://www.iec.ch/, 2008. (Cited on pages 33, 42, 290 and 291.)

[IEC62304 2006] IEC62304. *International Electrotechnical Commission: Medical device software - Software life-cycle processes*. http://www.iec.ch/, 2006. (Cited on pages 15, 33, 62, 72, 283, 290, 291 and 292.)

[IEEE-SA] IEEE-SA. *IEEE Standards Association*. http://standards.ieee.org/. (Cited on pages 2, 3, 7, 12, 13, 17, 22, 23, 26, 32, 33, 34, 62, 65, 72, 207, 281, 285, 289, 293, 294 and 295.)

[IEEE Std. 1012-1998] IEEE Std. 1012-1998. *IEEE Standard for Software Verification and Validation*. http://standards.ieee.org/. (Cited on page 294.)

[IEEE Std. 1074-1997] IEEE Std. 1074-1997. *IEEE Standard for Developing Software Life Cycle Processes*. http://standards.ieee.org/. (Cited on pages 293 and 294.)

[IEEE Std. 730-1998] IEEE Std. 730-1998. *IEEE Standard for Software Quality Assurance Plans*. http://standards.ieee.org/. (Cited on page 294.)

[IEEE Std 1990] 610.12-1990 IEEE Std. *IEEE Standard Glossary of Software Engineering Terminology*. page 1, 1990. (Cited on pages 294 and 295.)

[Imperial Chemical Industries 1977] ltd Imperial Chemical Industries, Chemical Industries Association. Chemical Industry Safety and Health Council. A guide to hazard and operability studies. Chemical Industry Safety and Health Council of the Chemical Industries Associatin, 1977. (Cited on pages 36 and 37.)

[Isern 2008] David Isern and Antonio Moreno. *Computer-based execution of clinical guidelines: A review*. International Journal of Medical Informatics, vol. 77, no. 12, pages 787 – 808, 2008. (Cited on page 252.)

[ISO] ISO. *International Organization for Standardization*. http://www.iso.ch/. (Cited on pages 2, 3, 7, 12, 13, 17, 22, 23, 26, 32, 33, 34, 62, 65, 72, 207, 281, 285, 289, 290, 292 and 296.)

[Jackson 2002] Daniel Jackson. *Alloy: a lightweight object modelling notation*. ACM Trans. Softw. Eng. Methodol., vol. 11, no. 2, pages 256–290, 2002. (Cited on pages 153, 201 and 239.)

[Jackson 2007] Michael Jackson. *The Problem Frames Approach to Software Engineering*. In APSEC, page 14, 2007. (Cited on page 67.)

[Jahanian 1994] Farnam Jahanian and Aloysius K. Mok. *Modechart: A Specification Language for Real-Time Systems*. IEEE Trans. Softw. Eng., vol. 20, no. 12, pages 933–947, 1994. (Cited on page 88.)

[Jee 2010] Eunkyoung Jee, Shaohui Wang, Jeong-Ki Kim, Jaewoo Lee, Oleg Sokolsky and Insup Lee. *A Safety-Assured Development Approach for Real-Time Software*. In 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA, pages 133–142, 2010. (Cited on page 130.)

[Jetley 2004] Raoul Praful Jetley, Cohan Carlos and S. Purushothaman Iyer. *A case study on applying formal methods to medical devices: computer-aided resuscitation algorithm*. International Journal on Software Tools for Technology Transfer, vol. 5, no. 4, pages 320–330, 2004. (Cited on pages 32, 64 and 128.)

[Jetley 2006] R. Jetley, S. Purushothaman Iyer and P. Jones. *A formal methods approach to medical device review*. Computer, vol. 39, no. 4, pages 61–67, April 2006. (Cited on pages 2, 22, 31, 32, 64, 128 and 295.)

[Jiang 2010] Zhihao Jiang, Miroslav Pajic, Allison T. Connolly, Sanjay Dixit and Rahul Mangharam. *Real-time Heart Model for Implantable Cardiac Device Validation and Verification*. In 22st Euromicro Conference on Real-Time Systems, (IEEE ECRTS'10), 07/2010 2010. (Cited on page 130.)

[Jones 1986a] C. B. Jones. Systematic software development using vdm. Prentice-Hall International, 1986. (Cited on page 51.)

[Jones 1986b] Clifford B. Jones. Systematic Software Development using VDM. Prentice Hall International Series in Computer Science. Prentice Hall, 1986. (Cited on pages 43, 44 and 62.)

[Joshi 2005] Anjali Joshi and Mats Per Erik Heimdahl. *Model-Based Safety Analysis of Simulink Models Using SCADE Design Verifier*. In SAFECOMP, pages 122–135, 2005. (Cited on page 64.)

[Kaivola 2009] Roope Kaivola, Rajnish Ghughal, Naren Narasimhan, Amber Telfer, Jesse Whittemore, Sudhindra Pandav, Anna Slobodová, Christopher Taylor, Vladimir Frolov, Erik Reeber and Armaghan Naik. *Replacing Testing with Formal Verification in Intel CoreTM i7 Processor Execution Engine Validation*. In Proceedings of the 21st International Conference on Computer Aided Verification, CAV '09, pages 414–429, Berlin, Heidelberg, 2009. Springer-Verlag. (Cited on page 42.)

[Keatley 1999] K L Keatley. *A review of the FDA draft guidance document for software validation: guidance for industry.* Qual Assur, vol. 7, no. 1, pages 49–55, 1999. (Cited on pages 17, 34, 62, 65, 76, 85, 154, 213, 285, 294 and 295.)

[Kernighan 1988] Brian W. Kernighan and Dennis Ritchie. C Programming Language. Prentice Hall, 1988. ISBN-100131103628. (Cited on page 95.)

[Khan 2008] M. Gabriel Khan. Rapid ECG Interpretation . Humana Press, 2008. (Cited on pages 47, 127, 129, 130, 131, 132, 133, 136, 141, 143, 147, 148, 149, 202, 254, 255, 256, 259, 260, 263, 264, 265, 266, 268, 269, 270, 271 and 278.)

[Knospe 2006] C. Knospe. *PID control*. Control Systems Magazine, IEEE, vol. 26, no. 1, pages 30 – 31, 2006. (Cited on page 216.)

[Kosara 2002] Robert Kosara, Silvia Miksch, Andreas, Andreas Seyfang and Peter Votruba. *Tools for Acquiring Clinical Guidelines in Asbru*, 2002. (Cited on page 253.)

[Kowalski 1985] R. Kowalski. *The relation between logic programming and logic specification*. In Proc. of a discussion meeting of the Royal Society of London on Mathematical logic and programming languages, pages 11–27, Upper Saddle River, NJ, USA, 1985. Prentice-Hall, Inc. (Cited on page 78.)

[Kye-Rok Jun 1994] Yeong Rak Seong Kye-Rok Jun and Tag Gon Kim. *A Cellular Automata Model of Activation Process in Ventricular Muscle*. pages 769 – 774, July 1994. (Cited on page 129.)

[Lamport 1994] L. Lamport. *A Temporal Logic of Actions*. ACM Transactions on Programming Languages and Systems, vol. 16, no. 3, pages 872–923, May 1994. (Cited on page 51.)

[Lamport 2002] L. Lamport. Specifying Systems: The TLA$^+$+ Language and Tools for Hardware and Software Engineers. Addison-Wesley, 2002. (Cited on page 51.)

[Lano 1999] K. Lano, J. Bicarregui and A. Sanchez. *Invariant-based Synthesis and Composition of Control Algorithms using B*. In FM'99 – B Users Group Meeting – Applying B in an industrial context: Tools, Lessons and Techniques, pages 69–86, 1999. (Cited on page 51.)

[Leavens 2006] Gary T. Leavens, Jean-Raymond Abrial, Don Batory, Michael Butler, Alessandro Coglio, Kathi Fisler, Eric Hehner, Cliff Jones, Dale Miller, Simon Peyton-Jones, Murali Sitaraman, Douglas R. Smith and Aaron Stump. *Roadmap for Enhanced Languages and Methods to Aid Verification*. In Fifth Intl. Conf. Generative Programming and Component Engineering (GPCE 2006), pages 221–235. ACM, October 2006. (Cited on page 56.)

[Lecomte 2007] Thierry Lecomte, Thierry Servat and Guilhem Pouzancre. *Formal Methods in Safety-Critical Railway Systems*. ClearSy, no. HCSS+, 2007. (Cited on page 41.)

[Lee 2002] Edward A. Lee. *Embedded Software*. Advances in Computers, vol. 56, pages 56–97, 2002. (Cited on page 74.)

[Lee 2006a] Insup Lee, George J. Pappas, Rance Cleaveland, John Hatcliff, Bruce H. Krogh, Peter Lee, Harvey Rubin and Lui Sha. *High-Confidence Medical Device Software and Systems*. Computer, vol. 39, no. 4, pages 33–38, 2006. (Cited on page 128.)

[Lee 2006b] Insup Lee, George J. Pappas, Rance Cleaveland, John Hatcliff, Bruce H. Krogh, Peter Lee, Harvey Rubin and Lui Sha. *High-Confidence Medical Device Software and Systems*. Computer, vol. 39, no. 4, pages 33–38, 2006. (Cited on pages 131, 165, 182, 189, 190 and 191.)

[Leuschel 2003] Michael Leuschel and Michael Butler. ProB: A Model Checker for B, pages 855–874. Lecture Notes in Computer Science. Springer, 2003. (Cited on pages 6, 26, 68, 153, 162 and 200.)

[Leveson 1983] Nancy G. Leveson and Peter R. Harvey. *Software fault tree analysis*. Journal of Systems and Software, vol. 3, no. 2, pages 173 – 181, 1983. (Cited on page 37.)

[Leveson 1991] Nancy G. Leveson. *Software safety in embedded computer systems*. Commun. ACM, vol. 34, pages 34–46, February 1991. (Cited on pages 4, 15, 23, 37, 65, 72 and 283.)

[Leveson 1993] N. G. Leveson and C. S. Turner. *An Investigation of the Therac-25 Accidents*. Computer, vol. 26, pages 18–41, July 1993. (Cited on pages 2, 21 and 32.)

[Leveson 1994] Nancy G. Leveson, Mats Per, Erik Heimdahl, Student Member, Holly Hildreth and Jon Damon Reese. *Requirements specification for process-control systems*. IEEE Transactions on Software Engineering, vol. 20, pages 684–707, 1994. (Cited on page 41.)

[Lindvall 1996] Mikael Lindvall and Kristian Sandahl. *Practical Implications of Traceability*. Software: Practice and Experience, vol. 26, no. 10, pages 1161–1180, 1996. (Cited on page 77.)

[Lohr 1990] Kathleen N. Lohr, Marilyn J. Field, United States. and Institute of Medicine (U.S.). Clinical practice guidelines : directions for a new program / committee to advise the public health service on clinical practice guidelines, institute of medicine ; marilyn j. field and kathleen n. lohr, editors. National Academy Press, Washington, D.C. :, 1990. (Cited on page 250.)

[Love 2006] Charles J. Love. Cardiac pacemakers and defibrillators. Landes Bioscience Publishers, 2006. ISBN 1-57059-691-3. (Cited on pages 131, 165 and 255.)

[Ltd 1996] B-Core(UK) Ltd. *B-Toolkit User's Manual*, relase 3.2 edition, 1996. (Cited on page 49.)

[Macedo 2008] Hugo Daniel Macedo, Peter Gorm Larsen and John Fitzgerald. *Incremental Development of a Distributed Real-Time Model of a Cardiac Pacing System Using VDM*. In Proceedings of the 15th international symposium on Formal Methods (FM'08), Lecture Notes in Computer Science, pages 181–197, Berlin, Heidelberg, 2008. Springer-Verlag. (Cited on pages 32, 64, 84, 159, 162, 163 and 164.)

[Magee 2003] J Harvey Magee. *Validation of medical modeling & simulation training devices and systems.* Stud Health Technol Inform, vol. 94, pages 196–8, 2003. (Cited on page 64.)

[Maibaum 2008] T. S. E. Maibaum and Alan Wassyng. *A Product-Focused Approach to Software Certification*. IEEE Computer, vol. 41, no. 2, pages 91–93, 2008. (Cited on page 64.)

[Maisel 2001] William H. Maisel, Michael O. Sweeney, William G. Stevenson, Kristin E. Ellison and Laurence M. Epstein. *Recalls and Safety Alerts Involving Pacemakers and Implantable Cardioverter-Defibrillator Generators*. JAMA: The Journal of the American Medical Association, vol. 286, no. 7, pages 793–799, 2001. (Cited on pages 61 and 128.)

[Makowiec 2008] Danuta Makowiec. *The Heart Pacemaker by Cellular Automata on Complex Networks*. In Proceedings of the 8th international conference on Cellular Automata for Reseach and Industry, ACRI '08, pages 291–298, Berlin, Heidelberg, 2008. Springer-Verlag. (Cited on page 138.)

[Malmivuo 1995] Jaakko Malmivuo. Bioelectromagnetism. Oxford University Press, 1995. ISBN 0-19-505823-2. (Cited on pages 127, 130, 131, 133, 136, 139, 140, 141, 147, 148, 149, 165, 178, 197, 204, 255 and 256.)

[Manna 2009] Valerio Panzica La Manna, Andrea Tommaso Bonanno and Alfredo Motta. *Poster on A simple pacemaker Implementation*. ACM, May 2009. (Cited on page 163.)

[Marciniak 2002] John J. Marciniak. Encyclopedia of software engineering. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2002. (Cited on pages 63 and 64.)

[Marcos 2001] Mar Marcos, Geert Berger, Frank van Harmelen, Annette ten Teije, Hugo Roomans and Silvia Miksch. *Using Critiquing for Improving Medical Protocols: Harder than It Seems*. In Proceedings of the 8th Conference on AI in Medicine in Europe: Artificial Intelligence Medicine, AIME '01, pages 431–441, London, UK, UK, 2001. Springer-Verlag. (Cited on page 252.)

[Mashkoor 2009] Atif Mashkoor, Jean-Pierre Jacquot and Jeanine Souquières. *Transformation Heuristics for Formal Requirements Validation by Animation*. In 2nd International Workshop on the Certification of Safety-Critical Software Controlled Systems - SafeCert 2009, York Royaume-Uni, 2009. (Cited on pages 78 and 84.)

[MATLAB] MATLAB. *SIMULINK - Simulation and Model Based Design*. http://www.mathworks.com. (Cited on pages 225, 227, 239 and 245.)

[McDermid 1991] John McDermid. Software Engineer's Reference Book. CRC Press, Inc., Boca Raton, FL, USA, 1991. (Cited on page 76.)

[Mead 2003] Nancy R. Mead, Nancy R. Mead and Christos Scondras. *International Liability Issues for Software Quality*, 2003. (Cited on pages 15, 283 and 297.)

[Méry 2009] Dominique Méry and Neeraj Kumar Singh. *Pacemaker's Functional Behaviors in Event-B*. Research Report (http://hal.inria.fr/inria-00419973/en/), MOSEL - LORIA - INRIA - CNRS : UMR7503 - Université Henri Poincaré - Nancy I - Université Nancy II - Institut National Polytechnique de Lorraine, 2009. (Cited on pages 5, 17, 25, 84, 163, 172, 200, 203, 207 and 284.)

[Méry 2010a] Dominique Méry and Neeraj Kumar Singh. *EB2C : A Tool for Event-B to C Conversion Support, Poster and Tool Demo submission and published in a CNR Technical Report in SEFM*. 2010. (Cited on pages 3, 5, 6, 8, 14, 16, 19, 23, 24, 27, 95, 124, 126, 162, 164, 207, 208, 213, 217, 242, 247, 282, 283, 284 and 286.)

[Méry 2010b] Dominique Méry and Neeraj Kumar Singh. *Real-Time Animation for Formal Specification*. In Marc Aiguier, Francis Bretaudeau and Daniel Krob, Editors, Complex Systems Design & Management, pages 49–60. Springer Berlin Heidelberg, 2010. (Cited on pages 3, 4, 6, 7, 13, 15, 17, 23, 24, 25, 26, 27, 65, 69, 73, 74, 76, 79, 84, 85, 128, 206, 211, 281, 283 and 284.)

[Méry 2010c] Dominique Méry and Neeraj Kumar Singh. *Technical Report on Formal Development of Two-Electrode Cardiac Pacing System*. (http://hal.archives-ouvertes.fr/inria-00465061/en/), MOSEL - LORIA - INRIA - CNRS : UMR7503 - Université Henri Poincaré - Nancy I - Université Nancy II - Institut National Polytechnique de Lorraine, 2010. (Cited on pages 5, 25, 163, 172, 200, 203 and 207.)

[Méry 2010d] Dominique Méry and Neeraj Kumar Singh. *Trustable Formal Specification for Software Certification*. In Tiziana Margaria and Bernhard Steffen, Editors,

Leveraging Applications of Formal Methods, Verification, and Validation, volume 6416 of *Lecture Notes in Computer Science*, pages 312–326. Springer Berlin / Heidelberg, 2010. (Cited on pages 6, 7, 14, 26, 92, 128 and 207.)

[Méry 2011a] Dominique Méry and Neeraj Singh. *A Generic Framework: From Modeling to Code*. Innovations in Systems and Software Engineering, pages 1–9, 2011. (Cited on pages 6, 25, 162 and 207.)

[Méry 2011b] Dominique Méry and Neeraj Kumar Singh. *Automatic Code Generation from Event-B Models*. In Proceedings of the 2011 Symposium on Information and Communication Technology (SoICT), Hanoi, Vietnam, 2011. ACM, ACM International Conference Proceeding Series. (Cited on pages 3, 5, 6, 7, 8, 16, 23, 24, 25, 26, 27, 95, 96, 98, 124, 126, 162, 164, 207, 208, 213, 217, 241, 242, 247, 282, 283, 284 and 286.)

[Méry 2011c] Dominique Méry and Neeraj Kumar Singh. *EB2J : Code Generation from Event-B to Java (Short Paper) in 14th Brazilian Symposium on Formal Methods (SBMF'11)*. 2011. (Cited on pages 5, 6, 8, 16, 24, 27, 95, 124, 126, 162, 164, 207, 208, 213, 217, 242, 247, 282, 283, 284 and 286.)

[Méry 2011d] Dominique Méry and Neeraj Kumar Singh. *Formal Development and Automatic Code Generation : Cardiac Pacemaker*. In International Conference on Computers and Advanced Technology in Education (ICCATE'11). Communications in Computer and Information Science (CCIS), Springer, 2011. (Cited on pages 8, 27, 124, 126, 162 and 207.)

[Méry 2011e] Dominique Méry and Neeraj Kumar Singh. *Formal Specification of Medical Systems by Proof-Based Refinement (in Press)*. ACM Transaction Embedded Computing Systems, 2011. (Cited on pages 3, 6, 7, 14, 23, 25, 26, 92, 169, 207, 224, 225 and 281.)

[Méry 2011f] Dominique Méry and Neeraj Kumar Singh. *Formalisation of the Heart based on Conduction of Electrical Impulses and Cellular-Automata*. In International Symposium on Foundations of Health Information Engineering and Systems (FHIES'11). 2011. (Cited on pages 3, 6, 7, 14, 23, 25, 26, 128, 130, 132, 154, 202 and 282.)

[Méry 2011g] Dominique Méry and Neeraj Kumar Singh. *Functional Behavior of a Cardiac Pacing System*. International Journal of Discrete Event Control Systems, vol. 1, no. 2, pages 129–149, 2011. (Cited on pages 5, 6, 7, 17, 19, 25, 26, 163, 164, 172, 200, 203, 207, 284 and 287.)

[Méry 2011h] Dominique Méry and Neeraj Kumar Singh. *Medical Protocol Diagnosis using Formal Methods*. In International Symposium on Foundations of Health Information Engineering and Systems (FHIES'11). 2011. (Cited on pages 6, 7, 18, 25, 27, 250, 251, 274, 275, 278 and 285.)

[Méry 2011i] Dominique Méry and Neeraj Kumar Singh. *Technical Report on Formalisation of the Heart using Analysis of Conduction Time and Velocity of the Electrocardiography and Cellular-Automata*. Technical report, http://hal.inria.fr/inria-00600339/en/, MOSEL - LORIA - INRIA - CNRS : UMR7503 - Université Henri Poincaré - Nancy I - Université Nancy II - Institut National Polytechnique de Lorraine, 2011. (Cited on pages 3, 6, 14, 23, 128, 130, 132, 154, 202, 203 and 282.)

[Méry 2011j] Dominique Méry and Neeraj Kumar Singh. *Technical Report on Interpretation of the Electrocardiogram (ECG) Signal using Formal Methods*. Technical report, http://hal.inria.fr/inria-00584177/en/, MOSEL - LORIA - INRIA - CNRS : UMR7503 - Université Henri Poincaré - Nancy I - Université Nancy II - Institut National Polytechnique de Lorraine, 2011. (Cited on pages 6, 18, 25, 250, 251, 264, 274, 275, 278 and 285.)

[Merz 2007] Stephan Merz. The Specification Language TLA$^+$, pages 381–430. Springer, 2007. See [Bjørner 2007]. (Cited on page 51.)

[Michael Balser 2004] Joyce van Croonenborg Christoph Duelli Frank van Harmelen Albert Jovell Peter Lucas Mar Marcos Silvia Miksch Wolfgang Reif Kitty Rosenbrand Andreas Seyfang Annette ten Teije Michael Balser Oscar Coltell. *Protocure: Supporting the Development of Medical Protocols through Formal Methods*. In Proceedings of the Symposium of Computerised Protocols and Guidelines(SCPG-04), Lecture Notes in Artificial Intelligence, Prague, April 2004. Springer Verlag. (Cited on page 253.)

[Miksch 2005] Silvia Miksch, Jim Hunter and Elpida T. Keravnou, Editors. Artificial Intelligence in Medicine, 10th Conference on Artificial Intelligence in Medicine, AIME'05, Aberdeen, UK, July 23-27, 2005, Proceedings, volume 3581 of *Lecture Notes in Computer Science*. Springer, 2005. (Cited on page 253.)

[Mili 1986] Ali Mili, Jules Desharnais and Jean Raynomd Gagné. *Formal models of stepwise refinements of programs*. ACM Comput. Surv., vol. 18, pages 231–276, September 1986. (Cited on page 63.)

[Miller 1985] Perry L. Miller. A critiquing approach to expert computer advice: Attending. Pitman Publishing, Inc., Marshfield, MA, USA, 1985. (Cited on page 252.)

[Miller 1995] S.P. Miller and M. Srivas. *Formal verification of the AAMP5 microprocessor: a case study in the industrial use of formal methods*. In Industrial-Strength Formal Specification Techniques, 1995. Proceedings., Workshop on, pages 2 –16, apr 1995. (Cited on page 41.)

[Miller 1998] Steven P. Miller. *Specifying the mode logic of a flight guidance system in CoRE and SCR*. In FMSP '98: Proceedings of the second workshop on Formal methods in software practice, pages 44–53, New York, NY, USA, 1998. ACM. (Cited on pages 64, 87 and 88.)

[Miller 1999] Donald W. Miller, Sandra J. Frawley and Perry L. Miller. *Using semantic constraints to help verify the completeness of a computer-based clinical guideline for childhood immunization*. Computer methods and programs in biomedicine, vol. 58, no. 3, pages 267–280, 1999. (Cited on pages 252 and 253.)

[Milner 1982] R. Milner. A calculus of communicating systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982. (Cited on page 40.)

[MIT-BIH] MIT-BIH. *MIT-BIH Database Distribution and Software :*. http://ecg.mit.edu/index.html. (Cited on page 206.)

[Moore 1997] P.R. DeWeese Moore J.W. and D. Rilling. *U. S. Software Life Cycle Process Standards*. Crosstalk: The DoD Journal of Software Engineering, 1997. (Cited on page 63.)

[Moreau 2001] L. Moreau and J. Duprat. *A Construction of Distributed Reference Counting*. Acta Informatica, vol. 37, pages 563–595, 2001. (Cited on page 51.)

[Morgan 1990] C. Morgan. Programming from specifications. Prentice Hall International Series in Computer Science. Prentice Hall, 1990. (Cited on page 50.)

[NASA Technical Team 2004] NASA Technical Team. *Nasa Software Safety Guidebook*. Rapport technique, Nasa Technical Standard, 2004. (Cited on page 39.)

[Neumann 1966] John von Neumann. *Theory of Self-Reproducing Automata. University of Illinois Press, Edited by Arthur W. Burks*. 1966. (Cited on pages 47, 129 and 138.)

[Neumann 1995] Peter Neumann. *Safeware: System Safety and Computers*. SIGSOFT Softw. Eng. Notes, vol. 20, pages 90–91, December 1995. (Cited on page 38.)

[NITRD 2009] NITRD. *High Confidence Software and Systems Coordinating Group, High-Confidence Medical Devices : Cyber-Physical Systems for 21st Century Health Care*. Rapport technique, NITRD, 2009. http://www.nitrd.gov/About/MedDevice-FINAL1-web.pdf. (Cited on pages 7, 12, 15, 26, 62, 64, 65, 72, 84, 128, 154, 207, 213, 281 and 283.)

[Overture] Overture. *Overture: Formal Modelling in VDM*. http://www.overturetool.org/. (Cited on pages 43, 44, 62 and 97.)

[Parnas 1972] D. L. Parnas. *On the criteria to be used in decomposing systems into modules*. Commun. ACM, vol. 15, pages 1053–1058, December 1972. (Cited on page 35.)

[Paulk 1995] M.C. Paulk. The capability maturity model: guidelines for improving the software process. The SEI series in software engineering. Addison-Wesley Pub. Co., 1995. (Cited on page 34.)

[Pearce 2007] David J. Pearce, Paul H.J. Kelly and Chris Hankin. *Efficient field-sensitive pointer analysis of C*. ACM Trans. Program. Lang. Syst., vol. 30, no. 1, page 4, 2007. (Cited on page 95.)

[Peleg 2003] Mor Peleg, Samson Tu, Jonathan Bury, Paolo Ciccarese, John Fox, Robert A Greenes, Silvia Miksch, Silvana Quaglini, Andreas Seyfang, Edward H Shortliffe, Mario Stefanelli and et al. *Comparing Computer-Interpretable Guideline Models: A Case-Study Approach*. Journal of the American Medical Informatics Association, vol. 10, page 2003, 2003. (Cited on page 251.)

[Pérez 2010] Beatriz Pérez and Ivan Porres. *Authoring and verification of clinical guidelines: A model driven approach*. Journal of Biomedical Informatics, vol. 43, no. 4, pages 520 – 536, 2010. (Cited on page 252.)

[Platzer 2008] André Platzer and Jan-David Quesel. *KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description)*. In The International Joint Conference on Automated Reasoning (IJCAR), pages 171–178. Springer, 2008. (Cited on page 218.)

[Plonsey 1987] Robert Plonsey and Roger C. Barr. *Mathematical modeling of electrical activity of the heart*. Journal of Electrocardiology, vol. 20, no. 3, pages 219 – 226, 1987. (Cited on page 129.)

[Pohl 2009] Carlos Paiz Pohl Christopher and Mario Porrmann. *vMAGIC-Automatic Code Generation for VHDL*. In International Journal of Reconfigurable Computing, 2009. (Cited on page 97.)

[Ponsard 2005] C. Ponsard, P. Massonet, A. Rifaut, J.F. Molderez, A. van Lamsweerde and H. Tran Van. *Early Verification and Validation of Mission Critical Systems*. Electronic Notes in Theoretical Computer Science, vol. 133, pages 237 – 254, 2005. Proceedings of the Ninth International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2004). (Cited on page 74.)

[Price 1995] D. Price. *Pentium FDIV flaw-lessons learned*. Micro, IEEE, vol. 15, no. 2, pages 86 –88, apr 1995. (Cited on pages 2, 21 and 32.)

[ProB] ProB. *The ProB Animator and Model Checker for the B Method*. http://www.stups.uni-duesseldorf.de/ProB/overview.php/. (Cited on pages 6, 26, 68, 129, 153, 162, 200, 238 and 239.)

[Ramesh 2001] Balasubramaniam Ramesh and Matthias Jarke. *Toward Reference Models for Requirements Traceability*. IEEE Trans. Softw. Eng., vol. 27, pages 58–93, January 2001. (Cited on page 77.)

[Real 2004] Jorge Real and Alfons Crespo. *Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal*. Real-Time Syst., vol. 26, no. 2, pages 161–197, 2004. (Cited on page 88.)

[Rehm 2010] Joris Rehm. *Proved development of the real-time properties of the IEEE 1394 Root Contention Protocol with the event-B method*. International Journal on Software Tools for Technology Transfer (STTT), vol. 12, fevrier 2010. (Cited on page 168.)

[Reinhardt 2007] Robert Reinhardt and Snow Dowd. Adobe Flash CS3 Professional Bible, page 1232. Wiley, 2007. ISBN-9780470119372. (Cited on page 82.)

[Reisig 2007] Wolfgang Reisig. Abstract state machines for the classroom, pages 1–32. Springer, 2007. See [Bjørner 2007]. (Cited on page 52.)

[Richter 2006] Jeffrey Richter. CLR Via C#, Second Edition. Microsoft Press, Redmond, 2006. (Cited on page 95.)

[Rittel 1973] Horst W. J. Rittel and Melvin M. Webber. *Dilemmas in a general theory of planning*. Policy Sciences, vol. 4, no. 2, pages 155–169, June 1973. (Cited on page 75.)

[RODIN 2004] RODIN. *Rigorous Open Development Environment for Complex Systems*. http://rodin-b-sharp.sourceforge.net, 2004. (Cited on pages 6, 14, 26, 49, 52, 53, 55, 57, 83, 96, 98, 99, 100, 102, 104, 129, 141, 155, 202, 251, 274 and 282.)

[Royce 1987] W. W. Royce. *Managing the development of large software systems: concepts and techniques*. In Proceedings of the 9th international conference on Software Engineering, ICSE '87, pages 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press. (Cited on page 63.)

[RTCA 1992] RTCA. *Do-178B, Software Considerations in Airborne Systems and Equipment Certification, Committee: SC-167*. http://www.rtca.org/, 1992. (Cited on pages 33 and 290.)

[Rumbaugh 1999] James Rumbaugh, Ivar Jacobson and Grady Booch, Editors. The unified modeling language reference manual. Addison-Wesley Longman Ltd., Essex, UK, UK, 1999. (Cited on pages 97 and 252.)

[Rushby 1995] John Rushby. *Formal Methods and their Role in the Certification of Critical Systems*. Rapport technique, Safety and Reliability of Software Based Systems (Twelfth Annual CSR Workshop), 1995. (Cited on pages 31, 40, 43 and 64.)

[Samson 1996] Mark Musen Samson, Mark A. Musen, Samson W. Tu, Amar K. Das and Yuval Shahar. *EON: A Component-Based Approach to Automation of Protocol-Directed Therapy*, 1996. (Cited on page 251.)

[Scacchi 1984] Walt Scacchi. *Managing Software Engineering Projects: A Social Analysis*. Software Engineering, IEEE Transactions on, vol. SE-10, no. 1, pages 49 –59, jan. 1984. (Cited on page 63.)

[Schmid 2000] Reto Schmid, Johannes Ryser, Stefan Berner, Martin Glinz, Ralf Reutemann and Erwin Fahr. *A Survey of Simulation Tools for Requirements Engineering*. Rapport technique, 2000. (Cited on page 79.)

[Schmitt 2006] Jonathan Schmitt, Alwin Hoffmann, Michael Balser, Wolfgang Reif and Mar Marcos. *Interactive Verification of Medical Guidelines*. In Jayadev Misra,

Tobias Nipkow and Emil Sekerinski, Editors, FM 2006: Formal Methods, volume 4085 of *Lecture Notes in Computer Science*, pages 32–47. Springer Berlin / Heidelberg, 2006. 10.1007/11813040_3. (Cited on page 252.)

[Schumann 2001] J.M. Schumann. Automated theorem proving in software engineering. Springer, 2001. (Cited on pages 4, 23, 35, 65 and 71.)

[Scientific 2007] Boston Scientific. *Pacemaker system specification, Technical report*. http://www.cas.mcmaster.ca/sqrl/ SQRLDocuments/PACEMAKER.pdf, 2007. (Cited on pages 84, 162, 166, 172 and 200.)

[Sebesta 2009] Robert W. Sebesta. Concepts of programming languages. Addison-Wesley Publishing Company, USA, 9th edition, 2009. (Cited on page 104.)

[Sekerinski 1998] E. Sekerinski and K. Sere, Editors. Program Development by Refinement - Cases Studies Using the B Method. FACIT. Springer, 1998. (Cited on pages 51 and 304.)

[Servat 2006] Thierry Servat. Brama: A new graphic animation tool for b models, pages 274–276. LNCS. Springer, 2006. (Cited on pages 74 and 83.)

[Seyfang 2006] Andreas Seyfang, Silvia Miksch, Mar Marcos, Jolanda Wittenberg, Cristina Polo-Conde and Kitty Rosenbrand. *Bridging the Gap between Informal and Formal Guideline Representations*. In Proceeding of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29 – September 1, 2006, Riva del Garda, Italy, pages 447–451, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press. (Cited on page 253.)

[Shahar 1998] Yuval Shahar, Silvia Miksch and Peter Johnson. *The ASGAARD Project: A Task-Specific Framework for the Application and Critiquing of Time-Oriented Clinical Guidelines*. In Artificial Intelligence in Medicine, pages 29–51, 1998. (Cited on page 251.)

[Shiffman 1994] Richard N. Shiffman and Robert A. Greenes. *Improving Clinical Guidelines with Logic and Decision-table Techniques: Application to Hepatitis Immunization Recommendations*. Med Decis Making, vol. 14, no. 3, pages 245–254, 1994. (Cited on page 252.)

[Shiffman 1997] Richard N Shiffman. *Representation of Clinical Practice Guidelines in Conventional and Augmented Decision Tables*. Journal of the American Medical Informatics Association, vol. 4, no. 5, pages 382–393, 1997. (Cited on page 252.)

[Siddiqi 1997] Jawed I. Siddiqi, Ian C. Morrey, Chris R. Roast and Mehmet B. Ozcan. *Towards quality requirements via animated formal specifications*. Ann. Softw. Eng., vol. 3, pages 131–155, January 1997. (Cited on page 78.)

[Sites 1974] Richard L. Sites. *Clean Termination of Computer Programs*. Ph.D. dissertation, Stanford University, Stanford, California, June 1974. (Cited on pages 100 and 126.)

[Smith 2001] Jeffrey E. Smith, Mieczyslaw K. Kokar and Kenneth Baclawski. *Formal Verification of UML Diagrams: A First Step Towards Code Generation*. In Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists, pages 224–240. GI, 2001. (Cited on page 97.)

[Smith 2008] Douglas Smith. *Generating Programs Plus Proofs by Refinement*. In Bertrand Meyer and Jim Woodcock, Editors, Verified Software: Theories, Tools, Experiments, volume 4171 of *Lecture Notes in Computer Science*, pages 182–188. Springer Berlin / Heidelberg, 2008. (Cited on pages 91 and 92.)

[Software 2001] Roy Miller Software, Roy W. Miller and Christopher T. Collins. *Acceptance Testing*, 2001. (Cited on page 70.)

[Sommerville 1995] Ian Sommerville. Software engineering (5th ed.). Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1995. (Cited on page 63.)

[Spivey 1987] J. M. Spivey. Understanding Z: a specification language and its formal semantics. 1987. (Cited on page 51.)

[Stärk 1998] R. Stärk, J. Schmid and E. Börger. Java and the Java Virtual Machine. Springer, 1998. (Cited on page 52.)

[Stepney 2000] Susan Stepney, David Cooper and Jim Woodcock. *An Electronic Purse: Specification, Refinement, and Proof*. Technical monograph PRG-126, Oxford University Computing Laboratory Programming Research Group, July 2000. (Cited on page 42.)

[Stroustrup 1994] Bjarne Stroustrup. The C++ Programming Language, Third Edition. Addison-Wesley, 1994. ISBN 0-201-88954-4. (Cited on pages 95 and 103.)

[Ten Teije 2006] Annette Ten Teije, Mar Marcos, Michel Balser, Joyce van Croonenborg, Christoph Duelli, Frank van Harmelen, Peter Lucas, Silvia Miksch, Wolfgang Reif, Kitty Rosenbrand and Andreas Seyfang. *Improving medical protocols by formal methods*. Artif. Intell. Med., vol. 36, no. 3, pages 193–209, 2006. (Cited on pages 250, 251 and 253.)

[Thomas 1993] Martyn Thomas. *The industrial use of formal methods*. Microprocess. Microsyst., vol. 17, pages 31–36, January 1993. (Cited on page 43.)

[Trafford 1997] Paul J. Trafford. *The Use of Formal Methods for Safety-Critical System*. PhD thesis, Kingston University, 1997. (Cited on pages 35, 36, 38 and 43.)

[Tretmans 2001] Jan Tretmans, Klaas Wijbrans and Michel R. V. Chaudron. *Software Engineering with Formal Methods: The Development of a Storm Surge Barrier Control System Revisiting Seven Myths of Formal Methods*. Formal Methods in System Design, vol. 19, no. 2, pages 195–215, 2001. (Cited on page 42.)

[Tuan 2010] Luu Anh Tuan, Man Chun Zheng and Quan Thanh Tho. *Modeling and Verification of Safety Critical Systems: A Case Study on Pacemaker*. Secure System Integration and Reliability Improvement, pages 23–32, june 2010. (Cited on page 163.)

[Turner 1985] D. A. Turner. *Functional programs as executable specifications*. In Proc. of a discussion meeting of the Royal Society of London on Mathematical logic and programming languages, pages 29–54, Upper Saddle River, NJ, USA, 1985. Prentice-Hall, Inc. (Cited on page 78.)

[Van 2004] Hung Tran Van, Axel van Lamsweerde, Philippe Massonet and Christophe Ponsard. *Goal-Oriented Requirements Animation*. In Proceedings of the Requirements Engineering Conference, 12th IEEE International, pages 218–228, Washington, DC, USA, 2004. IEEE Computer Society. (Cited on pages 74 and 78.)

[Vangheluwe 2000] Hans Vangheluwe and Ghislain C. Vansteenkiste. *The cellular automata formalism and its relationship to DEVS*. In Proceedings of the 14th European Simulation Multiconference on Simulation and Modelling: Enablers for a Better Quality of Life, pages 800–810. SCS Europe, 2000. (Cited on page 138.)

[Walters 1990] H. Walters. *Hybrid implementations of algebraic specifications*. In Hélène Kirchner and Wolfgang Wechler, Editors, Algebraic and Logic Programming, volume 463 of *Lecture Notes in Computer Science*, pages 40–54. Springer Berlin / Heidelberg, 1990. (Cited on pages 91 and 92.)

[Wang 2002] Dongwen Wang, Mor Peleg, Samson W. Tu, Aziz A. Boxwala, Robert A. Greenes, Vimla L. Patel and Edward H. Shortliffe. *Representation primitives, process models and patient data in computer-interpretable clinical practice guidelines:: A literature review of guideline representation models*. International Journal of Medical Informatics, vol. 68, no. 1-3, pages 59 – 70, 2002. (Cited on page 251.)

[Warmer 2003] Jos Warmer and Anneke Kleppe. The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2003. (Cited on page 252.)

[Wichmann 1992] B.A. Wichmann and British Computer Society. Software in safety-related systems. Special report. BCS, 1992. (Cited on pages 4, 23, 35, 37, 46, 65 and 71.)

[Wirth 1971] Niklaus Wirth. *Program development by stepwise refinement*. Commun. ACM, vol. 14, pages 221–227, April 1971. (Cited on page 63.)

[Woodcock 2006] Jim Woodcock. *First Steps in the Verified Software Grand Challenge*. IEEE Computer, vol. 39, no. 10, pages 57–64, 2006. (Cited on pages 84, 159, 161 and 162.)

[Woodcock 2007] J. Woodcock and R. Banach. *The Verification Grand Challenge*. Journal of Universal Computer Science, vol. 13, no. 5, pages 661–668, 2007. (Cited on pages 2, 7, 22, 26, 31, 84 and 162.)

[Woodcock 2008] Jim Woodcock, Susan Stepney, David Cooper, John A. Clark and Jeremy Jacob. *The certification of the Mondex electronic purse to ITSEC Level E6*. Formal Asp. Comput., vol. 20, no. 1, pages 5–19, 2008. (Cited on page 42.)

[Woodcock 2009] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui and John Fitzgerald. *Formal methods: Practice and experience*. ACM Computing Surveys, vol. 41, pages 19:1–19:36, October 2009. (Cited on pages 31 and 61.)

[Wright 2009] Steve Wright. *Automatic Generation of C from Event-B*. In Workshop on Integration of Model-based Formal Methods and Tools IM_FMT'2009 - in IFM'2009, Düsseldorf, Germany., 2009. (Cited on page 98.)

[Writing Committee 2008] Members Writing Committee, Andrew E. Epstein, John P. DiMarco, Kenneth A. Ellenbogen, III Estes N.A. Mark, Roger A. Freedman, Leonard S. Gettes, A. Marc Gillinov, Gabriel Gregoratos, Stephen C. Hammill, David L. Hayes, Mark A. Hlatky, L. Kristin Newby, Richard L. Page, Mark H. Schoenfeld, Michael J. Silka, Lynne Warner Stevenson and Michael O. Sweeney. *ACC/AHA/HRS 2008 Guidelines for Device-Based Therapy of Cardiac Rhythm Abnormalities: Executive Summary: A Report of the American College of Cardiology/American Heart Association Task Force on Practice Guidelines (Writing Committee to Revise the ACC/AHA/NASPE 2002 Guideline Update for Implantation of Cardiac Pacemakers and Antiarrhythmia Devices): Developed in Collaboration With the American Association for Thoracic Surgery and Society of Thoracic Surgeons*. Circulation, vol. 117, no. 21, pages 2820–2840, 2008. (Cited on pages 19, 166, 255, 256 and 287.)

[Xu 2011] Hao Xu and Tom Maibaum. *Generic Insulin Infusion Pump*. In International Symposium on Foundations of Health Information Engineering and Systems (FHIES'11). 2011. (Cited on page 32.)

[Yeo 2002] K. T. Yeo. *Critical failure factors in information system projects*. International Journal of Project Management, vol. 20, no. 3, pages 241–246, April 2002. (Cited on pages 2 and 21.)

[Zhang 2010] Yi Zhang, Paul L Jones and Raoul Jetley. *A Hazard Analysis for a Generic Insulin Infusion Pump*. Journal of diabetes science and technology Online, vol. 4, no. 2, pages 263–283, 2010. (Cited on pages 2, 21 and 32.)