Chapter 1
Introduction

Abstract The primary goal of this book is to advance the use of formal techniques for the development of computing systems with high integrity. Specifically, the book makes an analysis of critical system software that the formal methods are not well integrated into established critical systems development processes. This book presents formalism for a new 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 using several intermediate layers with rigorous safety assessment approach. The approach has been verified using the Event-B formalism. The efficacy of formalism has been evaluated through a "Grand Challenge" case study, relative to the development of a cardiac pacemaker.

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 [10, 16, 32, 38, 39] 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 be 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, etc. Additionally, and most significantly potential loss is customer trust for a product failure. In this context, a high degree of safety

N.K. Singh, *Using Event-B for Critical Device Software Systems*,
DOI 10.1007/978-1-4471-5260-6_1,

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 behaviour 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 [9, 14, 37] is a standard and popular approach to deal with the increasing complexity of a system with assurance of correctness in the modern software engineering practices. 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 standard certificates [6, 8, 12, 13] 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 an actual system. This approach helps to correct errors more easily and with less cost. We formulate the following objectives related to a critical system development:

- Establishing a unified theory for the critical systems development.
- Building a comprehensive and integrated suite of tools for the critical systems that can support verification activities, including formal specification, model validation, real-time animation and automatic code generation.
- Environment modelling for the development of a closed-loop system 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.
- 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 book 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. A development methodology including a set of techniques and tools is developed for handling the stakeholders 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 to implement a system.

The formal verification and model validation offers to meet the challenge of complying with FDA's QSR, ISO, IEEE, CC [6, 8, 12, 13] 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" [11, 21]. All the proposed approaches may also help to obtain the certification standards [6, 8, 12, 13] in the area of critical system development.

1.2 Approach

In this book, we present a development life-cycle methodology, a framework for real-time animator [19], refinement chart [31], a set of automatic code generation tools [7, 18, 22] and formal logic based heart model for closed-loop modelling [25, 29]. 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 behaviour at the concrete level. Our approach of specification and verification is based on the techniques of abstraction and refinement. Introducing a new set of tools helps to verify the 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 [19], which also provides a way for domain experts (i.e. medical experts) to participate in the system development process (medical device development). A basic description about development methodology and all associated techniques and tools are provided in the following paragraphs:

We propose a new methodology, which is an extension of the waterfall model [3, 5, 34, 35] and utilises rigorous approaches based on formal techniques 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 the safety assessment approaches [15, 33, 36] to comply with certification standards [6, 8, 12, 13]. 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 formalised using a formal modelling 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 visualisation 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 approach is 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, we emphasise the requirements traceability using a real-time animator [19]. Formal modelling 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 recognised to be a promising approach to support the process of validation of requirement's specification. The principle is to simulate the desired behaviours of a given system using formal models in the real-time environment and to visualise the simulation in some form which appeals to stakeholders. The real-time environment assists in the construction, clarification, validation and visualisation of a formal specification. Such an approach is also useful for evidence-based certification.

Refinement techniques [1, 2, 4] serve a key role for modelling 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 behaviour of the system. The final goal of using this refinement chart is to obtain a specification that is detailed enough to be effectively implemented, but also to correctly describe the system behaviours. 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 [7, 18, 22, 23] 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 the early stages of system development, which is an open problem in the area of modelling. The cardiac pacemaker and implantable cardioverter-defibrillators (ICDs) are key critical medical devices, which require closed-loop modelling (integration of system and environment modelling) 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 [27]. 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 the virtual biological environment [27].

Assessment of the proposed framework, and techniques and tools are scrutinised through the development of a cardiac pacemaker. The cardiac pacemaker is a pilot project of the international "Grand Challenge". This book covers a complete development process of a cardiac pacemaker using the proposed life-cycle framework and developed tools [17, 20, 24] from requirements analysis to code implementation.

Formal techniques are useful not only for critical-systems, but it can be used to verify required safety properties in other domains, for example, in the clinical domain to verify the correctness of protocols and guidelines [26, 27, 30]. 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 the 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 modelling 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) that covers a wide variety of protocol characteristics related to several heart diseases.

1.2.1 Outline

The book is structured in 11 chapters. Chapter 2 presents a basic background and development life-cycle related to the safety critical systems. Chapter 3 describes modelling techniques using the Event-B modelling language. In Chap. 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 Chap. 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 [19]. This architecture can be used for requirement traceability. In Chap. 6, the refinement chart is proposed to handle the complexity and for designing the critical systems. In Chap. 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 [7, 18, 22, 23, 28]. In Chap. 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 system. Chapter 9 presents a complete formal development of a cardiac pacemaker using proposed techniques and tools from requirements analysis to automatic code generation. In Chap. 10, 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 11 summarises this book.

References

1. Abrial, J.-R. (1996). *The B-book: Assigning programs to meanings*. New York: Cambridge University Press.
2. Abrial, J.-R. (2010). *Modeling in Event-B: System and software engineering* (1st ed.). New York: Cambridge University Press.
3. Acuña, S. T., & Juristo, N. (2005). *International series in software engineering. Software process modeling*. New York: Springer.
4. Back, R. J. R. (1981). On correct refinement of programs. *Journal of Computer and System Sciences*, *23*(1), 49–68.
5. Bell, R., & Reinert, D. (1993). Risk and system integrity concepts for safety-related control systems. *Microprocessors and Microsystems*, *17*, 3–15.
6. CC. Common criteria. http://www.commoncriteriaportal.org/.
7. EB2ALL (2011). Automatic code generation from Event-B to many programming languages. http://eb2all.loria.fr/.
8. FDA. Food and Drug Administration. http://www.fda.gov/.
9. Gaudel, M.-C., & Woodcock, J. (Eds.) (1996). *Lecture notes in computer science: Vol. 1051. Proceedings, FME'96: Industrial benefit and advances in formal methods*. Third international symposium of formal methods Europe, co-sponsored by IFIP WG 14.3, Oxford, March 18–22, 1996. Berlin: Springer.
10. Gibbs, W. W. (1994). Software's chronic crisis. *Scientific American*, September.
11. High Confidence Software and Systems Coordinating Group (2009). *High-confidence medical devices: Cyber-physical systems for 21st century health care* (Technical report). NITRD. http://www.nitrd.gov/About/MedDevice-FINAL1-web.pdf.
12. IEEE-SA. IEEE Standards Association. http://standards.ieee.org/.
13. ISO. International Organization for Standardization. http://www.iso.org/.
14. Jetley, R., Purushothaman Iyer, S., & Jones, P. (2006). A formal methods approach to medical device review. *Computer*, *39*(4), 61–67.
15. Leveson, N. G. (1991). Software safety in embedded computer systems. *Communications of the ACM*, *34*, 34–46.
16. Leveson, N. G., & Turner, C. S. (1993). An investigation of the Therac-25 accidents. *Computer*, *26*, 18–41.
17. Méry, D., & Singh, N. K. (2009). *Pacemaker's functional behaviors in Event-B* (Research report). MOSEL-LORIA-INRIA-CNRS: UMR7503-Université Henri Poincaré-

Nancy I-Université Nancy II-Institut National Polytechnique de Lorraine. http://hal.inria.fr/inria-00419973/en/.
18. Méry, D., & Singh, N. K. (2010). *EB2C: A tool for Event-B to C conversion support*. Poster and tool demo submission, published in a CNR technical report in SEFM.
19. Méry, D., & Singh, N. K. (2010). Real-time animation for formal specification. In M. Aiguier, F. Bretaudeau, & D. Krob (Eds.), *Complex systems design & management* (pp. 49–60). Berlin: Springer.
20. Méry, D., & Singh, N. K. (2010). Technical report on formal development of two-electrode cardiac pacing system. MOSEL-LORIA-INRIA-CNRS: UMR7503-Université Henri Poincaré-Nancy I-Université Nancy II-Institut National Polytechnique de Lorraine. http://hal.archives-ouvertes.fr/inria-00465061/en/.
21. Méry, D., & Singh, N. K. (2010). Trustable formal specification for software certification. In T. Margaria & B. Steffen (Eds.), *Lecture notes in computer science: Vol. 6416. Leveraging applications of formal methods, verification, and validation* (pp. 312–326). Berlin: Springer.
22. Méry, D., & Singh, N. K. (2011). Automatic code generation from Event-B models. In *Proceedings of the second symposium on information and communication technology*, SoICT'11 (pp. 179–188). New York: ACM.
23. Méry, D., & Singh, N. K. (2011). *EB2J: Code generation from Event-B to Java*. Short paper presented at the 14th Brazilian symposium on formal methods, SBMF'11.
24. Méry, D., & Singh, N. K. (2011). Functional behavior of a cardiac pacing system. *International Journal of Discrete Event Control Systems*, *1*(2), 129–149.
25. Méry, D., & Singh, N. K. (2011). Technical report on formalisation of the heart using analysis of conduction time and velocity of the electrocardiography and cellular-automata. MOSEL-LORIA-INRIA-CNRS: UMR7503-Université Henri Poincaré-Nancy I-Université Nancy II-Institut National Polytechnique de Lorraine. http://hal.inria.fr/inria-00600339/en/.
26. Méry, D., & Singh, N. K. (2011). Technical report on interpretation of the electrocardiogram (ECG) signal using formal methods. MOSEL-LORIA-INRIA-CNRS: UMR7503-Université Henri Poincaré-Nancy I-Université Nancy II-Institut National Polytechnique de Lorraine. http://hal.inria.fr/inria-00584177/en/.
27. Méry, D., & Singh, N. K. (2012). Closed-loop modeling of cardiac pacemaker and heart. In *Foundations of health informatics engineering and systems*.
28. Méry, D., & Singh, N. K. (2012). *Formal development and automatic code generation: Cardiac pacemaker*. New York: ASME Press.
29. Méry, D., & Singh, N. K. (2012). Formalization of heart models based on the conduction of electrical impulses and cellular automata. In Z. Liu & A. Wassyng (Eds.), *Lecture notes in computer science: Vol. 7151. Foundations of health informatics engineering and systems* (pp. 140–159). Berlin: Springer.
30. Méry, D., & Singh, N. K. (2012). Medical protocol diagnosis using formal methods. In Z. Liu & A. Wassyng (Eds.), *Lecture notes in computer science: Vol. 7151. Foundations of health informatics engineering and systems* (pp. 1–20). Berlin: Springer.
31. Méry, D., & Singh, N. K. (2013). Formal specification of medical systems by proof-based refinement. *ACM Transactions on Embedded Computing Systems*, *12*(1), 15:1–15:25.
32. Price, D. (1995). Pentium FDIV flaw-lessons learned. *IEEE MICRO*, *15*(2), 86–88.
33. Redmill, M. C. F., & Catmur, J. (1999). *System safety: HAZOP and software HAZOP* (1st ed.). Chichester: Wiley.
34. Schumann, J. M. (2001). *Automated theorem proving in software engineering*. New York: Springer.
35. Wichmann, B. A., & British Computer Society (1992). *Software in safety-related systems* (Special report). BCS.
36. Wilkinson, P. J., & Kelly, T. P. (1998). Functional hazard analysis for highly integrated aerospace systems. In *Certification of ground/air systems seminar* (pp. 4–146). New York: IEEE. Ref. No. 1998/255.
37. Woodcock, J., & Banach, R. (2007). The verification grand challenge. *Journal of Universal Computer Science*, *13*(5), 661–668.

38. Yeo, K. T. (2002). Critical failure factors in information system projects. *International Journal of Project Management*, *20*(3), 241–246.
39. Zhang, Y., Jones, P. L., & Jetley, R. (2010). A hazard analysis for a generic insulin infusion pump. *Journal of Diabetes Science and Technology*, *4*(2), 263–283.