Rigorous Safety-Critical Cyber-Physical Systems Development using Formal Methods

Neeraj Kumar Singh

HDR

Mémoire présenté en vue de l'obtention du

HABILITATION À DIRIGER DES RECHERCHES

Délivré par : *l'Institut National Polytechnique de Toulouse (INP Toulouse)*

Présentée et soutenue le *21/06/2024* **par :**

Neeraj Kumar Singh

Rigorous Safety-Critical Cyber-Physical Systems Development using Formal Methods

---

JURY

Michael Butler	Professeur, University of Southampton	Rapporteur
Laurent Fribourg	Directeur de recherche, ENS Paris-Saclay	Rapporteur
Jean-Pierre Talpin	Directeur de recherche, INRIA – Rennes	Rapporteur
Yamine Ait-Ameur	Professeur, Toulouse INP/IRIT	Examinateur
Saddek Bensalem	Professeur, Université Grenoble Alpes	Examinateur
Jin-Song Dong	Professeur, National University of Singapore	Examinateur
Stefania Gnesi	Researcher, ISTI-CNR – Italy	Examinatrice
Dominique Méry	Professeur, Université de Lorraine	Examinateur
Alan Wassyng	Professeur, McMaster University	Examinateur

Unité de Recherche :

Institut de Recherche en Informatique de Toulouse (UMR 5505)

Dedicated to Lord Krishna, my loving parents, family,
wife Arati, and my cherished little angel Chetna

Abstract

Today, we are surrounded by digital technologies and highly complex systems, where safety-critical cyber-physical systems have taken central place in our lives and in various industrial sectors to improve human lives and boost economies by solving key issues in a variety of domains, including defense, transportation, space, healthcare and biomedical, agriculture, manufacturing, smart grids and energy, and everyday living. With great utility, however, safety-critical cyber-physical systems brought very important issues in their development, particularly in system modelling, security and privacy, heterogeneity, composition, and certification, which could jeopardize our well-being as well as the development and reliability of cyber-physical systems. Our increased reliance on safety-critical cyber-physical systems also prompted us to consider the ethics of these systems and how future technologies might limit risks related to failure, safety, privacy, responsibility, liability, and other issues.

We argue that addressing some of these essential questions requires combining formal approaches with key domains like domain knowledge engineering, system modelling, and certification for rigorous formal engineering of safety-critical cyber-physical systems. Formal methods play a key role to model such complex systems correctly. Domain knowledge engineering is useful for specifying essential elements that can be used to develop domain models and establishing relationships between system models. System modelling enables the development of generic frameworks, modelling and designing patterns, formal theories and proofs, and implementation for addressing design challenges. Finally, certification methods aid in the certification of complex safety-critical cyber-physical systems and their components.

This manuscript synthesises our research efforts on the development and investigation of methods for dealing with formal engineering processes such as modelling, refinement and simulation, domain knowledge engineering, design automation, heterogeneity, composition, safety, and certification issues for safety-critical cyber-physical systems. Our first contribution focuses on domain knowledge engineering for dealing with their various core concepts. The next contribution focuses on system modelling, covering various approaches such as automatic refinement, modelling and designing framework, patterns and theories, reflexive meta-modelling, environment modelling and automatic code generation. The last contribution focuses on certification and the development of assurance cases. Finally, we deploy these approaches to the design of safety-critical cyber-physical systems from various domains. We conclude by describing the perspectives of our research, which include two main directions: (i) perspectives on theories, models, patterns, and tools, and (ii) perspectives on safety-critical cyber-physical systems.

Acknowledgements

I would like to express my deepest gratitude to everyone, who has interacted with me, whether directly or indirectly, played a key role in my academic career, or crossed my path, and helping me reach this point in my journey and realise my HDR.

First and foremost, I wish to thank the reviewers, Michael Butler, Laurent Fribourg, and Jean-Pierre Talpin, for their invaluable time and effort in carefully reading and providing feedback on my manuscript. I am also grateful to the jury members, Yamine Ait-Ameur, Saddek Bensalem, Jin-Song Dong, Stefania Gnesi, Dominique Méry, and Alan Wassyng, for their participation in my HDR defence. Their presence made the event a memorable, and I appreciate their questions, discussions, and insightful ideas that will guide my future research.

I would like to extend my sincere gratitude to the two individuals who played pivotal roles in the realisation of my HDR. Firstly, I wish to express my deep appreciation to Yamine Ait-Ameur for his unwavering support. His insightful discussions, encouragement, and fruitful research collaborations have been instrumental in shaping my work. His extensive knowledge and logical approach have been a continual source of inspiration, and his thought-provoking suggestions and explorations have been particularly beneficial to this study. I am also deeply grateful to Dominique Mery, my Ph.D. advisor, for his guidance, support, and motivation throughout my HDR journey. His leadership and mentorship have been instrumental in helping me grow both personally and professionally, and I am forever grateful for his expertise and wisdom. Additionally, I would like to thank Ferhat Khendek for providing valuable feedback on the initial draft of my HDR.

I extend my sincere thanks to Ana Cavalcanti, Andy Wellings and Jim Woodcock at the University of York, UK, for their warm welcome as a postdoctoral fellow and for providing the opportunity to work on an exciting EPSRC project. I also wish to extend my appreciation to Tom Maibaum, Alan Wassyng, and Mark Lawford from the McMaster University McSCert group for offering me a second postdoctoral opportunity and generously supported my research on software certification and safety cases with their unwavering guidance. Their mentorship played a significant contributed to my professional growth.

I would like to thank my colleagues on the ACADIE and APO teams for their outstanding support, especially Marc Pantel, Xavier Cregut, Benoît Morgan, Aurélie Hurault, Philippe Queinnec, Philippe Mauran, Olivier Cots, and Joseph Gergaud for their helpful discussions and assistance during my research and teaching. Furthermore, I wish to express my gratitude to the administrative staff at INP-ENSEEIHT and IRIT for their invaluable assistance with my administrative and scientific responsibilities.

I am profoundly grateful to my co-authors, without whom some of this effort would not be feasible. I would like to extend special thanks to the talented and exceptional doctoral students who worked under my co-supervision: Guillaume Dupont, Alexandra Halchin, Ismail Mendil, and Peter Riviere. It was an absolute pleasure collaborating with them, and I appreciate their exceptional skills, thoughtful approach, and sense of humour, which made our collaboration a truly enjoyable and productive experience. I also thank all the master's students and interns with whom I had the privilege to work. Thank you all for sharing your experiences with me.

The work presented in this document is the result of several collaborations with outstanding academic colleagues. I am grateful for the opportunity to work with such an exceptional group of researchers, including Christian Attiogbé, Sergiy Bogomolov, J. Duchêne, Flavio Ferrarotti, Mamoun Filali, Marc Frappier, Raju Halder, David Hewson, T. S. Hoang, Fuyuki Ishikawa, M. H. Khan, Tsutomu Kobayashi, P. K. Kalra, Regine Laleau, Michael Leuschel, Rajeev Pandey, Philippe Palanque, Ramesh S., Manoranjan Satpathy, Klaus-Dieter Schewe, Hichem Snoussi, Paulius Stankaitis, Kanishka Tayagi, Hao Wang, and many others.

Finally, I would like to extend my deepest gratitude to my loving parents, brothers, and wonderful family for their unwavering support and encouragement throughout my journey. I am deeply grateful for the constant motivation and inspiration I have received from my friends, who have been true sources of motivation and inspiration. Special thanks to Chantal Morand, Srikanth Christophe, and Briand's family for their kindness and support in Toulouse. Above all, I would like to give special thanks to my wonderful wife, Arati, and our beautiful daughter, Chetna, for bringing so much joy and love into my life. Thank you all for being an important part of my journey.

Contents

CHAPTER 1

Introduction

1.1 Context and motivation

Today, we are surrounded by digital technologies and highly complex systems, such as cyber-physical systems (CPS), that play an important role in our lives and economies. Cyber-physical systems integrate cyber systems (computational systems such as microprocessors and digital communication networks) with other physical systems (electromechanical, chemical, structural, and biological systems) [Lee 2008, Lee 2016]. There are several CPS examples, including driver-less trains, smart buildings, household appliances, medical devices, and everyday items such as cleaning robots, wearable bands, and electric rollers and bikes. Cyber-physical systems enable us to improve our quality of life in a variety of domains, including defence, transportation, space, healthcare and biomedical, agriculture, manufacturing, smart grids and energy, and everyday living.

As previously stated, cyber-physical systems are used in our daily lives to improve our quality of life, so there is an increasing demand for new technology, which forces the rapid adoption of commercial firmware and software for them. Rapid adoption of CPS increases vulnerabilities, which could lead to catastrophic system failures. Failure of these systems could result in loss of life, as well as reputation and financial damage. For example, in the medical domain, the USA Food and Drug Administration (FDA) has issued several recalls for cardiac pacemaker and implantable cardioverter-defibrillator (ICD). These recalls have resulted in a large number of serious illnesses and deaths. According to the FDA report, between 1900 and 2002, 17,323 devices (8834 pacemakers and 8489 ICDs) were ex-planted and 61 deaths (30 Pacemaker patients, 31 ICD patients) were reported due to erroneous behaviour. Similarly, the FDA has issued a number of recalls in which insulin infusion pump (IIP) failures have been linked to a large number of serious illnesses and deaths. According to the FDA, 17000 adverse events were reported between 2006 and 2009, including 41 deaths caused by faulty IIPs. The FDA determined that the deaths and adverse events associated with cardiac pacemakers, ICDs and IIPs were caused by product design and engineering flaws, including firmware issues [Maisel 2006, Chen 2014][1][98]. Similarly, in the transportation domain, car accidents claim thousands of lives each year and cause permanent disabilities, resulting in annual costs in the billions of dollars in the United States alone [Zaloshnja 2004]. Although human error is responsible for the majority of these car accidents, failures in hardware or software components can cause accidents and endanger human life [Peters 2003][15].

Traditional CPS development employs effective methods and tools for designing both computation and physical systems borrowed from other domains such as embedded systems, the internet of things, autonomous vehicles, and so on. In fact, system designers used ad hoc approaches to design CPS, with system designers having a hazy notion of requirements based on physical environment that were encoded in computation algorithms. There were no design patterns that could be used to develop other CPS. CPS development has recently enabled us to create new systems with complex dynamics and high reliability by adhering to some standard guidelines and designing patterns that support the basic foundation for CPS, specifically in dealing with the complex nature of computational and physical systems. In particular, Model-based System Engineering [Dori 2016], DevOps [Hegedus 2021] , V-model [Gr´aßler 2021], Scrum [Wagner 2014] are most common development life-cycles that adapted by industries for designing complex cyber-physical systems.

Over the past fifty years, formal techniques have shown some promising results in several domains, including healthcare, automotive, avionic and nuclear by identifying possible errors through formal reasoning. The formal reasoning has great impact in developing the system requirements or checking the correctness of functional requirements. In the current industrial practices, formal methods have been used to meet the standard requirements or certification requirements. For example, ISO 26262 [Organization 2011] standard has adopted the formal methods to design a passenger vehicle, particularly to meet safety requirements of Automotive Safety Integrity Level (ASIL) D. Validation of requirements specification is an integral and essential part of the requirements engineering. Validation

is a process of checking, together with stakeholders, whether the requirements specification meets its stakeholders' intentions and expectations [McDermid 1991]. Similarly, avionic standards DO-178B [RTCA/DO-178B 1992] and DO-278A [DO-278 011] permitted formal methods without addressing specific process requirements; however, DO-178C [RTCA/DO-178C 011] is accompanied by a new Radio Technical Commission for Aeronautics (RTCA) Guideline DO-333 [RTCA/DO-333 011] "Formal methods supplement to DO-178C and DO-278A" to meet certification objectives [Gigante 2012, Rushby 1993].

Formal methods have been used extensively in the development of cyber-physical systems to ensure the desired safe behaviour in accordance with the given requirements and operating environment. Academics and industry have used a variety of formalisms and rigorous techniques (e.g., VDM [Bjørner 1978, Jones 1986], Z [Spivey 1989], ASTRÉE [Cousot 2007], SCADE [Berry 2007], Event-B [Abrial 2010a], B Method [Abrial 1996], ASM [Börger 2003], CSP [Hoare 1985], Circus [Woodcock 2002], UNITY [Chandy 1989], PVS [Owre 1992], Coq [Bertot 2010], Isabelle/HOL [Nipkow 2002], Alloy [Jackson 2002], SPIN [Holzmann 1997], PAT [Sun 2009], NuSMV [Cimatti 2002], Uppaal [Bengtsson 1996]) in the development of CPS. Intuitively, formal methods have been used to identify precise and unambiguous requirements for developing complex critical cyber-physical systems, such as the ADGS-2100 Adaptive Display and Guidance Window Manager [Whalen 2005]; an airborne collision avoidance system (ACAS X) [Jeannin 2015]; the DARPA HACMS [Fisher 2017] program for developing high assurance software for vehicles, such as quadcopters, helicopters, and automobiles; machine-checked verification of the seL4 microkernel [Klein 2009]; Siemens transportation systems [Badeau 2005]; formal verification of avionics software products [Souyris 2009]; model checking techniques for avionics and automotive systems development [Miller 2010]; railway interlocking systems in Prover iLock [iLock]; formal verification of medical systems [Bowen 1993, Jetley 2004][1]; and space and avionic system [Rushby 1993, Butler 1996, Su 2014a][74]. Formal verification of industrial systems, on the other hand, is extremely difficult. Due to *limits of formalisation* and *practical limits* [Kneuper 1997], it can only be used to validate certain operations of the selected system.

The past several years of development experiences, including evidence, show that the separation of information science and physical science has resulted in a divergence in scientific foundations and technologies, which has become severely limiting to progress in the design of CPS [Lee 2008]. For example, system modelling and programming languages lack a mechanism to represent time and physical characteristics, aggregating all required physical design considerations. On the physical side, there is also a lack of computation and communication platform features (e.g., scheduling, network delays, and so on) that are required in the core of system development. Research communities of cyber and physical systems are working apart due to resulting barrier between them leads to work into isolated disciplines and resulting in compartmentalised design flows that lead to difficulties and failures of CPS due to growing complexity [Sztipanovits 2019].

Now we are on the edge of our knowledge of how to combine cyber and physical systems, as well as design a safe and secure CPS in various domains. If we continue to rely on existing methods and tools, soon we will be incapable to address all aspects of CPS and risk endangering our living communities, and the environment with unsafe and unpredictable systems. These shortcomings are related to technical constraints, the core foundation for developing different types of CPS in various domains, standard requirements, and certification. Due to their complexity and heterogeneity, CPS currently pose significant challenges in modelling, designing, and coordinating physical and cyber systems, domain engineering, safety, privacy, certification, and other areas. Some of them are discussed below.

Modelling and simulation

Modelling and simulation [DODD-5000.61 018] are fundamental activities in all engineering disciplines, including CPS. Modelling allows us to define a desired cyber-physical system by limiting the system's components and features that are relevant to the given goal, whereas simulation allows us to run a model to interpret and perform a series of tasks to examine the predicted behaviour of a cyber-physical system while taking into account various aspects of the physical system. The designed model can be improved during the CPS design process by including necessary details such as functionalities and properties. If the CPS design model is sufficiently accurate, the developed CPS model can be used to analyse dynamic behaviour and predict correct functionalities while eliminating unwanted behaviour, and

simulation can be used to analyse only specific aspects of CPS rather than the entire system, so it is never complete in the same way that testing techniques are [Fitzgerald 2016]. Note that testing techniques are not exhaustive, but they have many types of testing to uncover potential defects, but it cannot satisfy an absence of bug.

Different types of modelling approaches exist, such as *behaviour modelling*, which allows a system designer to model the behaviour of a CPS, and *architecture modelling*, which considers the architecture of a CPS. These modelling approaches play an important role during CPS design by allowing deep analysis on various system characteristics and assisting engineers in making the best optimum choices for selecting different hardware and software components while taking system constraints into account. The architecture modelling allows for the representation of key connections between various components of a cyber-physical system that play an important role in the organisation of software and hardware. Graphical modelling notations such as SysML [Friedenthal 2008], UML [Rumbaugh 2004], and others are the most commonly used approaches for describing complete CPS architecture. Note that such graphical notations can be used to illustrate the core organisation of the CPS as well as how the CPS components interact and share data. The behaviour modelling is essential for realising and predicting potential desired cyber-physical system behaviour. This modelling approach is both faster and less expensive than developing a physical prototype. As we all know, a cyber-physical system is made up of both cyber and physical systems, different modelling notations are required to represent them. To model physical systems and cyber systems, the two main approaches are *continuous-modelling* formalism and *discrete-modelling* formalism, respectively. The continuous modelling formalism represents physical processes using a set of differential equations, whereas the discrete modelling formalism uses automata to represent sequential behaviour and control flow. In fact, CPS can be modelled as a hybrid system because it has both continuous and discrete dynamics, and the continuous dynamics can switch depending on aperiodic internal or external discrete events. The required communication services of a CPS are modelled considering the network topology and communication protocol, in which the packets are transmitted in discrete time [Fitzgerald 2016].

Another important modelling challenge in CPS is dealing with real-time concepts. Continuous-time is essential for describing physical models and predicting physical activities accurately. All internal and external interactions and computation results in CPS are tightly coupled with time; if interaction activities and results are not computed in a timely manner, the system is rendered useless [Kopetz 2011]. Because there is no way to estimate how long an operation will take when an actual system executes, time passing phenomena is mostly abstractly specified in modelling. There are two sorts of CPS real-time performance requirements: *soft real-time system* and *hard real-time system*. The soft real-time system considers physical time for accuracy but does not have severe implications, whereas the hard real-time system does have serious effects if the provided real-time criteria do not meet. Hard real-time systems are always linked to physical CPS equipment, and any failure results in a catastrophic failure [Kopetz 2011].

Treating the various components of a CPS independently in modelling is a challenge for engineers to evaluate the trade-off between design options incorporating diverse configurations of both cyber and physical components. There is a crucial need for new modelling and simulation approaches that can be used to support a framework for unifying cyber and physical systems, as well as handle architecture, behaviour, and timing concepts of various classes of cyber-physical systems.

Domain engineering

CPS is made of cyber and physical systems. Most of modelling languages handle only cyber systems and engineers characterise physical systems abstractly assuming some hypothesis. It is extremely important to explicitly specify domain knowledge with a system in order to improve the quality of the development process and to accept new changes in CPS requirements [Ait-Ameur 2016]. Consideration of domain knowledge in software engineering processes is seen as a significant step in the field of system modelling and analysis.

The triptych [Jackson 1993, Zave 1997, Bjørner 2017] method covers three main phases of the software development process: *domain definition*, *requirements prescription* and *software design*. $\mathcal{D}, \mathcal{S} \longrightarrow \mathcal{R}$ expresses a formal notation, in which $\mathcal{D}$ represents domain concepts in the form of properties, axioms, relations, functions and theories; $\mathcal{S}$ represents a system model; and $\mathcal{R}$ specifies desired system requirements. This notation indicates that the domain description ($\mathcal{D}$) and system model ($\mathcal{S}$) are valid in relation to the requirements ($\mathcal{R}$). The proposed structure must respect

the separation between system and physical environment, and the simulated system must accomplish the environment properties [Jackson 1993].

To develop a shared understanding of the components and concepts that comprise CPS, and to expose the CPS specific functions to domain experts, a domain model is necessary. This model may offer only key characteristics, functionalities, and potential interactions that can be used for building a cyber-physical system. To share common understanding among different stakeholders, ontology can be used to represent various core elements of the CPS, such as communication, sensors, actuators, controllers, and so on [Hildebrandt 2018]. Such domain models can be developed once and can be reused in the development of other CPS. Due to the increasing complexity of CPS in different domains, there is a clear need for domain models of physical systems as well as a domain model for characterising abstract behaviour, including different kinds of cyber system properties.

Design automation

Design automation and deployment are already present in computer-aided design technologies, but supporting partial or full automation in the development of CPS is lagging behind due to the complex interaction of cyber and physical systems, as well as different types of needs by a class of CPS belonging to different domains such as avionics, medical, transpiration, and so on. Currently, tool and technology companies primarily provide automation solutions for a limited set of problems that cannot be scaled to handle the development of complex CPS. CPS is a heterogeneous system composed of several physical systems and various models of computation and communication. The heterogeneity nature of different class of CPS leads software and system engineers to make product specific design-flows, which is bad indication for design automation. In fact, the software and system engineers must consider low-level product specific design decisions, thus there is a lack of interest by tooling industries. We are unable to meet productivity and time to market targets for CPS due to a lack of design automation.

Today, we are capable of using computers in all aspects of problems; thus, we should be able to leverage tools and processes used in complex engineering disciplines such as embedded systems across a broad range of CPS covering domains such as transportation, medical, avionics, and so on. All engineering disciplines share core design principles such as abstraction and refinement, but they have different tools and techniques for different applications. Off course there are pros and cons with each of them [CPS Steering Group 2008]. The increasing demand for new methods and tools for developing CPS necessitates the investigation of new methodologies, including supporting tools for dealing with complex CPS and certification standards requirements by unifying existing methods, development strategies, including techniques and tools from other engineering disciplines.

Heterogeneous systems

Heterogeneity exists in all types of complex systems. Cyber-physical systems inherit heterogeneity as a result of the composition of different components and their interoperability, as well as the requirement for essential design to meet requirements and design decisions. Furthermore, CPS has a wide range of physical requirements, including those for specifying dynamics, power, and physical size, as well as system-level requirements such as safety, security, and fault tolerance.

It should be noted that the concepts of *'separation of concerns'* have played an important role in managing multi-objective design problems, when design views are orthogonal, which means that design decisions in one view do not influence design decisions in other views. It is not feasible in the case of CPS due to complex interactions between system layers and design views that are frequently not modelled [CPS Steering Group 2008]. Physical dynamics, sensors, actuators, software, controller, and communication network are all parts of CPS design that must work together appropriately for the systems to perform well. Modelling formalisms, analysis techniques, and tools for developing these numerous aspects originated independently and remain different and diverse. For example, integration of discrete-event and continuous-time modelling paradigms. There is no unifying formalism capable of modelling all of these elements equally effectively. In fact increasing heterogeneity in CPS, the current modelling approach and design methods produce poor results; and we don't have any key solution that can directly deal with modelling heterogeneity. However, interoperability and model-based design have shown some promising results in dealing with

complex CPS [Bhattacharyya 2020]. More sophisticated solutions, methodologies, and approaches that can handle CPS heterogeneity from requirement engineering through deployment are required.

Composition

Composition is an essential operation in all engineering disciplines for dealing with complexity and meeting goals within given constraints associated to hardware, software, cost and so on. Compositionality and composability are two important operations that enable computing system properties from local component properties and component properties that do not change during interaction with other components, respectively. Inaccuracy in these factors causes unwanted behaviour and introduces irregularities in core components functionalities.

Component-based design is widely accepted as a first-class citizen in engineering disciplines, where homogeneity in terms of the properties composed and the semantic framework used in modelling is a feature shared by all successful compositional design frameworks. As a CPS is made up of both cyber and physical systems, designing one presents new challenges due to the intertwined nature of digital control with physical processes and the environment. In this case, there is a growing need to formally develop new modelling techniques that can assist system designers in developing a closed-loop model of a cyber system and a physical system. This closed-loop modelling approach ensures not only individual behaviour of both cyber and physical systems, but also emergent behaviour that may emerge as a result of the systems' composition. Furthermore, this can be an effective approach to ensuring the correctness of the functional behaviour and CPS requirements [14].

Note that the cyber system also includes sensors, actuators, a controller, a computing platform, and a communication network, there is a need to ensure the overall correctness of the cyber system. If there is any undesirable behaviour in CPS, the system enters a hazardous state, which can lead to accidents. To avoid such an unfavourable outcome, we must ensure overall correctness prior to deployment; otherwise, discovering an error late in the development life-cycle may result in a significant financial loss. To perform reasoning on a composite system, cyber and physical systems, as well as individual components, compositional methods are required to ensure the correctness of a cyber-physical system [Sztipanovits 2007].

Security and privacy

Security and privacy are important for establishing trust in CPS. To manage both technical and nontechnical details, there is a direct link between trust and CPS. In the case of CPS, physical systems may be attacked via cyberspace, and cyberspace can be attacked via physical devices. An attacker may cause serious failure, resulting in massive financial losses, or may create a panic scenario, endangering human lives. Most CPS are designed as stand-alone units, with no distributed computer-based control to ensure reliability. The massive network infrastructure, including protocols, designed to support multimedia and entertainment is unsuitable for the development of secure cyber-infrastructure. Using existing network services but enhanced with security to develop distributed CPS may have serious consequences if the CPS under attack cannot be easily recovered. As a result, we must investigate the area of security and safety in order to design safe and secure CPS by understanding the various types of failures associated with CPS and empowering our knowledge to protect against these failures [CPS Steering Group 2008].

Certification

Certification is a major challenge in developing a safety-critical cyber-physical system. Today, we may certify computer-based systems by first constructing them and then testing them in accordance with standards that place requirements on the development process as well as testing-based evidence. The safety-critical standards establish development process objectives in a manner akin to software engineering principles, thus the acceptance criteria are always in favour of the development process rather than the product being made. Of course, this ensures the utilisation of "excellent" methods but not the "good" output [Maibaum 2008]. This approach is not scalable, and the cost of certifying complex systems, such as medical or avionics, can be prohibitively expensive.

Regulators, for instance the U.S. Food and Drug Administration (FDA), have been dissatisfied with the frequency of the recalls of many of the products evaluated in such a process based regime. Of course, we can make our standards specify the product-focused evidence that is required, as well as the acceptance criteria for this evidence. However, software engineering has a poor track record in this area. One approach is to identify critical properties of a system that are required to achieve tolerable risk in terms of the system's safety, security, and reliability. Assurance Cases are gaining popularity as a means of documenting such claims about these critical properties of a system, along with evidence and supporting reasoning for why the claims are valid. To help improve the quality of products submitted for approval, the FDA has turned to assurance cases [U.S. Food and Drug Administration 2010]. Assurance cases are one method of documenting a convincing argument about the trustworthiness of the resulting system, based on the identification of specific pieces of evidence and the fulfilment of explicit acceptance criteria involving both product and process [16]. To address this issue, we can look into compositional certification, which allows us to certify different parts of a complex cyber-physical system separately before combining them, as well as the development of Assurance Case Templates, which provide explicit guidance on how to write an effective assurance case for a specific CPS.

1.2 Research direction

A cyber-physical system has a wide range of application areas and is dependent on various technologies related to cyber systems, physical systems, and networking. There are several research avenues that could be pursued in order to design safety-critical cyber-physical systems. Some of the most significant challenges have been described above, and we have used some of them to pave our research.

In recent years, our research has focused on modelling, designing, and implementing safety-critical cyber-physical systems using software engineering principles and techniques, as well as formal methods. We have three broad research areas in this direction:

1. Domain knowledge engineering;
2. System modelling;
3. Software and system certification;

Our research agenda revolves around three main questions:

1. What is domain knowledge, and what is the relationship between domain model and system model?
2. How should a complex system and its environment be designed using a correct by construction state-based method?
3. What can be done to assist with software and system certification?

In each question, we ask how software engineering methods, development processes, and certification standards are used and contested in the design of safety-critical cyber-physical systems.

1.3 Organisation of the manuscript

This manuscript synthesises our research contributions on domain knowledge engineering, system modelling, and certification, for developing safety-critical cyber-physical systems. It is organised in four chapters.

Chapter 2 summarises our work on domain knowledge engineering for dealing with various core concepts of safety-critical cyber-physical systems. In this work, we study the concepts of implicit and explicit modelling at high level, and then use these concepts as well as domain concepts, to develop a refactoring methodology for complex formal models that supports modularity, domain knowledge integration, re-usability, and maintainability. The refactoring methodology is evaluated by refactoring a complex formal model of an ECG clinical assessment protocol from the

medical domain. Furthermore, we propose a unified framework that integrates domain knowledge, system specifications, and safety requirements in Event-B formal modelling setting and proof system. In this work, we show how explicit domain knowledge representation as ontologies can benefit formal system design models. We develop an Event-B meta-theory that describes an ontology based on generic and abstract datatypes and operators. A complex avionic case study, Traffic Collision Avoidance System (TCAS), validates the proposed framework and theories.

Chapter 3 summarises our work on the fundamental concept of system modelling for designing safety-critical cyber-physical systems. Our main focus throughout this work was on developing generic formal frameworks, modelling patterns, architecture patterns, approximation patterns, refinement automation, simulation, environment modelling, and code generation, all with the goal of modelling, designing and verifying safety-critical cyber-physical systems using state-based methods like Event-B. All of these concepts are applicable to any type of system. Two refinement strategies are proposed to automate the process of formalising system requirements from tabular expressions using a correct-by-construction approach. Further, a generic and extensible reusable formal framework, consisting of a method and a set of tools, is proposed for developing and verifying critical systems. This framework consists of a large set of theories that extend Event-B with mathematical features required to model continuous behaviours (e.g., differential equations). Finally, three formal patterns for hybrid system design are defined: approximation, centralised control with multiple plants, and distributed hybrid systems. The development life-cycle for rigorous development of critical systems is proposed supporting requirement analysis to code generation. In addition, we propose a formal framework, F3FLUID (Formal Framework For FLUID), and the structuring of Event-B models using model-view-controller (MVC) to address the key challenges of safety-critical interactive systems. Furthermore, we present the reflexive EB4EB framework, which enables users to explicitly manipulate Event-B features through the use of reflection and meta-modelling concepts, as well as the extension of this framework to improve Event-B reasoning mechanisms for expressing deadlock freeness, invariant weakness analysis, reachability and temporal properties. We also propose the development of a virtual environment model, closed-loop modelling, a virtual environment model for verification, simulation and clinical trials, supporting techniques and tools. Finally, we propose a new extension of the code generation tool, EB2ALL, by developing a new plug-in, EB2Sol, for generating Solidity code from Event-B models for the Ethereum platform, as well as extending existing plugins (EB2C, EB2C++, EB2J, and EB2C#) to incorporate new modelling constructs. Several applications are also discussed to demonstrate our approaches.

Chapter 4 summarises our work on certification and the development of assurance cases for safety-critical cyber-physical systems. Our main focus throughout this work has been on developing an integrated verification framework that can be used to aid in the certification process. The research focuses on the integrated verification of system design models for transportation systems, specifically railway systems. It was accomplished as part of RATP's B-PERFect project, which aimed to apply formal verification using the PERF approach to integrated safety-critical models of embedded software expressed in a single unifying modelling language: High Level Language (HLL). If manufacturers want to market their products, certification bodies have recommended that they submit an Assurance Case. The reasoning was that it would assist manufacturers in developing safer and more reliable systems, as well as provide certification bodies with a better foundation for evaluating these submissions. We have been investigating the use of Assurance Case Templates to guide the development of a software-intensive critical systems. A template of this type will also provide explicit guidance on how to write an effective assurance case for a specific product within the scope of the template's identified product scope. We believe that a product-domain-specific template can serve as a standard for developing and certifying a safety-critical cyber-physical system in that specific product-domain.

Chapter 5 summarises various applications pertaining to safety-critical cyber-physical systems. Our main goal is to demonstrate the usability, reliability, maintainability, portability, efficiency, correctness and scalability of our proposed methods, techniques, and tools for domain knowledge engineering, system modelling, and certification and assurance case on a variety of complex safety-critical cyber physical systems. We focus on three types of systems in particular: *Hybrid Systems*, *Interactive Systems* and *Medical Systems*. Several case studies are developed for each domain to demonstrate the application of proposed methods and techniques related to domain knowledge engineering, system modelling, and certification.

Finally, in Chapter 6, we conclude by presenting the perspectives for future research that our previous work has opened.

Biographical note

Since completing my PhD, the work presented in this manuscript has been the primary focus of my research. In the interest of thematic consistency, some of my works have been omitted. Appendix A contains a complete list of my contributions.

Prior to beginning work on the modelling, design, and implementation of safety-critical cyber-physical systems, I completed a PhD in the area of formal modelling and design of critical device software systems using the correct by construction approach. I proposed a new development life-cycle, as well as a set of related techniques and tools for developing highly critical systems using formal techniques ranging from requirements analysis to automatic source code generation. I then spent a year and a half as a postdoc researching the development of formalisms, techniques, and tools for the correct construction and reasoning of Safety-Critical Java (SCJ) programs, before moving on to requirement engineering, formal development, functional safety standards, safety cases, and certification of critical systems in a variety of domains, including nuclear power generation, medical systems, and automotive in my second postdoc. The research presented in this manuscript is a continuation of my PhD and postdoctoral work. My diverse background in a variety of domains, including modelling, designing, simulation, and implementation, has been extremely beneficial as I begin to investigate the core challenges of safety-critical cyber-physical systems, specifically cyber and physical systems modelling using correct by construction approaches, domain knowledge engineering, meta-modelling, safety cases development, certification and so on.

CHAPTER 2

Domain Knowledge Engineering

This chapter covers the work in papers [38, 40, 43, 54, 59][12][19]. This work was done in collaboration with Yamine Aït-Ameur (INPT-ENSEEIHT, France), Régine Laleau (Université Paris-Est Créteil, France), Dominique Méry (University of Lorraine, France), Philippe Palanque (Toulouse III - Paul Sabatier University, France); and with following students: Ismail Mendil (PhD student at INPT-ENSEEIHT/IRIT, France under co-supervision of Yamine Aït-Ameur, Dominique Méry, Philippe Palanque and myself), and Peter Riviere (PhD student at INPT-ENSEEIHT/IRIT, France under co-supervision of Yamine Aït-Ameur and myself).

This chapter summarises our work relating to domain knowledge engineering for safety-critical cyber-physical systems. Our main focus throughout this work was to investigate implicit and explicit modelling, model refactoring, and Event-B theories for handling domain knowledge. Domain knowledge engineering is essential in the design of complex systems. During system development, such knowledge is always encoded implicitly while some assumptions are taken into account. We believe that one of the primary causes of system failure or missing requirements is a lack of domain concepts in the development of safety-critical CPS. As a result, domain engineering concepts and their integration with systems must be carefully reconsidered. Further, we investigated the use of domain concepts for model refactoring in the formal methods area to support modularity, domain knowledge integration, re-usability, and maintainability. In similar vein, we propose the development of Event-B theories for handling domain knowledge. In this work, we present an Event-B meta-theory for describing an ontology model and formalising an ontology modelling language.

In the remainder of the chapter, we sequentially describe our contributions to implicit and explicit modelling, model refactoring, and Event-B theories for handling domain knowledge along with applications.

2.1 Implicit and Explicit modelling and Refactoring

2.1.1 Context

Domain analysis is a subset of *domain engineering* that provides background information to system engineers so that they can understand the problem and analyse system requirements [Bjørner 2009]. System engineers may employ a variety of languages to analyse complex system requirements. They must perform various types of verification and validation activities based on the selected languages and the associated analysis techniques. Analysing requirements is a time-consuming and lengthy process in which engineers elaborate their models using complex design descriptions but fail to explicitly model relevant domain-related information. Domain knowledge is not explicitly addressed in the system development process in general. Such knowledge is always encoded implicitly during system development, while some assumptions are taken into account. It is highly desirable to explicitly define domain knowledge with a system to improve the quality of the development process and to accommodate new changes in system requirements [Ait-Ameur 2016]. Consideration of domain knowledge in software engineering practices is regarded as an important step in system modelling and analysis.

In this direction, the triptych [Jackson 1993, Zave 1997, Bjørner 2017, Bjørner 2019] approach covers three main phases of the software development process: *domain description*, *requirements prescription* and *software design*. $\mathcal{D}, \mathcal{S} \longrightarrow \mathcal{R}$ represents a formal notation in which $\mathcal{D}$ represents domain concepts in the form of properties, axioms, relations, functions and theories; $\mathcal{S}$ represents a system model; and $\mathcal{R}$ represents the intended system requirements. Similarly, Jackson's structure [Jackson 1993] $\mathcal{E}, \mathcal{S} \vdash \mathcal{R}$ describes the requirements, where $\mathcal{E}$ represents the given environment, $\mathcal{S}$ represents the specification, and $\mathcal{R}$ represents the requirement. These proposed structures always

keep the distinction between system and the physical environment, as well as ensuring that the identified environment properties are satisfied by the modelled system.

In the system engineering process, identifying and analysing the requirements that the system must meet is the first step in developing a formal specification for a system. There are several requirements engineering methods such as KAOS [van Lamsweerde 2009], i* [Yu 1997], Problem Frames approach [Jackson 1995] and Problem Oriented Software Engineering (POSE) [Hall 2008], but they agree that understanding and describing the domain in which the system will behave correctly is required in order to express the right requirements and, as a result, build the right specification that meets the requirements. In [van Lamsweerde 2009], the authors introduced the concepts of *expectation* and *hypothesis*, which are specific cases of prescriptive and descriptive statements, respectively, and both must be satisfied by the system environment. There are various mathematical analysis approaches and modelling languages to support domain modelling concepts in the form of models, meta-models, ontologies, and so on. In [Ait-Ameur 2016], the authors proposed annotating design models with domain-specific knowledge for state-based methods. Recently, the textbook [Aït Ameur 2021] reviewed many cases of system models exploiting explicit models of domain knowledge in medical systems [59][12], e-voting systems [Gibson 2021], distributed systems, and other areas.

As previously stated, domain knowledge is mostly encoded in system models (implicitly), and there is no distinction between domain and system models. This may result in the development of complex models as well as a lack of reuse of domain models when developing another systems. Furthermore, such modelling mechanisms do not explicitly consider the triptych approach, implying that the domain model must be explicitly defined. To address this issue, we borrowed refactoring principles.

Refactoring is a popular programming approach that allows us to restructure source code without changing the system's functional behaviour. This technique aids in the systematic cleaning up of developed code by replacing complex instructions with simple instructions, reducing the risk of introducing bugs, introducing modularity, and improving the code's readability and maintainability [Fowler 1999, Opdyke 1992, Du Bois 2004]. The initial idea of refactoring was proposed by Opdyke [Opdyke 1992] and Griswold [Griswold 1992] in their dissertations. Fowler et al. [Fowler 1999] described the code refactoring approaches, methods, and tools.

Refactoring techniques are not limited to programming languages. They have been adopted by formal specification modelling languages: [Kobayashi 2016] for Event-B, [Yaghoubi Shahir 2012] for ASM, [Gheyi 2004] for Alloy, and [McComb 2004] for Object-Z. Whiteside et al. [Whiteside 2011] proposed a proof script refactoring approach for constructing, restructuring, and maintaining the development of formal proofs to support complex proofs. Kobayashi et al. [Kobayashi 2016] proposed the refactoring approach to restructure the refinements in Event-B. The main contribution is refinement decomposition based on a slicing strategy of a large model.

Our main goal is to guide refactoring by explicitly modelling domain knowledge in a system model, reducing the complexity of proof structures, improving the maintainability of the developed formal model, exposing any existing bugs, and improving the readability and reusability of the explicit domain model.

2.1.2 Our contributions

In this context, we propose the domain concepts based on ontology to integrate with the system model in an explicit way [12][54]. We use the Event-B language to develop a domain model using ontologies as well as a system model. Furthermore, the four step modelling methodology [Ait-Ameur 2016] is extended for model refactoring and identifying a set of modelling patterns applicable for refinement-based formal development [59], as well as a complete model analysis that reveals intuitive messages on how to perform refactoring on large complex models. The four step modelling methodology and refactoring approach, and the main results derived from them, are presented in the following section.

Four step modelling methodology

This section presents a four step modelling methodology borrowed from [Ait-Ameur 2016] to formalise a complex medical protocol [12]. Fig. 2.1 depicts the modelling methodology, which includes domain modelling, system modelling, model annotation, and model verification. These modelling steps are as follows:

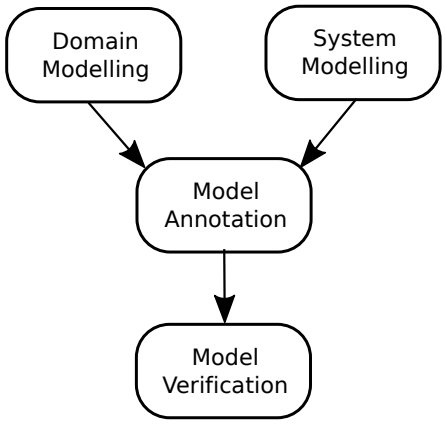

Figure 2.1: Four steps modelling methodology

- **Domain Modelling.** Domain knowledge is required when making assumptions about a given system. The required information for a domain can usually be considered hypothetically based on prior experiences and domain knowledge. It should be noted that by defining concepts, entities, relationships, constraints, and rules, an ontology modelling language can be used to characterise and formally specify domain knowledge in the form of domain ontology. We use the Event-B [Abrial 2010a] modelling language to formalise the required domain concepts derived from the domain ontology, which can be described in Event-B context using *sets*, *constants*, *axioms* and *theorems*.

- **System Modelling.** Developing a safe system while taking into account all of the necessary functionalities is a challenging problem. To design a safe system, we can use any formal modelling language to describe a desired behaviour under the given specification. The chosen modelling language and verification approach enable us to validate the required behaviour. In our approach, we also use the Event-B modelling language for modelling a system model, which allows us to gradually develop the system behaviour while satisfying the required safety properties using machines and contexts.

- **Model Annotation.** Model annotation is a mechanism for describing design model entities and ontology concepts in order to establish a relationship between the domain model and the system model. The self-contained annotation mechanism can be used to annotate the system and domain models. These models can be constructed using either the same or different formal notations. An independent annotation mechanism, such as a plugin, can be used to connect the domain model and the system model. We do not have a specific annotation mechanism in our approach because we use the same modelling language (Event-B) for both domain and system models. As a result, we have a free implicit annotation mechanism (i.e. *see* context relationship) for integrating domain and system models. To integrate the domain model and system model in Event-B, we use the domain model as a set of contexts for describing the desired properties and functional behaviour while developing the system model.

- **Model Verification.** This is the final step in the modelling methodology and can be completed after annotating a system model with a domain model. The ontology-expressed domain properties enrich the annotated design model. The annotated model should be verified in two steps. The first verification must be performed prior to annotation on the designed system model (which may no longer be correct after annotation) to ensure consistency, and the second verification must be performed after annotation to ensure overall consistency while taking domain knowledge into account. It is worth noting that the verification in the second step also allows us to check the new emerging properties as a result of the integration of the domain model and the system model via the annotation mechanism.

The refactoring methodology described in the following section is developed using the four-step modelling methodology.

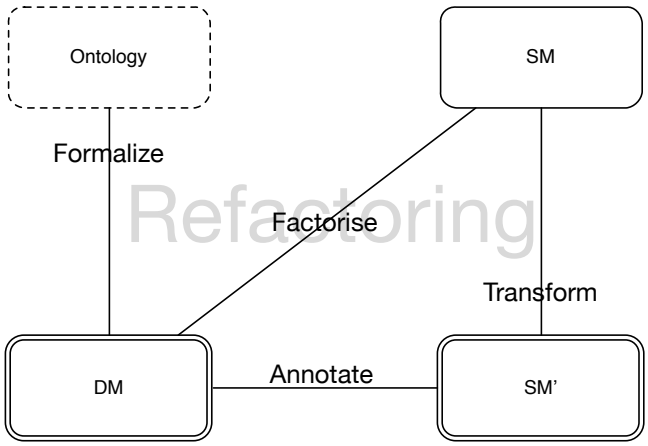

Figure 2.2: Generic Refactoring Methodology

Refactoring Methodology

Refactoring techniques, which allow changing the structure of a model without changing the system functionalities and behavioural objectives of the model, are a common way of restructuring, introducing modularity, minimising the complexity of proof structures, and improving the maintainability and readability of a formal specification [Fowler 1999, Opdyke 1992, Du Bois 2004]. In general, there are two types of refactoring techniques: *structural refactoring* and *behavioural refactoring*. Structural refactoring modifies the structure of a formal model without changing its behaviour or reachable states. This structuring mechanism allows a developer to transfer the same safety properties to the new refactored model because this refactoring ensures observability equivalence. The behaviour of a formal model may change as a result of behavioural refactoring. In fact, when using this approach, there are partially reachable states when compared to the formal model prior to refactoring [Mitsch 2014]. In our work, we employ structural refactoring.

We propose using structural refactoring to express domain knowledge explicitly in a formal model. The graphical layout of the input and output of the defined structural refactoring moving source models to target models is shown in Fig. 2.2. The upper part of this figure depicts the source models (system models (SM) and ontology).

We believe that ontologies[1] exist but they are not used by system models SM because they do not explicitly refer to the domain model (ontology). The target models are represented in the lower part of the Fig. 2.2 by domain models (DM) derived from the ontology and refactored system models (SM'). The horizontal lines represent model dependencies (e.g., visibility, extension) between models, while the vertical lines represent refactoring operations. For example, the target domain models (DM) are developed by formalising ontologies. The target system models (SM') are refactored from the source system models SM.

There is a set of refactoring operations identified. The approach we propose consists in determining whether a refactoring operation can be applied to any complex formal model developed progressively using a *correct by construction* approach, and whether domain-specific conceptual knowledge is implicitly formalised in a system model. Each refactoring operation can be viewed as a before-after predicate that preserves the properties of the source models while making explicit domain knowledge in the target models (refactored models).

To aid in the refactoring of system models SM, we identified a set of structural development operations that correspond to specific model mappings. These mappings must meet the characteristics of ontologies, particularly the unique referencing mechanism. We have identified the following operations.

- **Formalise_DM.** The process of developing a domain model by selecting relevant ontologies associated with

[1] Several ontologies and domain models have proposed by several organisations, standards, companies, etc. The process of building these ontologies is beyond the scope of this chapter.

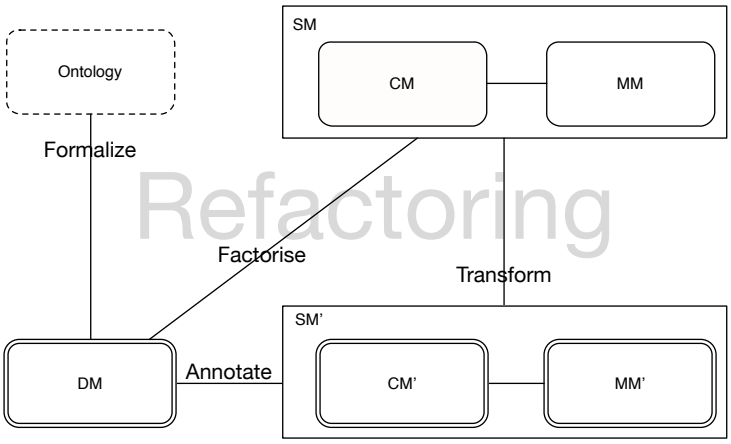

Figure 2.3: Refactoring Methodology for Event-B

the studied system is defined as DM. This DM may be formalised as a context or a theory, depending on the formal method used. The consistency of DM must be guaranteed (axioms providing definitions of domain concepts shall be inhabited).

- **Factorise_SM_2_DM.** The operation moves from the system model SM to the ontology or DM via the *definitions of concepts* (e.g. definitions related to variables, invariants, or theorems) of SM. If these concepts are not available, they are elevated to the ontological level; otherwise, they are added as redundant concepts (derived concepts using ontology modelling operators).

- **Transform_SM_2_SM'.** A target system model SM' is created from a source system model SM by adding relationships to the DM model, such as direct references to DM concepts or mappings between SM and DM concepts. This operation may require the rewriting of both static (axioms, theorems, and so on) and behavioural concepts (guards, before-after predicates, substitutions, etc.).

  Note that in this case, the newly emerging invariants and theorems can be expressed in the SM' model. Domain knowledge explication entails them.

The previous operations mention the concept of model mapping. Ontology engineering provides several types of mappings such as equivalence, subsumption, and algebraic mappings that can be formalised in Event-B. Finally, after refactoring, the SM models must be verified to ensure that the refactored model is correct in relation to the original model.

In the context of the Event-B modelling language, we use the generic refactoring methodology to develop the domain model and system model together. Fig 2.3 depicts an extended graphical layout of the generic refactoring methodology to show the various components of the Event-B models. In the extended figure, the system model SM is composed of the context model CM and the machine model MM. Similarly, the refactored system model SM' is made up of the context model CM' and the machine model MM'.

2.1.3 Application

We revisited the ECG clinical protocol [90] to demonstrate the four step modelling methodology [12] as well as the refactoring methodology [59]. In this section, we recall ECG and develop the domain model, context model and system model progressively using refactoring methodology.

An electrocardiogram (EKG or ECG) [Khan 2008] signal presents an electrical activity of the human heart in continuous form to show the depolarisation and re-polarisation phenomena. A typical cycle of ECG (see Fig. 2.4)

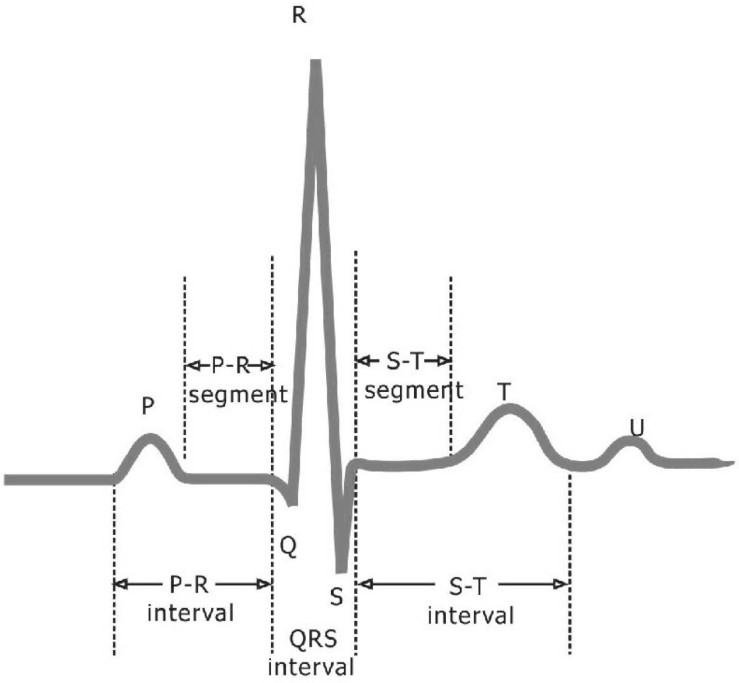

Figure 2.4: ECG Deflections

represents a sequence of waves and intervals, which is denoted as P-QRS-T-U. These waves and intervals are defined as: *P-wave* - a small deflection caused by the depolarisation of atria before contraction to show an electrical wave propagation from the SA node through the atria; *PR interval* - an interval between the beginning of the P-wave to the beginning of the Q-wave; *PR segment*- a flat segment between the end of the P-wave and the start of the QRS interval. *QRS interval* - an interval between the P-wave and T-wave with greater amplitude to show the depolarisation of the ventricles; *ST interval* - an interval between the end of the S-wave and the beginning of the T-wave; *ST segment* - a flat segment starts at the end of the S-wave and finishes at the start of the T-wave; *T-wave* - a small deflection caused by the ventricular re-polarisation, whereby the cardiac muscle is prepared for the next cycle of ECG; and *U-wave* - a small deflection immediately following the T-wave due to re-polarisation of the Purkinje fibers.

In this work, we adapt existing works [Khan 2008, Gonçalves 2007, ECG Ontology , Gonçalves 2011] for designing and developing the ECG domain model. It should be noted that the developed ECG domain model, which is based on existing ontologies, contains very abstract information about the heart and ECG while hiding the main complexities. It is important to include complex details in order to consider every aspect of domain knowledge. To keep things simple, the domain model developed is only used to realise the case study of the ECG protocol. Using the refactoring methodology, we develop the domain model from available ontologies and the existing system model. There are several databases and ontologies that we are aware of that represent the ECG. We have used the OBO (Open Biomedical Ontologies) Process Ontology to describe the conceptual knowledge of the biological process of the ECG, which is classified as the fundamental relation, spatial relation, temporal relation, and participation relation [Bittner 2007]. The two most important fundamental relations in this work are is_a and $part\_of$.

$$A\ is\_a\ B = \forall x[\mathbf{inst}(x, A) \Rightarrow \mathbf{inst}(x, B)]$$
$$A\ part\_of\ B = \forall x[\mathbf{inst}(x, A) \Rightarrow \exists y(\mathbf{inst}(y, B)\ \&\ x\ \mathbf{part\_of\_inst}\ y)]$$

The is_a relation states that every instance of class A is an instance of class B and the second relation states that *A part_of B* holds if and only if: for every individual x, if x instantiates A then there is some individual y such that y

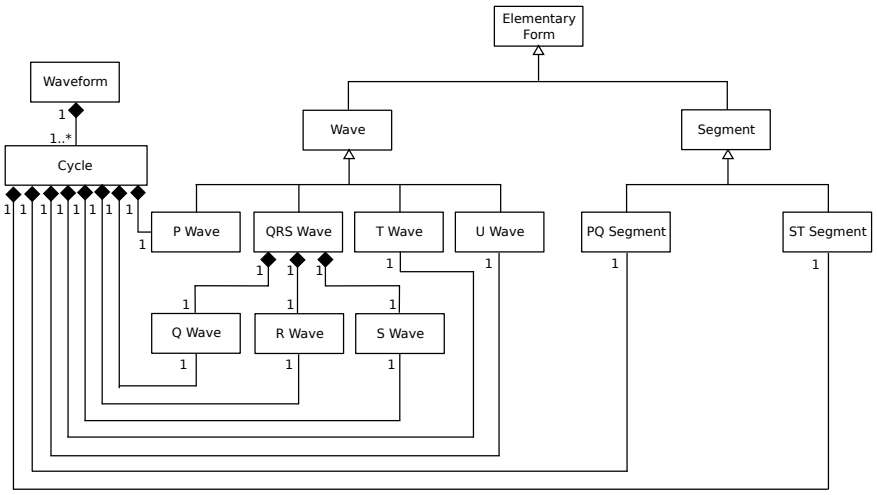

Figure 2.5: ECG ontology

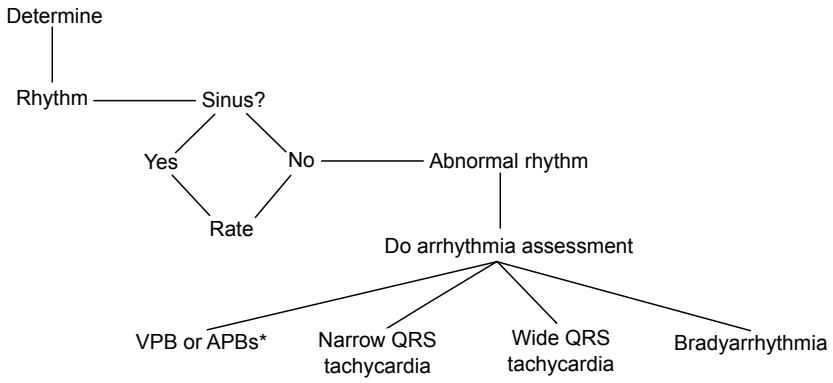

Figure 2.6: Basic Diagram of Assessing Rhythm and Rate [Khan 2008]

instantiates B and x is a part of y. In the previous definitions, **inst** is a relation between a class instance and a class which it instantiates and the **part_of_inst** is a relation between two class instances. Other relations are defined in ontology modelling languages. All of them are rigorously defined in Event-B.

Fig. 2.5 presents a high level description of the ECG using the OBO relations for deflections known as waves and segments. The elementary concepts are represented using the is_a and $part\_of$ relationships. There is the ***part_of*** relationship between the elementary entity and the wave and segment entities. In a typical ECG cycle, there are two kinds of segments, PQ Segment and ST segment. The ***is_a*** relationships are used to denote the relations between Segment, and ST segment and PQ segment. Similarly, the Wave entity is also divided into the different types of waves: P wave, QRS wave, T wave and U wave. These waves are also related to the Wave entity using the ***is_a*** relationship. There is the ***part_of*** relationships between the QRS Wave and Q wave, R wave and S wave. In a similar way, the Cycle entity and different waves (P wave, Q wave, R wave, S wave, T wave and U wave) entities and segments (PQ segment and ST segment) are connected with the ***part_of*** relationship. The initial set of axioms are defined by applying the *Formalize_DM* refactoring operation (see Fig. 2.2 and Fig. 2.3).

We describe the stepwise development of the domain model and system model covering the given requirements. This is a generic development where the domain model and system model evolve progressively. Fig. 2.6 depicts a standard clinical procedure for analysing the ECG. The initial assessment step allows us to check the sinus rhythm and the heart rate (state of the heart), formally defined in the abstract model using domain knowledge and the required

Model	Old Model			Refactored Model		
	Total number of POs	Automatic Proof	Interactive Proof	Total number of POs	Automatic Proof	Interactive Proof
Abstract Model	41	33(80%)	8(20%)	43	22(52%)	21(48%)
First Refinement	61	54(88%)	7(12%)	49	36(74%)	13(26%)
Second Refinement	41	38(92%)	3(8%)	39	32(82%)	7(18%)
Third Refinement	51	36(70%)	15(30%)	47	39(83%)	8(17%)
Fourth Refinement	60	35(58%)	25(42%)	50	36(72%)	14(28%)
Fifth Refinement	43	22(51%)	21(49%)	36	29(81%)	7(19%)
Sixth Refinement	38	14(36%)	24(64%)	30	22(74%)	8(26%)
Seventh Refinement	124	29(23%)	95(77%)	114	74(65%)	40(35%)
Eighth Refinement	52	30(57%)	22(43%)	53	33(63%)	20(37%)
Ninth Refinement	21	9(42%)	12(52%)	15	12(80%)	3(20%)
Tenth Refinement	67	43(64%)	24(36%)	65	54(84%)	11(16%)
Total	**599**	**343(58%)**	**256(42%)**	**543**	**391(73%)**	**152(27%)**

Table 2.1: Proof Statistics

domain specific clinical properties. The clinical properties use the domain knowledge to specify the ECG assessment protocol. These clinical properties are introduced in the context model using the refactoring operations *Transform_SM_2_SM'* (see Fig. 2.2 and Fig. 2.3). *Note that these properties were introduced implicitly in the previous model of the ECG protocol [90].*

In the abstract model, we define state variables to represent sinus state, heart rate, and heart state, including a set of safety properties. The defined safety properties are modified according to the refactoring operation *Transform_SM_2_SM'* (see Fig. 2.2 and Fig. 2.3). The ECG protocol abstract model includes three events that assess the heart state by analysing the heart rhythm and normal or abnormal heart rate. These events specified the required behaviour using domain model knowledge and the clinical properties provided.

The abstract model is further enriched by gradually introducing the necessary assessment steps in a sequence of refinements, which corresponds to the standard analysis step of the ECG protocol [Khan 2008]. A detailed formal development of this ECG protocol is available on the website[2].

Model Verification

This section describes the proof statistics of the developed model using refactoring approach. As we know that this development is based on the Event-B modelling language, which allows us to check the *consistency checking* and *refinement checking*. Table 2.1 presents the proof statistics of the progressive development of the old ECG model and the refactored ECG model. In this development applying the proposed refactoring approach, we achieve 543 (100%) proof obligations, in which 391 (73%) POs are proved automatically, and the remaining 152 (27%) are proved interactively using the Rodin provers, while the old development has more POs. Note that the generated POs of new refactored model also include other possible POs related to refactoring operations. The old model has more POs, including more manual interactions, due to the complex predicates and implicit domain knowledge. This refactoring approach has simplified the modelling constructs and development process that allows us to automate the several proof strategies of the refactored model. Moreover, the interactive proof obligations are also very simple that are proved with the help of SMT solver.

First, we would like to point out that our approach has been deployed on a non trivial development issued from the medical domain. Note that the obtained new refactored ECG model is simpler than the old ECG model. Some of the states and behavioural properties that were previously defined implicitly in the old ECG model are now defined explicitly in the new refactored ECG model. The new obtained model refers to shared ontological definitions.

[2]http://singh.perso.enseeiht.fr/Conference/ICECCS2018/ECGModels.zip

Moreover, according to the Table 2.1, the proof efforts have been reduced when compared to the previous formal model developed in [90]. In particular, the number of interactive proofs has been drastically reduced. Indeed, the domain model properties are proven once and for all in DM, and they are used as hypotheses to prove the system model SM properties.

The results shown in Table 2.1 indicate that using a refactoring approach with explicit domain knowledge significantly improved the formal development process and produced new simplified proof strategies.

In this study, we have discovered several anomalies in the ECG interpretation protocol, which we divide into three main categories: *ambiguity*, *inconsistency* and *incompleteness*. Ambiguity is a well-known anomaly that can represent more than one possible meaning of a fact, potentially leading to decision confusion. In our work, for example, we had to determine whether the terms “ST-depression” and “ST-elevation” have the same meaning. For similar data/input, an inconsistency anomaly always results in a conflicting result or a different decision. For example, in our work, we have discovered an inconsistency in the form of applicable conditions, which state that the given conditions apply to both "male" and "female" subjects, but elsewhere in the protocol it is advised that the given conditions do not apply to "female" subjects. Incompleteness is caused by either a missing piece of information or a lack of information in the original document. For example, the original protocol includes "normal variant" factors to be considered when assessing T-wave. However, the meaning of “normal variant” is not defined in the protocol. Note that we have not listed all anomalies. In our work, we have identified these anomalies which may help for improving the quality of medical protocols.

Summary of our contribution to model refactoring:
We propose a refactoring approach that allows us to refactor a complex formal model, where the formal model is developed using a *correct by construction* approach and the domain concepts are modelled implicitly. We propose a set of operations that allows us to refactor a system model while considering domain specific knowledge in the form of ontology to produce the domain model and system model while preserving the correctness system’s functional behaviour. Our main contributions include: developing a refactoring technique related to the *correct by construction* approach; explicitly using domain specific knowledge in a system model; defining a set of modelling patterns; and defining a restructuring mechanism in formal development. Finally, a complex medical case study, ECG clinical assessment protocol, is used to evaluate the proposed approach.
Project: IMPEX – Implicit and Explicit Semantics Integration in Proof-based Developments of Discrete Systems (funded by ANR)
Publications: [12][40, 54, 59]
Software: Refactoring patterns, ECG protocol, and models

2.2 Event-B Theories for Handling Domain Knowledge

2.2.1 Context

In order to achieve high confidence, safety-critical cyber-physical systems must employ a variety of verification and validation techniques, including certification and standard processes. Formal methods have been proven to be a backbone for tackling complex problems such as specifying and reasoning functional behaviour of complex systems. They advocate for the development of a formal model that specifies the desired system behaviours as well as a set of required safety properties. There are several formal methods and tools available to support both system modelling and verification using model checkers (e.g., Promela/SPIN [Holzmann 1997], NuSMV [Cimatti 2002], Uppaal [Bengtsson 1996]) or interactive theorem provers (e.g., Isabelle/HOL [Nipkow 2002], Coq [Bertot 2010], PVS [Owre 1992]).

Addressing domain knowledge for safety-critical CPS is an another challenging problem. If we consider a system specification S and a set of property requirements R, then property verification consists in demonstrating that the requirements can be proven from the specification by establishing $S \vdash R$. In this case, the designed formal model associated to a specification S must make explicit all the knowledge required to write a specification, in particular the

domain knowledge provided by the domain and the context where the system is supposed to evolve i.e. S encapsulates the whole formal system description needed to establish R. As previously stated, the triptych approach, as well as other seminal works [Jackson 1993, Zave 1997, Bjørner 2017, Bjørner 2019], can be used to explicitly describe domain knowledge using $\mathcal{D}, \mathcal{S} \longrightarrow \mathcal{R}$.

In general, system engineering approaches, particularly formal methods, lack explicit constructs that enable the designer to define formal models of domain knowledge, as well as mechanisms for importing other existing models. In [Calegari 2016], the authors proposed formalisation of domain knowledge to standardised it using formal modelling languages and/or meta-models. Transformations are frequently required in the set up of formal method to reuse previously defined domain knowledge. As a result, heterogeneous formalisations emerge, posing a risk to sharing and reuse.

We believe that ontologies, as *an explicit shared specification of a conceptualisation* [Gruber 1993], meet the requirement of domain knowledge sharing and reuse. We advocate that domain knowledge should be formally modelled as datatypes theories with axioms, theorems and reasoning mechanisms, *once and for all*, in the system development formal method. In addition, we assert that this formalisation will have no impact on system modelling languages and models. Note that, in order to avoid semantic heterogeneity, ontologies, system specifications, and requirements must all be formalised in a single mathematical setting.

2.2.2 Our contributions

In this context, we propose to use the Event-B [Abrial 2010a] proof and refinement formal method to express both domain knowledge as ontologies formalised using Event-B theories, and system specification and requirements formalised as Event-B models (machines and invariants) [19]. As ontologies constructs are not present as first order concepts in Event-B, we introduce an Event-B meta-theory, based on generic and abstract datatypes and operators, describing an ontology model formalising an ontology modelling language (e.g. OWL [Antoniou 2004]), further instantiated to derive specific domain ontologies. These ontologies become *shareable*, *reusable* and *referenceable* by any Event-B model using typing, operators and properties guaranteed by proving both ontology instantiated theorems and Well-Definedness (WD) Proof Obligations (POs).

Core Event-B and Theory Extension

Event-B [Abrial 2010a, Abrial 2010b] is a modelling language based on set-theory that enables to model a system by supporting a *correct by construction* approach, which allows to design a complex system using stepwise refinement by introducing the required system behaviour and desired functionalities in a new refinement step. Each refinement step is verified by generated proof obligations corresponding to an abstract model and new refined behaviour. The stepwise modelling process finally lead to a concrete implementation of system. In the Event-B language, *context* and *machine* are two important components, which describe static behaviour and dynamic behaviour, respectively. The static properties can be described using *carrier sets*, *enumerated sets*, *constants*, *theorems* and *axioms*, while a machine can be described using *variables*, *invariants*, *events* and *theorems*. A list of events can be used to modify state variables by providing appropriate guards to characterise the dynamic behaviour of a system. In order to preserve the desired behaviour of a system, we define a list of safety properties using invariants and theorems. Moreover, to introduce the convergence properties in the model, we can use *variant* clause in a machine. In refinement step, an event can be refined by (1) keeping the events as it is; (2) splitting the event into several new events (3) strengthening the guards and actions (to make non-deterministic to deterministic). However, a new refinement level also allows to introduce a new event by modifying the new state variables.

To handle more complex and abstract concepts beyond set theory and FOL, an Event-B extension for externally defined mathematical objects has been proposed in [Abrial 2009, Butler 2013]. Formally, a theory is a set of type-generic *data-types* together with constructive and/or axiomatic *operators* and *properties*, encapsulated in a special component that can then be referenced in Event-B models.

Rodin [Abrial 2010b] is an open source, Eclipse-based Integrated Development Environment (IDE) for modelling in Event-B. It offers resources for model editing, automatic PO generation, project management, refinement and proof,

model checking, model animation and code generation.

Ontologies as Event-B Theories

We define our ontologies in Event-B using data-types based on set theory and first order logic using defined ontology modelling languages (OML) like OWL [Antoniou 2004] or PLIB [Pierra 2004]. Our approach proposes a formal parameterised theory that serves as a meta-theory for the OML, with each ontology described as a theory instance of this meta-theory. More precisely, the ontologies we use are based on an OWL[3] in which domain knowledge is formalised as collections of classes, properties, and instances.

Listing 2.1 shows an extract of `OntologiesTheory` theory allowing the formalisation of OWL-based ontologies. It is parameterised by `C`, `P`, and `I` which stand for classes, properties and instances. The `Ontology(C,P,I)` data type is built using the `consOntology` constructor based on seven components: `classes`, `properties`, `instances` (i.e. set of classes, properties and instances respectively), `classProperties` for associating classes to properties, `classInstances` for relating instances to classes, `classAssociations` defining a set of property-named binary associations and `instanceAssociations` for representing the associations between instances. To manipulate, access, and update an ontology, a set of operators is defined. To ensure correct use and the preservation of a valid ontology structure during instantiation, all defined operators are associated with well-defined (WD) conditions. When an operator is used, the generated WD proof obligations must be proven. As a result, depending on the OML used, Event-B theories allow for the modelling of complex domain knowledge. The developed theory is successfully proved in Rodin. The whole development of this theory is described in [43][19].

THEORY OntologiesTheory
TYPE PARAMETERS
C, P, I
DATATYPES
Ontology(C, P, I)
CONSTRUCTORS
consOntology(classes: ℙ(C), properties: ℙ(P), instances: ℙ(I), classProperties: ℙ(C × P), classInstances: ℙ(C × I), classAssociations: ℙ(C × P × C), instanceAssociations: ℙ(I × P × I))
OPERATORS
isWDClassProperites ...
getClassProperties ...
isWDInstancesAssociations ...
getInstanceAssociations ...
isWDOntology ...
instanceHasPropertyValuei ...
getInstancesOfaClass ...
...
THEOREMS
isATransitivityThm: $\forall o, c1, c2, c3 \cdot o \in Ontology(C, P, I) \land$
$c1 \in C \land c2 \in C \land c3 \in C \land$
$ontologyContainsClasses(o, \{c1, c2, c3\})$
$\Rightarrow (isA(o, c1, c2) \land isA(o, c2, c3) \Rightarrow isA(o, c1, c3))$

Code Snippet 2.1: Excerpt of ontologies theory OML

Domain Ontology for Critical Interactive Systems

The `DisplayabilityTheory` Event-B theory (IO ontology - Listing 2.2) is used to define a generic domain knowledge model for interactive objects (IOs) by instantiating the ontology theory (see Listing 2.1). It axiomatises a collection of specific operators with WD conditions entailing *displayability* properties of critical IOs. In addition, several new domain specific operators, such as `visible`, `hidden`, `critical`, `safe`, are defined as instances of `IOInstances`, with `hasVisibility`, `hasCriticality` as elements of `IOProperties`. All the defined operators are associated with

[3]https://www.w3.org/TR/owl-features/ theory

THEORY DisplayabilityTheory
IMPORT OntologiesTheory
AXIOMATIC DEFINITIONS
IOOntology
TYPES
IOClasses, IOProperties, IOInstances
OPERATORS
isIOOntologyWD . . .
visible . . .
hidden . . .
critical . . .
safe . . .
isVisiblei . . .
. . .
AXIOMS
...
axm5 : $\forall o, ipv, i \cdot o \in \ldots \Rightarrow (isVisiblei(o, ipv, i) \iff instanceHasPropertyValuei(o, ipv, i, hasVisibility, visible)$
...
axm18 : $\forall o, ipv, i \cdot o \in \ldots \Rightarrow (setCriticaliWD(o, ipv, i) \iff i \in dom(dom(ipv)) \land isVisiblei(o, ipv, i))$
axm19 : $\forall o, ipv1, ipv2, i \cdot o \in Ontology(IOClasses, IOProperties, IOInstances) \land ipv1 \in \mathbb{P}(IOInstances \times IOProperties \times IOInstances) \land ipv2 \in \mathbb{P}(IOInstances \times IOProperties \times IOInstances) \land i \in IOInstances \Rightarrow (ipv2 = setCriticali(o, ipv1, i) \iff ipv2 = (ipv1 \setminus \{i \mapsto hasCriticality \mapsto safe\}) \cup \{i \mapsto hasCriticality \mapsto critical\})$

Code Snippet 2.2: Exerpt of Displayability theory

WD conditions. Note that a large part of ARINC 661 [ARINC 2019] standard describing Cockpit Display Systems (CDS) interfaces used in all aircrafts has been formalised.

Further, the `Displayability Theory` can be instantiated in the Event-B context to define the specific IOs concepts and properties used in the HMI models. We have used the developed theories on TCAS case study described in the next section.

2.2.3 Application

The development of a critical interactive system (CIS): TCAS - Traffic Collision Avoidance System - demonstrates the importance of system design model annotation based on explicit formalised domain knowledge. TCAS is an airborne avionics system that serves as a last-resort safety net to reduce the risk of midair collisions. TCAS monitors aircraft in the surrounding airspace, using position data sent by their transponders to detect potential collisions. When an impedent collision is detected, TCAS sends a Resolution Advisory (RA) to the flight crews of the affected aircraft. These advisories instruct them to climb or descend at a specific vertical rate in order to avoid collisions [EUROCAE 2013, EUROCONTROL 2017]. TCAS computes a virtual protected volume that includes the positions of nearby aircrafts. This volume is affected by the aircraft's speed and trajectory. It is constantly updated. Some volume-related information is displayed on a cockpit screen for use by the flight crew.

We formalise the required domain model and system model of TCAS in Event-B. The domain model is developed based on theories of `OntologiesTheory` and `DisplayabilityTheory`. The developed models address safety critical properties such as TCAS must display the current status of all aircrafts on the PFD (Primary Flight Display) cockpit screen, and critical aircrafts must always be visible. This safety requirement is presented in Listing 2.3 from the extracted Event-B machine `TheoryOperatorsBasedModel`. A detailed description is presented in [43]. A set of new POs is generated while adhering to the defined domain model and is used in the development TCAS model. To ensure the correctness of the developed system models, all generated POs are successfully discharged.

There are several advantages to using Event-B theories to handle domain knowledge when developing domain models. It advocates 1) explicit modelling of domain knowledge using ontologies as a well-accepted formal modelling

MACHINE TheoryOperatorsBasedModel
SEES InstantiationContext
VARIABLES $system$
INVARIANTS
inv1 : $isVariableOfOntology(aircraftOntology, system)$
INITIALISATION
THEN
act1 : $system : |system' \subseteq instanceAssociation$
EVENT $CorrectAircraftStatusUpdate$
ANY i
WHERE
grd1 : $ontologyContainsInstances(aircraftOntology, \{i\})$
grd2 : $isVisibleWDi(aircraftOntology, system, i)$
grd3 : $isVisiblei(aircraftOntology, system, i)$
grd4 : $isSafeWD(aircraftOntology, system, i)$
grd5 : $isSafe(aircraftOntology, system, i)$
grd6 : $isWDSetCriticali(aircraftOntology, system, i)$
THEN
act1 : $\mathbf{system} := \mathbf{setCriticali}(\mathbf{aircraftOntology}, \mathbf{system}, \mathbf{i})$
...

Code Snippet 2.3: Ontology theory based annotated model

framework, and 2) separation of domain and system models. The proposition yields three important advantages in formal modelling state-of- the-art. Indeed, it becomes possible to 1) refer to (annotation) domain models concepts (types, operators, etc.), 2) automatically bring, in the system model, checking of well-definedness proof obligations for robustness purposes, and 3) allow asynchronous evolution of both domain and system models thanks to the separation of concerns.

Summary of our contribution to handle domain knowledge: We propose a uniform framework that integrates domain knowledge, system specifications, and safety requirements in a unique formal modelling setting and proof system offered by Event-B. This work demonstrates how explicit handling of domain knowledge, represented as ontologies, can benefit formal system design models. We develop an Event-B meta-theory describing an ontology based on generic and abstract datatypes and operators. To model both the system models and domain model, we use the Event-B modelling language and theories. Our main contributions include: developing meta-theories for ontology; and developing domain specific meta-theories for interactive systems. Finally, a complex avionic case study, TCAS, is used to evaluate the proposed approach.
Project: FORMEDICIS – Formal Methods for the Development and the Engineering of Critical Interactive Systems (funded by ANR)
Student supervision: Ismail Mendil (PhD, 2018 – 2023)
Publications: [38, 43][19]
Software: Domain specific meta-theories (ARINC 661), models
Models: https://www.irit.fr/~Ismail.Mendil/recherches

CHAPTER 3

System Modelling

This chapter covers the work in papers [20, 22–25][35, 36, 39, 41, 42, 44–48, 51–53, 55–58, 61, 65, 69–72, 74, 75, 77, 84][14, 15][8, 9]. This work was done in collaboration with Yamine Aït-Ameur (INPT-ENSEEIHT, France), Arnaud Dieumegard (IRT Saint Exupéry, France), Alexei Iliasov (Newcastle University, UK), Fuyuki Ishikawa (National Institute of Informatics, Japan), Eric Jenn (IRT Saint Exupéry, France), Tsutomu Kobayashi (Japan Science and Technology Agency, Japan), Mark Lawford (McMaster University, Canada), Thomas S. E. Maibaum (McMaster University, Canada), Dominique Méry (University of Lorraine, France), David Navarre (Toulouse III - Paul Sabatier University, France), Philippe Palanque (Toulouse III - Paul Sabatier University, France), Marc Pantel (INPT-ENSEEIHT, France), Alexander B. Romanovsky (Newcastle University, UK), Paulius Stankaitis (Newcastle University, UK), Hao Wang (Norwegian University of Science and Technology Gjøvik, Norway), and Alan Wassyng (McMaster University, Canada); and with following students: Guillaume Dupont (PhD student at INPT-ENSEEIHT, France under co-supervision of Yamine Aït-Ameur, Marc Pantel and myself), Romain Geniet (Master student at University of Rennes 1, France, under co-supervision of Yamine Aït-Ameur and myself), Yanjun Jiang (Master student at McMaster University, Canada under co-supervision of Thomas S. E. Maibaum and myself), Ismail Mendil (PhD student at INPT-ENSEEIHT, France under co-supervision of Yamine Aït-Ameur, Dominique Méry, Philippe Palanque and myself), Peter Riviere (PhD student at INPT-ENSEEIHT/IRIT, France under co-supervision of Yamine Aït-Ameur and myself), and Sasan Vakili (Master student at McMaster University, Canada under co-supervision of Mark Lawford and myself).

This chapter summarises our work on the fundamental concept of system modelling for designing safety-critical cyber-physical systems. Our main focus throughout this work was on developing generic formal frameworks, modelling patterns, architecture pattern, approximation patterns, refinement automation, simulation, environment modelling, and code generation, all with the goal of modelling, designing and verifying safety-critical cyber-physical systems using state-based methods like Event-B. All of these concepts are applicable to any type of system. Two refinement strategies are proposed to automate the process of formalising system requirements from tabular expressions using a correct-by-construction approach. Further, a generic and extensible reusable formal framework, consisting of a method and a set of tools, is proposed for developing and verifying critical systems. This framework consists of a large set of theories that extend Event-B with mathematical features required to model continuous behaviours (e.g., differential equations). Finally, three formal patterns for hybrid system design are defined: approximation, centralised control with multiple plants, and distributed hybrid systems. The development life-cycle for rigorous development of critical systems is proposed supporting requirement analysis to code generation. In addition, we propose a formal framework, F3FLUID (Formal Framework For FLUID), and the structuring of Event-B models using model-view-controller (MVC) to address the key challenges of safety-critical interactive systems. We also present the reflexive EB4EB framework, which enables users to explicitly manipulate Event-B features through the use of reflection and meta-modelling concepts. In addition, we propose the development of a virtual environment model, closed-loop modelling, a virtual environment model for verification, simulation and clinical trials, supporting techniques and tools. Finally, we propose a new extension of the code generation tool, EB2ALL, by developing a new plug-in, EB2Sol, for generating Solidity code from Event-B models for the Ethereum platform, as well as extending existing plugins (EB2C, EB2C++, EB2J, and EB2C#) to incorporate new modelling constructs. Several applications are also discussed to demonstrate our approaches.

In the remainder of the chapter, we sequentially describe our contributions to the development of automatic refinement, modelling & designing framework and patterns, meta-modelling reflexive EB4EB framework, environment modelling, and code generation.

3.1 Automatic Refinement

3.1.1 Context

Requirement engineering (RE) provides a framework for a better understanding of system requirements by simplifying system complexity using formal and informal techniques. It plays an important role in analysing system requirements, and functional and non-functional system behaviours to achieve the properties of consistency, unambiguity and completeness. Tabular expressions [Parnas 1992] support a technique for requirement engineering that uses (potentially complex) relations for documenting and analysing system requirements, in order to define them precisely and concisely. Tabular expressions have been used successfully in software development for more than thirty years. It is a visual representation of a function in a tabular layout that has a precise semantics and a formal notation. Moreover, this tabular representation of system requirements satisfies the important properties of *disjointness* and *completeness*. In other words, a tabular expression is well-formed only when the input domain is covered completely (completeness), and when there is no ambiguity in the behaviour described by the tabular expressions (disjointness). Unfortunately, it is difficult to use tabular expressions in a straightforward way to check the correctness of functional requirements of combined tables and to check the given safety properties of a system.

On the other hand, Formal methods play a significant role in verifying the system requirements, and in guaranteeing the correctness, reliability and safety of developed system software. These methods have been applied successfully to design and develop critical systems, such as avionics, medical and automotive [1][Lee 2006, Bowen 1993][94]. In particular, formal methods have been used to check functional requirements and safety requirements by developing system models. There are many tools that support formal methods in specific, limited areas, such as C code analysis using different tools, for example, CBMC [Kroening 2014], BLAST [Beyer 2007], and Frama-C [Cuoq 2012], open source operating system (OS) microkernel verification using Isabelle/HOL [Klein 2009], model checking tools for SCADE and Simulink models, compiler certification using Coq (CompCert) [Stewart 2015], development and verification of a specification using B [Lecomte 2017, Lecomte 2007] or VDM [Overture , Jones 1986]. In formal modelling, refinement plays an important role for handling a large complex system by developing the whole system incrementally, in which each incremental step can be used to introduce new functionalities while preserving the required safety properties. Note that most formal methods lack refinement processes, so we must design the entire model once, which can be a very large and complex model with difficult to check all the required properties.

Practicalities of performing automatic refinements are largely an open problem. It is, clearly, unrealistic to carry out such refinements entirely by hand, which is well illustrated by the complexity of the examples in [Iliasov 2010, Kobayashi 2014][74]. Some refinement steps are, however, inherently difficult to automate. Our work highlights how automation to guide the refinement process is feasible.

3.1.2 Our contributions

In this context, we propose two different refinement strategies that can help to automate the process of formalising system requirements from tabular expressions using a *correct-by-construction* approach [61][25][8][102].

The proposed refinement strategies can be used to generate both a single formal model and multiple formal models from tabular expressions. The single formal model contains a formal description of the system requirements without using refinement, while the multiple formal models contain an abstract model and a series of refinement formal models of the given system requirements. We show how the refinement strategies can be used to transform tabular expressions into formal models that aid in determining the correctness of functional behaviour and modelling structure of a system. In particular, we discuss how different stages of the refinement may take advantage of automation. The refinement strategies directly reflect particular refinement based model designs that encapsulate the way in which system is refined from abstract to concrete behaviours to meet safety properties, which is how refinement laws of Event-B are applied. The refinement approach allows us to build a formal model incrementally, where the first model represents only abstract behaviour, and the incremental models are enriched by more concrete behaviours. The generated formal models are used later to define safety properties and to check system consistency using formal verification. We chose to use the Event-B modelling language, because it allows an incremental refinement approach based on a *correct-by-*

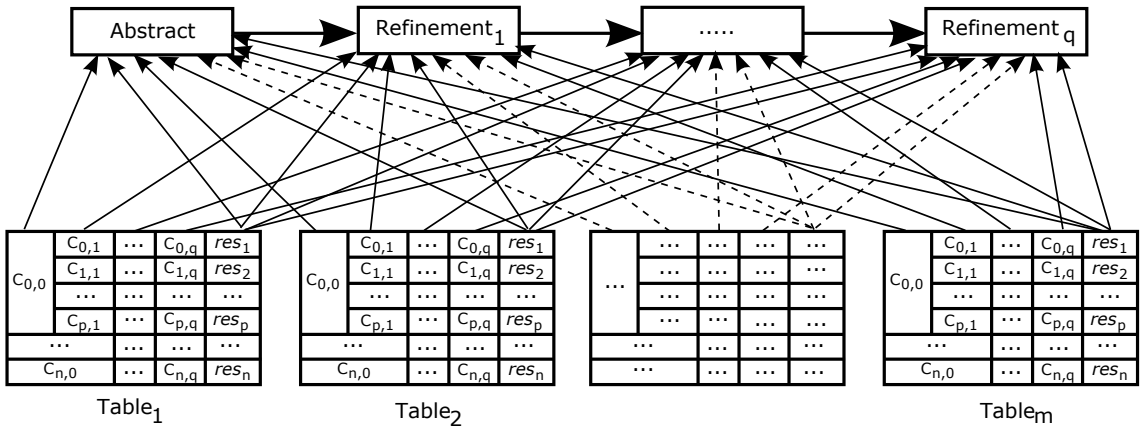

Figure 3.1: First Refinement Strategy

construction that facilitates generating formal models from tabular expressions. The proposed refinement strategies, and the main results derived from them, are presented in the following section.

Refinement Strategies

A common way of constructing a formal specification is to start from a very simple abstract model that captures only basic system behaviour, and to add new features or system requirements to the abstract model to develop a concrete system by satisfying the additional requirements. To find a correct abstract model is not easy. Often we need to change an abstract model many times, so that an extension of a system, by providing concrete details, satisfies the abstract requirements. An extension is a set of new features and system requirements that always zoom into a detailed system behaviour without changing the original abstract behaviour. We refer to this type of modelling method as *superpositioning* [Back 1996].

Superposition seems to be a good candidate in the field of formal modelling, because it allows us to construct a complicated formal specification by incremental refinement steps. Each new refinement step always focuses on a single design decision. In other words, it permits us to tackle one issue at a time, rather than having to make a joint design decision and settle a number of interrelated design questions [Back 1996].

This section describes two new refinement strategies for generating formal models from documented system requirements. The documented system requirements are described in tabular expressions that satisfy the *completeness* and *disjointness* properties. Our objective is to formalise tabular expressions and then define safety properties for the developed models to verify the documented system requirements. To produce formal models from tabular expressions is not an easy task due to there being no refinement relation between the tables, lack of techniques to support table compositions and implicit information about correct ordered system behaviour. In order to produce formal models from tabular expressions, we propose two refinement strategies that allow us to construct a model progressively by traversing tabular requirements using a *correct-by-construction* approach. The proposed strategies are suitable for any formal language that can support refinement based development. As mentioned, we use the Event-B modelling language, which supports refinement based progressive development. A formal definition of the transformation rule for the proposed refinement strategies is given below.

Definition 1 *Let T be a set of tabular expressions, each of which satisfies the properties of disjointness and completeness. Then a transformation rule R is a function that produces a collection of machines M, defined in Event-B language syntax for the given input tabular expressions T:*

$$R : T \rightarrow M$$

Note that R is defined as a total function, which means that for each input table t of T, it generates an Event-B model m of M, i.e., when $t \in dom(R)$.

The following refinement strategies are based on formulating our behaviour specifications in HCTs with a predefined structure, namely condition columns, as shown in Fig. 3.1. Here we are stipulating that all rows have multiple columns, and the same number of columns. This is easy to achieve, even if it may look clumsy.

First Refinement Strategy

The first refinement strategy is depicted in Fig. 3.1. To construct an abstract model and successive refinement models, the refinement strategy for producing formal models from tabular expressions takes into account system requirements defined in tabular expressions. A formal definition of the transformation rule using the first refinement strategy for producing formal models from tabular expressions is defined below.

Definition 2 *A refinement strategy is a transformation rule $R_1 : T \rightarrow M$ that constructs Event-B models M_e for input tabular expressions. The generated model M_e is defined as,*

$$
\begin{aligned}
&M_e = AM \sqsubseteq CM_1 \sqsubseteq CM_2 \sqsubseteq \cdots \sqsubseteq CM_q \\
&AM = \forall t \in T, \Pi_{c_0,res}(t) \\
&CM_1 = AM \cup (\forall t \in T, \Pi_{c_1,res}(t)) \\
&CM_2 = CM_1 \cup (\forall t \in T, \Pi_{c_2,res}(t)) \\
&\ldots \\
&CM_q = CM_{q-1} \cup (\forall t \in T, \Pi_{c_q,res}(t))
\end{aligned}
$$

where AM is an abstract machine, $CM_1, CM_2, \ldots, CM_q$ are a series of concrete machines, Π is a projection relation to select table's column defined later for event derivation, $c_0, c_1, c_2, \ldots, c_q$ are columns of the table (t), res is the set of output columns, and $\sqsubseteq$ denotes a refinement relation. It is important to know that each table t of T contains a set of required variables, including type definition, to describe system requirements in tabular form, which can be used during the process of model generation ($AM, CM_1 \ldots CM_q$).

Deriving abstract model. A formal model generated using the first refinement strategy consists of an abstract model and a list of refined models. An abstract model is important because it tells us exactly what the system is supposed to do without telling us how. An abstract model can be produced from a set of tabular expressions by observing the first condition column and the output column of each table by defining the required variables, constants, and events. Each row of each table is an event. The guard of that event is the first column in that row, and the action of that event is the result column. If there are multiple results rows for the condition column row, they are combined with a disjunction. As a result, the event's actions are either deterministic or non-deterministic.

We can use this approach iteratively to traverse the first condition column and output column to create a list of events from a set of tabular expressions to achieve the first initial model AM. This first initial model covers the first condition column and output column of all the tabular expressions that is formally defined in Definition 2 as $AM = \forall t \in T, \Pi_{c_0,res}(t)$.

Deriving refined models. The abstract level events are based on the first condition column that can be refined in this refinement level by splitting the abstract event into more events to equal number of rows of the second column. We can now generate a set of successive refinement levels by analysing the remaining condition columns and output column of tabular expressions, which are defined in Definition 2 as $CM_1, CM_2, \ldots, CM_q$. These refinements are based on events: (1) keeping an event as it is; (2) splitting an event into more than one event; and (3) guard strengthening of an event. These three kinds of refinements are used in the transformation rule R_1. Guard strengthening allows us to enrich guard predicates, in which a concrete event must be enabled only if the corresponding abstract event is enabled. Action simulation enables to make an action deterministic, which means that if a concrete event's action assigns a value to a variable that is also declared in the abstract machine, it must be proven that the concrete event's behaviour corresponds

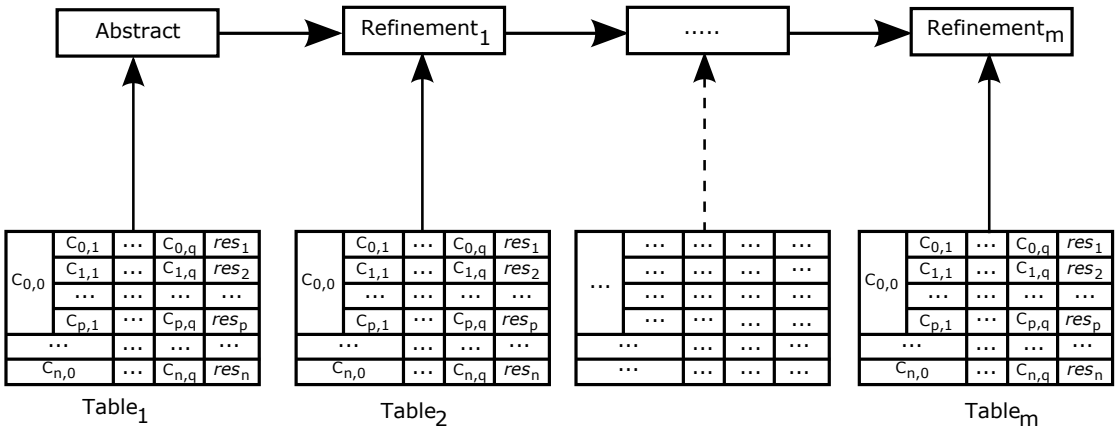

Figure 3.2: Second Refinement Strategy

to the abstract behaviour. Instead of creating a new event in a refinement level, an abstract event can be extended. In this case, the refined event contains all the guards and actions of the abstract event. The new refined event can be used to add new guards and actions, and this specific technique is useful when additional features or system requirements are gradually introduced into a model by refinement. By applying these simple refinement rules on the abstract model, we can construct a set of successive refinement models from tabular expressions. To obtain the first refinement model, we observe the second condition column and respective output column defined as $CM_1 = AM \cup (\forall t \in T, \Pi_{c_1,res}(t))$ in Definition 2. In Fig. 3.1, the first refinement, CM_1, derives from the second column of each tabular expression. The given conditions of the second column become new guard predicates of the refined events, and these new guards allow strengthening of the abstract guards. If the action is already deterministic, the action of each refined event will be similar to the action of the abstract event. If a refined event's action is non-deterministic, the new action will be either deterministic or non-deterministic. Note that if an output column groups more than one row with distinct values, the action will always be non-deterministic.

To obtain the next refinement levels, we can repeat a similar process as per the total number of condition columns to construct the final concrete model from tabular expressions. To do this we can use the same refinement principles related to guard strengthening and action simulation. It is important to know that each row can have one or more columns, but for applying this approach to produce a formal model, we need to repeat the iterative process of adding information from condition columns until all the condition columns are covered. The abstract model and total number of refinements will be equal to the maximum number of columns. In the last refinement, we can get all the possible rows from each table, where each row results in an event. The guards of each event contain possible conditions, and the actions of each event contain possible output values.

Second Refinement Strategy

The second refinement strategy also generates formal models from tabular expressions, but this approach has two benefits: first, this approach can be used for developing Event-B models in an "on-the-fly" manner; and second, this approach is an effective approach for formalising a very large and complex system, if the system requirements are presented in several tables. In this approach, we observe each tabular expression in a sequential order to construct a concrete formal model. A formal definition of the transformation rule using this refinement strategy for producing formal models from tabular expressions is defined below.

Definition 3 *A refinement strategy is a transformation rule $R_2 : T \rightarrow M$ that constructs Event-B models M_e for input tabular expressions. The generated model M_e is defined as,*

$$M_e = AM \sqsubseteq CM_1 \sqsubseteq CM_2 \sqsubseteq \cdots \sqsubseteq CM_m$$

$AM = t_0 \in T, \Pi_{c_{0..p},res}(t_0)$
$CM_1 = AM \cup (t_1 \in T, \Pi_{c_{0..q},res}(t_1))$
$CM_2 = CM_1 \cup (t_2 \in T, \Pi_{c_{0..r},res}(t_2))$
...
$CM_m = CM_{m-1} \cup (t_m \in T, \Pi_{c_{0..s},res}(t_m))$

where AM is an abstract machine, $CM_1, CM_2, \ldots, CM_n$ are a series of concrete machines, Π is a projection relation to select table's column, $t_0, t_1, t_2, \ldots, t_m$ are a set of tables, $c_{0..p}$, $c_{0..q}$, $c_{0..r}$, ..., $c_{0..s}$ are a set of columns of different tables (i.e., t_0, t_1), res is a set of output columns and $\sqsubseteq$ denotes a refinement relation. It is important to know that each table t of T contains a set of required variables, including type definition, to describe system requirements in tabular form, which can be used during the process of model generation ($AM, CM_1 \ldots CM_n$). Note that this translation strategy only works if the tables do not share variables, which means that output variables should not be modified by two tables; otherwise, all of these tables must be included in the same refinement level.

Deriving abstract model. In this approach, we can start to design an abstract model from any table, and then we can select other tabular expressions in a sequential order to introduce a new system behaviour by applying refinement laws, and preserving abstract behaviour. In Fig. 3.2, each tabular expression is introduced at a new refinement level that is defined in Definition 3 as $CM_1 = AM \cup (t_1 \in T, \Pi_{c_{0..q},res}(t_1))$. The *skip* refinement allows us to introduce other events to maintain state variables. Importantly, a new refinement level allows us to introduce a set of new events. In this refinement strategy, we do not use any guard strengthening and action simulation refinement laws, like to our first refinement strategy. By using *skip* refinement, we introduce a new set of events corresponding to the tabular expressions. To design a formal model from tabular expressions, we traverse a tabular expression, in which condition columns are used for defining the guard predicates and output columns are used for defining actions of the events. Each row of a tabular expression that corresponds to the last condition column generates an event.

Deriving refined models. At each refinement level, we always select a new tabular expression to introduce new features and system requirements. It should be noted that the total number of refinements will depend on the total number of tabular expressions, and sometimes a small number of tables can be formalised together. In fact, if each table in a set of tables produces the same output variable, we must input all these tables (have same output variables) together in the refinement process. Moreover, to satisfy the refinement relation between two consecutive models, to develop a consistent model, and to prove all the generated proof obligations related to refinement, we need to identify a dependency order between the tables that can be used further to generate the formal models. In order to address this issue, we chose an ad hoc approach that takes into account all related tables in a single refinement step. However, there are other solutions like merging multiple tables, providing dependency information in advance, and constructing tables without sharing results variables in the output column.

Note that our above proposed refinement strategies that can be used for developing different types of systems are significantly different from each other. A list of differences is given in Table 3.1, which can assist software engineers in choosing a suitable refinement strategy for their system development.

Safety Properties

Informally, a safety property stipulates that *"bad things"* do not happen during system execution. A formalised specification that satisfies a safety property involves an invariance argument. It should be noted that the generated formal model in Event-B only includes type invariants for each refinement, which are derived directly from the selected tabular expressions. There are no required safety properties in the generated models. We can introduce local safety properties, global safety properties, or both to check the required safety behaviour in the generated model. These safety properties must be added manually because they are completely missing from the tables. However, we can provide additional safety properties for each table that can be used as safety invariants for each refined model, which can be generated automatically.

First Refinement Strategy	Second Refinement Strategy
(1) Complete abstract model	(1) Partially abstract model
(2) Only few refinement steps (total number of condition columns)	(2) Many refinement levels (total number of tables)
(3) High complexity	(3) Low complexity
(4) Good for developing a small system	(4) Good for developing a large system
(5) Analyse global behaviour of a system	(5) Analyse specific behaviour of a system
(6) Introduce only global safety properties	(6) Easy to introduce local or table specific safety properties
(7) Does not support "On-the-fly" development	(7) "On-the-fly" development of Event-B models

Table 3.1: Difference Between Refinement Strategies

In our work, every tabular expression satisfies the properties of *disjointness* and *completeness*, which can be checked by our tool TX2EB [Jiang 2015] before generating the Event-B models. The Event-B model should satisfy the same properties, i.e., disjointness and completeness by construction (at the last level of refinement), hence we do not generate POs corresponding to these properties in the Event-B models. In addition, we want to check the correctness of global functional behaviour as well as table-specific function behaviour, which can be provided by introducing global and local safety properties, respectively.

Formal Model Generation from Tabular Expression

In this section, we provide a list of translation rules that are sufficient for translating tabular expressions into Event-B models. We start with a formal definition of the transformation rule (see Definition 1) that is adopted in this work for generating formal models from tabular expressions. The refinement strategies are defined in Definition 2 and Definition 3. The formal definition is used to traverse the tabular expressions systematically to identify the events, actions, guards and associated variables.

The translation process generates formal models from tabular expressions as a Rodin dependent project, which contains *context* and *machine* files together in a project folder. A name for the generated Rodin project is derived from the name of the tabular expressions. The context and machine files are used to store the static and dynamic properties of a given system that is generated from system requirements documented in tabular expressions. A set of supported symbols, datatypes and expressions of the translation process is given in Table 3.2, which shows a set of tabular expression syntaxes equivalent to the Event-B modelling language. A detailed discussion on the translation rules is available in [Jiang 2015]. A brief description of the translation rules for transforming the tabular expression components into Event-B modelling components is given below:

- **Tabular Definition:** Table 3.2 presents the translation rules for tabular expressions. Each table mainly consists of two parts: *variable declarations* and *function definition*. The declared variables can be categorised as input, output, controlled, global, monitor, private and local types. The function definition shows the condition and result columns in tabular form. We consider these two components in the model generation process to generate formal models.

- **Variable:** The defined *variables* of tabular expression are parsed and translated to Event-B variables, in which all the variable names are defined in the VARIABLE clause of the *machine*. Our treatment of all input, output, controlled, global, monitor, and private variables is the same, and the local variables are defined within the context of the event for simple intermediate computation. To determine variable classes and maintain a direct relationship between tabular expression and Event-B variables, all generated variable names are prefixed according to the naming convention.To determine variable classes and maintain a direct relationship between tabular expression and Event-B variables, all generated variable names are prefixed according to the naming convention. These prefixes are: $c\_$ - controlled variables, $f\_$ - internal functions, $g\_$ - global variables, $i\_$ - input variables, $k\_$ - constant, $m\_$ - monitored variables, $o\_$ - output variables, $p\_$ - private variables, $y\_$ - complex type and $e\_$ - enumerated token. All of these prefixes are followed by a variable name.

Tabular Expression	Event-B Language		Comments
$x\ OR\ x : int$	x	(VARIABLES)	Datatype
	$x \in \mathbb{Z}$	(INVARIANTS)	
	$x :\in \mathbb{Z}$	(INITIALISATION)	
$x : bool$	x	(VARIABLES)	Datatype
	$x \in BOOL$	(INVARIANTS)	
	$x :\in BOOL$	(INITIALISATION)	
$x : ESet$	$partition(ESet, \{a_1\}, \{a_2\}, \{a_3\})$	(AXIOMS)	Enumeration Type
	x	(VARIABLES)	
	$x \in ESet$	(INVARIANTS)	
	$x :\in ESet$	(INITIALISATION)	
$x : \{t : int \mid t > 0\}$	x	(VARIABLES)	Subtype
	$x \in \{t \mid t \in \mathbb{Z} \wedge (t > 0)\}$	(INVARIANTS)	
	$x :\in \{t \mid t \in \mathbb{Z} \wedge (t > 0)\}$	(INITIALISATION)	
x_{-1}	x		Previous State
$x + y$	$x + y$		Addition
$x - y$	$x - y$		Subtraction
$x * y$	$x * y$		Multiplication
x/y	$x \div y$		Division
$x \sim= y$	$x \neq y$		Not Equal To
$x == y$	$x = y$		Equal
$x = y$	$x := y$		Assignment
$x > y$	$x > y$		Greater Than
$x >= y$	$x \geq y$		Greater Than or Equal To
$x < y$	$x < y$		Less Than
$x <= y$	x$\leq y$		Less Then or Equal To
$\sim x$	$\neg(x = TRUE)$		NOT
$x\&\&y$	$(x = TRUE) \wedge (y = TRUE)$		AND
$x \| \| y$	$(x = TRUE) \vee (y = TRUE)$		OR
$EXIST\ [st], st \in bool$	$\exists\, x.x \in BOOL \wedge x = TRUE \wedge x = st$		Existential Quantifier

Table 3.2: Tabular Expressions (tables) to Event-B translation syntax

– **Variables Types:** Table 3.2 shows the translation rules for different types supported by tabular expressions. We categorise these types as : 1) primitive datatypes (integer, reals, booleans); 2) enumerated types; and 3) arrays. For primitive datatypes and enumerated types, we can easily find the direct corresponding types in Event-B. The array type in Event-B is essentially a function with a range of integers for the domain and a proper range type. Note that the user defined type is not covered in this paper, but it is easily supported by tabular expressions and Event-B. All these datatypes are defined in the *context* of a model as static properties using the AXIOMS clause, or it can be defined in the *machine* of a model as dynamic properties using the INVARIANT clause.

  The user defined *enumeration class* of a tabular expression is translated as the *enumerated set* of Event-B context. The *enumerated type* is translated as the *carrier set*, the *enumerated values* are translated as the *constants*, with an axiom added to show the partition relation.

– **Event:** To define a list of events from a tabular expression, we select each row of the table to translate equivalent to an ordinary event (all non-initialisation events are called ordinary events) in an Event-B model. The expression in a table cell is parsed into an abstract syntax tree (AST), which is then recursively translated to an Event-B expression. However, an initialisation event is mandatory in an Event-B model, so this event can be generated using deterministic or non-deterministic action predicates without any guard.

– **Event Name:** An event name of the generated model is always derived from the cell number of a grid and

sub-grid of a tabular expression.

– **Event Guard:** Table 3.2 presents the translation rules for guard predicates. If the current mode is not a refinement, the event guards are generated by traversing the conditions from root grid (cell in the first column) to leaf grid (cell in the last column). In refinement mode, the event guards are also generated by traversing the conditions from root grid to leaf grid using refinement strategy (see Definition 2 and Definition 3). According to Definition 2, each condition column represents a new refinement layer, and according to Definition 3, each new table represents a new refinement layer. Note that sometimes we need to use more than one table in the same refinement level due to dependency between tables.

– **Event Action:** Table 3.2 presents the translation rules for action predicates. If the current mode is not a refinement, an event action is generated through parsing the result column. In the multiple outputs mode, each action is generated from left to right through parsing, and its name is encoded as *actionX*, where X is the top grid column number (see Definition 1, Definition 2 and Definition 3). However, in refinement mode, the event action is slightly different and can be generated in the form of deterministic and non-deterministic action predicates. In the last refinement, the non-deterministic actions become deterministic as similar to the "no refinement mode" by introducing new guards and refining abstract events.

– **Expressions and Predicates:** Table 3.2 presents the translation rules for expressions and predicates, which can be used by both the guard predicates and action before-after predicates. These expressions and predicates use unary and binary operators to express the rules. In the case of operations, for all the unary and binary numerical expressions, relational expressions and logical expressions, we can find the obvious corresponding operators in Event-B. In case of assignment in event's action, there is an assignment operator in tabular expressions to form variable assignment statement, which can directly translate into action predicate using Event-B's assignment operator.

To support this refinement strategies for producing formal models from tabular expressions, we have developed a model generation tool TX2EB [Jiang 2015]. A detailed discussion on the translation rules is available in [Jiang 2015]. This tool generates an abstract model and a series of refinement models from tabular expressions automatically. Note that all the input tables must be successfully parsed and checked with TET tool [Eles 2011]. The generated formal specification can be analysed by the existing tool, Rodin [Abrial 2010b], for verifying the required functional behaviours and the given safety properties. Note that the safety properties can be added by a user for each refinement in the generated model.

3.1.3 Applications

The application of the refinement strategies exemplified by the development of several complex case studies, including the Insulin Infusion Pump (IIP) [25][9]. The refinement strategies are used to formalise and to formally verify an Insulin Infusion Pump (IIP) using incremental refinement in Event-B. The IIP requirements are described in tabular expressions that are used to produce the formal models. In the IIP case study, we verify functional behaviours including various system operations, that are required to maintain insulin delivery, user profile management, and the calculation of required insulin. The complete formal development builds incrementally-refined models of IIP formalising the required functional behaviour by preserving its required safety properties. The primary use of the models is to assist in the construction, clarification, and validation of the IIP requirements. We also use the Rodin [Abrial 2010b] tool to check the generated formal models. To automate the task of refinement strategies, we have used our developed tool TX2EB [Jiang 2015], which allows us to generate Event-B models from tabular expressions. The complete formal specification is available for inspection in the appendix of report [102], which is more than 1500 pages long. Moreover, this case study is elaborated in Chapter 5.

Summary of our contribution for automatic refinement:
We propose two refinement strategies that can automate the process of formalising system requirements from tabular expressions using a *correct-by-construction* approach. We use these refinement strategies to transform tabular expressions into formal models that determine the correctness of functional behaviour and modelling structure of a system. We also highlighted challenges for automation: primarily, composition of tabular expressions, use of sequential ordering of tables, and table traversing complexities. Due to the variety of layouts of tabular expressions, there are still open issues related to the automation of tables that ought to be supported, and hence we do not claim completeness at this stage. On the other hand, our results showed that the proposed refinement strategies can largely be automated to generate formal models from tabular expressions. Moreover, the proposed approach is scalable to handle large and complex systems, in which system requirements are presented in tabular form. In order to apply the proposed refinement strategies, we selected the Event-B modelling language, which allows incremental refinement based on a *correct-by-construction* approach, for generating formal models from tabular expressions. Further, the Rodin tools can be used to verify formally the produced model. To assess the effectiveness of our proposed refinement strategies, we have developed several case studies from different domains.
Project: Certification of Safety Critical Software-Intensive Systems (funded by Ontario Research Fund – Research Excellence (ORF-RE), and IBM), Centre for the Engineering of Complex Software-Intensive Systems, NECSIS (funded by Automotive Partnership Canada (APC))
Student supervision: Mischa Geven (M.A.Sc., 2014), Nicholas Proscia(M.A.Sc., 2014)
Publications: [61][25][8][102].
Software: TX2EB, models

3.2 Modelling & Designing Frameworks and Patterns

3.2.1 Context

As our lives become increasingly reliant on various types of embedded systems, there is a growing demand for safety-critical cyber-physical systems that improve reliability, safety, performance, and autonomy. Such systems cover a wide range of domains such as avionics, transportation, nuclear, and medical applications, among others. Today's critical systems employ highly sophisticated interactive systems made of both hardware and software components, as well as their complex dynamics exhibit both continuous and discrete behaviour while interacting with the physical environment via sensors and actuators. Moreover, human safety is a global challenge that requires practical knowledge and technical skills in embedded systems, and software engineering including human factors and systems engineering [Carayon 2009]. To ensure the safety of operations, the intended behaviour of these critical systems must comply with usability, dependability, and security. A failure in these systems could result in loss of life, including reputation and economical damage. Developing such complex critical systems was known to be a difficult and time-consuming task in the 90s [Myers 1993] and this has become even more complex due to complex system characteristics and user requirements capturing key aspects of human behaviour, system components, functionalities, and operating environment. Beyond that, the complexity lies in gathering data about operators such as requirements, needs, functionalities and tasks [Maiden 1993], particularly because different application domains necessitate different methods and techniques [Sutcliffe 2020]. Furthermore, there is an increasing demand for developing embedded safety-critical cyber-physical systems to improve reliability, safety, performance and autonomy. Traditional techniques, like testing and simulation, become more crucial, time-consuming and expensive when used to deploy safety-critical cyber-physical systems. Verification and validation activities can be carried out to aid in the certification process, ensuring that the developed product is safe for use in the real world.

Since a long time, formal methods play an important role for the modelling and analysis as well as ensuring the correctness of functional behaviour by checking system requirements [Gray 1999, Harrison 1990]. As a result, the formal methods community has proposed a number of modelling techniques and tools to handle different issues related to designing safety-critical systems. Since software plays an important role in different safety-critical do-

mains, regulatory agencies, like the FDA, need effective means to evaluate the software embedded in the devices in order to certify the developed systems, and to assure the safe behaviour of each system [Chen 2014, Keatley 1999, NITRD 2009, Lee 2006]. These methods are divided into two categories: *model checking* and *proof-based* approaches. However, model checking approaches always suffer from the classical state explosion problem. Proof-based approaches, on the other hand, are based on proof techniques and symbolic verification and can be used to characterise any type of critical systems. There are many tools that support formal methods in specific, limited areas, such as C code analysis using different tools, for example, CBMC [Kroening 2014], BLAST [Beyer 2007], HyTech [Henzinger 1997] and Frama-C [Cuoq 2012], open source operating system (OS) microkernel verification using Isabelle/HOL [Klein 2009], model checking tools for SCADE and Simulink models, compiler certification using Coq (CompCert) [Stewart 2015, Bertot 2010], development and verification of a specification using Event-B [Su 2014b, Butler 2016] and B [Lecomte 2017, Lecomte 2007] or VDM [Overture , Jones 1986]. Regulatory agencies are striving for rigorous techniques and methods to provide safety assurance. We also note that many formal techniques need to be much better targeted at practical software development and certification than they seem to be at present [Wassyng 2013].

There are a few challenging case studies that have been developed through scaling formal methods approaches. In [Jeannin 2015], the authors present a formal verification of ACAS X, an airborne collision avoidance system, using a hybrid systems theorem prover. In [Klein 2009], the authors present the formal, machine-checked verification of the seL4 microkernel. This approach is applied to produce a C code implementation from an abstract specification under the assumption of the correctness of hardware, assembly code and compiler. Miller et al. [Miller 2010] present an example of using model checking techniques in the model-based development of avionics and automotive systems to demonstrate the applicability of formal methods in industrial settings. In a similar vein, [Souyris 2009] presents an experience in the formal verification of avionics software products using different tools that can scale to industrial requirements. Prover iLock [iLock] is a commercial tool for producing fully documented, tested and verified application software for railway interlocking systems. C.L. Heitmeyer et al. [Heitmeyer 2009, Heitmeyer 2008] have used formal methods to meet the criteria for software certification. They describe how formal methods are applied to safety-critical software systems for the production of certification evidence. The evidence includes top-level specification, functional behaviour, safety properties, proofs and source codes annotated with pre-and post-conditions. In [Heitmeyer 1996], the authors describe a technique for formal analysis of consistency checks, and error detection in requirement specifications. The specifications for the requirements are expressed in the tabular notation SCR (Software Cost Reduction).

The software life-cycle was introduced during the 1950s and 1960s to support the design and development of a software system that primarily involves software development activities related to planning, organising, coordination, staffing, budgeting and management [Scacchi 2002]. Since the 1960's, several kinds of software development life-cycle models have emerged, such as the waterfall model [Boehm 1976], stepwise refinement model [Wirth 1971], incremental model [Sommerville 2004], spiral model [Boehm 1988] , Agile Software Process [Cockburn 2006], and V model [Sommerville 2004], motivated by increasing system complexities, rapid development and to tackle different classes of critical and non-critical systems [Boehm 1976, Sommerville 2004]. Formal verification of industrial systems is very challenging and difficult. It can be applied to validate only certain operations of the selected system due to *limits of formalisation* and *practical limits* [Kneuper 1997].

A research report [Rushby 1995] is presented by John Rushby that describes the certification issues for advanced technology. This report summarises the use of formal methods for developing and verifying the software and hardware requirements, designs, and implementations. Moreover, it includes the benefits, weaknesses, and difficulties for applying these methods for developing critical systems, including guidelines for applying formal methods in support of certification for critical systems. [NITRD 2009] presents an approach for developing a final product using intermediate steps, where we need several methods and techniques that can provide intermediate products in the form of evidence, facts and reports to aid certifying the final product. For example, requirement specification, formal design, verification report for checking the requirements and testing reports are required to assist in certifying critical systems. Regulatory agencies are striving for rigorous techniques and methods to provide safety assurance. Many people believe that formal methods have the potential to develop dependable, safe and secure systems that are also more amenable to certification with required features that can be used to certify dependable critical systems.

In fact, the scope of formal methods is limited in the current software development process in industries, particularly, for developing safety-critical systems. This is due to the complex nature of formal methods and software engineers require high mathematical skills for developing a specification and conducting proofs. Rigorous verification and validation techniques must be enabled in the engineering process of designing complex and safety-critical systems. In this direction, we believe that two major requirements must be addressed in order to handle this challenge. First, compositional and incremental design, verification, and validation techniques must be defined in order to handle the complexity of such systems. A critical issue that still needs to be addressed is the construction of incrementally correct systems through the composition/decomposition/refinement of other correct components. Second, these designs must be reusable to avoid redundant verification and validation activities. This requirement refers to the definition of a reusable framework for the development of critical systems. A systematic procedure that allows a designer to produce correct system designs must be established. Furthermore, there is a crucial need for new methods and better tool support for the development of safety-critical systems and to guarantee the functional correctness of the developed systems. In addition, there is also a need for a new framework to address safety and security issues related to the design and engineering of complex safety-critical systems. None of the existing life-cycles models uses formal methods at every phase of the system development, as well as there is no environment model for critical systems, which can be used for simulating and testing the system requirements at an early stage of the system development during design and development. The availability of new methods and tools could significantly save time and cost for developing and certifying safety critical systems.

3.2.2 Our contributions

In this context, we propose a generic formal framework [20, 23][41, 44–46, 48, 53, 55, 56], development life-cycle [25], F3FLUID (Formal Framework For FLUID) formal framework, and Event-B model structuring pattern using *model-view-controller*(MVC) [22, 24][42, 47, 51, 58] for modelling, designing and verifying various classes of critical systems. The proposed framework is generic and extensible, and it consists of a method and a set of tools that engineers can apply and use when developing such systems. These frameworks and patterns includes a large set of theories that extend Event-B with mathematical features required to model discrete and continuous behaviours, and a series of patterns based on refinement that allow for easier design. The formal frameworks, patterns and results are summarised below.

A Generic Formal Framework for Designing Hybrid Systems

Fig. 3.3 depicts a generic formal framework, derived from refinement patterns, to design complex hybrid systems [20]. This framework addresses two common design patterns: structural and behavioural. The structural patterns are defined to model different class of architecture patterns: *single-to-single*, *single-to-many*, and *many-to-many*. These architecture patterns are used to represent one controller with one plant (*single-to-single*) [55, 56], one controller with many plants (*single-to-many*) [48], and many controllers with many plants (*many-to-many*) of hybrid systems [46]. On the other hand, the behaviour patterns allow the system's behaviour to be formalised by approximating continuous behaviour to obtain a real behaviour of the system under the given constraints [44, 46].

There are four main components in the proposed framework: theories, extended theories, generic, and system specific, which are separated by horizontal and vertical lines. Theories is a collection of mathematical theories to define reals, continuous functions, and differential equations for designing hybrid systems in the top-left corner. This collection is also enriched by the addition of approximation theories. Furthermore, in the bottom-left corner, the domain-specific theories are provided that are extended from the core theories. We provide a set of generic architecture patterns and approximation models in the top-right corner, which are equipped with basic variables, parameters, and bare-bone functionalities (i.e., sensing, actuating) for designing complex hybrid systems using the refinement approach. Note that these generic models use theories to define core components of the generic models. Finally, the system specific component in the bottom-right corner can be used to derive a system specific abstract model by instantiating generic models. Domain specific theories are also used in the system specific formal model to express domain

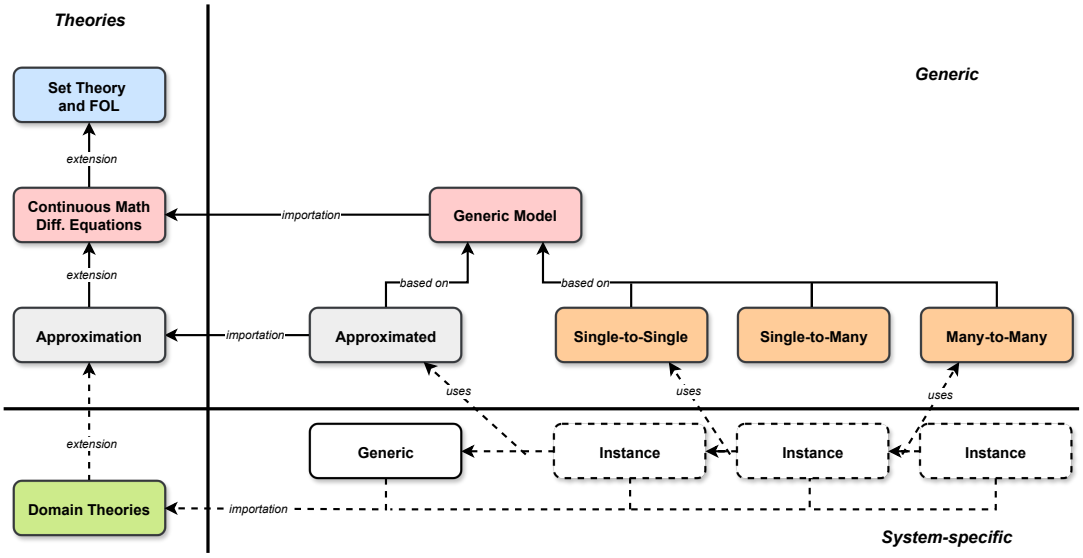

Figure 3.3: A Generic Formal Framework for Designing Hybrid Systems

specific requirements and properties. Note that each component of the framework can be extended by adding new components to its core. For example, other generic models for modelling different hybrid system structures can be defined. New patterns can also be defined as a generic refinement of an existing model, possibly supported by new theories. A detailed description about this framework, including core components, theories, domain theories and generic patterns, is provided in [20][Dupont 2021]. Below, we describe essential components to support this framework.

Hybrid Modelling Features

Hybrid systems deal with time and interleaving nature of discrete and continuous behaviour. As we know, the Event-B modelling language supports only discrete modelling thus we extend it to handle the required hybrid features by using Event-B theories plugins [Butler 2013]. In this section, we describe essential ingredients required for modelling hybrid systems.

Time. Time can evolve in a discrete ("step-wise") or continuous manner in a system. In the case of a hybrid system, we need a time to define continuous behaviour, which is represented as dense time $t \in \mathbb{R}^+$ to model continuous evolution (without any "jump").

System state. There are two different kinds of states variables: *continuous state* and *discrete state*. Discrete state is an Event-B native state variable that can be updated using the Event-B before-after predicate (BAP). On the other hand, we define a new operator, continuous before-after predicate (CBAP), for updating continuous state variables. The CBAP operator is defined below, along with continuous assignment and continuous evolution operators.

Operator 1 (Continuous Before-After Predicate) *Let $t, t' \in \mathbb{R}^+$ two time points with $t' > t$. Let $x_p \in \mathbb{R}^+ \to S$ a continuous state variable. Finally, let $\mathcal{P} \subseteq (\mathbb{R}^+ \nrightarrow S) \times (\mathbb{R}^+ \nrightarrow S)$ a predicate on the before and after values of the state variable and $H \subseteq S$ an evolution domain constraining the evolution of x_p. The continuous before-after predicate modelling the change of x_p on time interval $[t, t']$ following predicate $\mathcal{P}$ and constrained by evolution domain H, denoted $x_p :|_{t \to t'} \mathcal{P}$ & H, is defined as so:*

$$
\begin{aligned}
x_p :|_{t \to t'} \mathcal{P}(x_p, x_p') \;\&\; H \equiv\; & [0, t[\lhd x_p' = [0, t[\lhd x_p && (PP) \\
& \land \mathcal{P}([0, t] \lhd x_p, [t, t'] \lhd x_p') && (PR) \\
& \land \forall t^* \in [t, t'], x_p(t^*) \in H && (LI)
\end{aligned}
$$

Using the domain restriction operator $\lhd$, the continuous before-after predicate operator ensures that the past of the continuous variable remains unchanged (past preservation, *PP*) while its future is modified to correspond to the given predicate $\mathcal{P}$ (predicate, *PR*). This assignment also ensures time is progressing along with the physical plant, going from time t to $t' > t$, and that on the resulting interval $[t, t']$ nothing else happens with regard to given invariant H (local invariant, *LI*) [Dupont 2021].

Operator 2 (Continuous Assignment) *Let $t, t' \in \mathbb{R}^+$ two time points, with $t' > t$. Let a continuous state variable $x_p \in \mathbb{R}^+ \to S$, a function $f \in \mathbb{R}^+ \to S$ and an evolution domain H. The continuous assignment of f to x_p with evolution domain H, denoted $x_p :=_{t \to t'} f \,\&\, H$, is defined as so:*

$$x_p :=_{t \to t'} f \,\&\, H \;\equiv\; x_p :|_{t \to t'} x_p' = f \,\&\, H \tag{3.1}$$

It "appends" a piece of an (explicit) function to the continuous state variable.

Operator 3 (Continuous Evolution) *Let $t, t' \in \mathbb{R}^+$ two time points, with $t' > t$. Let a continuous state variable $x_p \in \mathbb{R}^+ \to S$, a differential equation $\mathcal{E}$ and an evolution domain H. The continuous evolution of x_p along differential equation $\mathcal{E}$ with evolution domain H, denoted $x_p :\sim_{t \to t'} \mathcal{E} \,\&\, H$, is defined as so:*

$$x_p :\sim_{t \to t'} \mathcal{E} \,\&\, H \;\equiv\; x_p :=_{t \to t'} \eta \,\&\, H \tag{3.2}$$

*where η is **any solution of $\mathcal{E}$ on** $[t, t']$ that does not violate evolution domain H.*

A Theory of Approximation. A theory of approximation is required in order to use approximation refinement in Event-B. The approximation operator is presented below, along with the definitions of the expansion and shrinking operators.

Operator 4 (Approximation) *Let $\delta \in \mathbb{R}^+$ (i.e. $\delta \in \mathbb{R}$ and $\delta \geq 0$) and $x, y \in E$. x is **approximately equal** to y by δ (or x is a δ-approximation of y), denoted $x \approx^{\delta} y$ if:*

$$x \overset{\delta}{\approx} y \;\equiv\; d(x, y) \leq \delta$$

Operator 5 (δ-Expansion) *Let $\delta \in \mathbb{R}^+$ and $S \subseteq E$. The δ-expansion of S is denoted $\mathcal{E}_\delta(S)$ and is defined as,*

$$\mathcal{E}_\delta(S) = \{y \in E \mid \exists x \in S, x \overset{\delta}{\approx} y\} = \{y \in E \mid \exists x \in S, d(x, y) \leq \delta\}$$

Operator 6 (δ-Shrinking) *Let $\delta \in \mathbb{R}^+$ and $S \subseteq E$. The δ-shrinking of S is denoted $\mathcal{S}_\delta(S)$ and is defined as,*

$$\mathcal{S}_\delta(S) = \{x \in S \mid \inf_{y \in E \setminus S} d(x, y) > \delta\} = \{x \in S \mid \forall y \in E \setminus S, d(x, y) > \delta\}$$

Theory Implementation

All the required theories related to continuous functions, differential equations and approximation, including properties, are implemented using Event-B theory plugins [Butler 2013]. The complete formal development is available at `https://irit.fr/~Guillaume.Dupont/models.php`.

All the developed theories are categorised as: *general algebra*, *reals*, *functions*, *differential equations*, *approximations* and *domain theories*. These theories are built in a modular fashion, allowing any theory to be extended by adding new operators and their properties without affecting the core development of any other theories. Listing 3.1 presents an excerpt of the developed theory for differential functions. The main operators are:

- $\mathbf{DE}(S)$ type for differential equations which solutions are valued in set S;
- $\mathbf{ode}(f, \eta_0, t_0)$ represents the *Ordinary Differential Equation* (ODE) $\dot{\eta}(t) = f(\eta(t), t)$ with initial condition $\eta(t_0) = \eta_0$ (note that we could define other constructors for other types of dynamics);

- $\mathbf{solutionOf}(D, \eta, \mathcal{E})$ is the predicate stating that function η is a solution of equation $\mathcal{E}$ on subset D;
- $\mathbf{Solvable}(D, \mathcal{E}, H)$ is the predicate stating that equation $\mathcal{E}$ has a solution defined on subset D so that the solution satisfies the constraint H;
- An encoding of the *Cauchy-Lipschitz theorem*, that allows to demonstrate the solvability of a given equation under some specific conditions;

The development of theories consists of more than 150 operators and 350 properties.

THEORY DiffEq
TYPE PARAMETERS E, F
DATATYPES
DF (F)
CONSTRUCTORS
ode$(fun : \mathbb{P}(\mathbb{R} \times F \times F), initial : F, initialArg : \mathbb{R})$
OPERATORS
solutionOf *predicate* $(D : \mathbb{P}(\mathbb{R}), \eta : \mathbb{R}^+ \nrightarrow F, \mathcal{E} : \mathbf{DE}(F))\ldots$
Solvable *predicate* $(D : \mathbb{P}(\mathbb{R}), \mathcal{E} : \mathbf{DE}(F))$
direct definition
$\exists x \cdot x \in (\mathbb{R}^+ \nrightarrow F) \land D \subseteq dom(x) \land \mathbf{solutionOf}(D, x, \mathcal{E})$
AXIOMS
CauchyLipschitz:
$\forall \mathcal{E}, D, D_F \cdot \mathcal{E} \in \mathbf{DE}(F) \land \ldots \Rightarrow \mathbf{Solvable}(D, \mathcal{E})$

Code Snippet 3.1: Differential Equation Theory Snippet

In similar fashion, approximation theories are developed, including the approximation, shrinking and expanding operators. Furthermore, the required domain knowledge for particular hybrid systems can be encoded by developing domain theories that can extend the theory of differential equations (or approximation). Such domain theories may contain specific constants, control functions, domain specific properties, as well as the required axioms and theorems that may aid in the formal development and proof the hybrid systems.

A Generic Model for Hybrid Systems

The controller-plant loop architecture pattern, which includes the controller, actuator, plant, and sensor, governs the majority of hybrid system behaviour. In this architecture, controller controls the plant characterised by continuous behaviour or differential equations. A sensor monitors and measures physical aspects of an environment and plant and sends an electrical signal to the controller, whereas an actuator uses the controller's electrical signal to perform a physical action. All of these components are linked together to form a closed loop model. We are interested to design a correct controller for hybrid systems as well as a correct integration of various components (discrete and continuous). We derive a generic model of hybrid systems based on this controller-plant loop architecture pattern to represent the core components of a controller. Furthermore, this generic model can be instantiated and refined for the development of complex controllers as well as the adaptation of various architectural patterns.

In this generic model, we define three state variables to represent time t, discrete states x_s, and continuous states x_p using typing invariants ($inv1$-$inv3$). An extra invariant is defined in $inv4$ to ensure that the continuous states are defined from the origin of time to the current time. These defined variables are initialised by the INITIALISATION event. In this event time t is set to 0 and the discrete and continuous state variables are not-deterministically assigned. Note that these state variables can be further refined in order to define more precise discrete and continuous behaviour.

The generic model includes four core abstract events: *Transition*, *Actuate*, *Sense*, and *Behave*. The *Transition* event allows for changes in the current discrete states to be caused by controller decisions or any input from the plant, environment, or user. The guard of this event states that input states are defined ($grd1$) and the action allows to updates the discrete states ($act1$) non-deterministically. The *Actuate* event is a type of continuous event that allows only continuous states to be updated using the continuous before after predicate operator. This event's action ($act1$)

MACHINE $Generic$
VARIABLES t, x_s, x_p
INVARIANTS
inv1: $t \in \mathbb{R}^+$
inv2: $x_s \in \text{STATES}$
inv3: $x_p \in \mathbb{R}^+ \nrightarrow S$
inv4: $[0, t] \subseteq \text{dom}(x_p)$
EVENTS
INITIALISATION
THEN
$act1 : t := 0$
$act2 : x_s :\in \text{STATES}$
$act3 : x_p :\in \{0\} \rightarrow S$
END

EVENT Transition
ANY s
WHERE
$grd1 : s \in \mathbb{P}1(\text{STATES})$
THEN
$act1 : x_s :\in s$
END

EVENT Actuate
ANY eq, s, H, t'
WHERE
$grd1 : t' > t$
$grd2 : eq \in \mathbf{DE}(S)$
$grd3 : \mathbf{Solvable}([t, t'], x_p, eq, H)$
$grd4 : s \subseteq \text{STATES}$
$grd5 : x_s \in s$
$grd6 : H \subseteq S$
$grd7 : x_p(t) \in H$
THEN
$act1 : x_p :|_{t \rightarrow t'} eq \,\&\, H$
END

Code Snippet 3.2: Generic model events

updates continuous states (x_p) that comply with a differential equation (plant dynamics) under the solvability condition ($grd2$ - $grd3$) and a constraint H on the plant evolution domain ($grd6$ - $grd7$). Furthermore, there are more guards to satisfy mode automaton ($grd4$ - $grd5$) and the time is advanced ($grd1$).

EVENT Sense
ANY s
WHERE
$grd1 : s \in \mathbb{P}1(\text{STATES})$
$grd2 : p \in \mathbb{P}(\text{STATES} \times \mathbb{R} \times S)$
$grd3 : (x_s \mapsto t \mapsto x_p(t)) \in p$
THEN
$act1 : x_s :\in s$
END END

EVENT Behave
ANY s
WHERE
$grd1 : t' > t$
$grd2 : eq \in \mathbf{DE}(S)$
$grd3 : \mathbf{Solvable}([t, t'], x_p, \mathcal{P}, \top)$
THEN
$act1 : x_p :|_{t \rightarrow t'} eq \,\&\, \top$
END

Code Snippet 3.3: Generic model events

The *Sense* event is a type of discrete event used to model the hybrid system's sensing behaviour. The event's action enables non-deterministic updating of discrete states ($act1$) by observing both discrete and continuous states with respect to time ($grd1$ - $grd3$). At this abstract level, sensing is considered instantaneous, but a delay can be introduced through refinement. The last event *Behave* is also a continuous event that updates continuous states similar to the *Actuate* event. This event allows the plant to express any behaviour (outside of normal control) caused by environmental factors such as wind, heat, pressure, and so on. The event's action ($act1$) updates continuous states (x_p) that comply with a differential equation (plant dynamics) under the solvability condition ($grd2$ - $grd3$) without any constraint $\top$ (means anything can be happened) on the plant evolution domain, and time is advanced ($grd1$).

All of the above generic model events are abstractly defined in order to capture the core functionalities of hybrid systems. The generic model and events can be refined to develop any complex hybrid system using any architecture design pattern. We have used this generic model for implementing different architecture patterns [20] and approximation patterns [41].

Hybrid Model Simulation

Our goal is to provide a simulation environment for the generic formal framework. So far, there is no mechanism for simulating or animating an Event-B hybrid model. Note that the ProB [Leuschel 2003] tool only supports model checking and animating the Event-B discrete model. We propose a simulation framework for animating and simulating Event-B hybrid systems in order to validate our hybrid models and ensure that controller and plant behaviour are correct. The proposed framework enables the transformation of hybrid Event-B models into Simulink/Stateflow models [MathWorks 2021]. The translation mechanism is straightforward and simple, with the discrete part of the

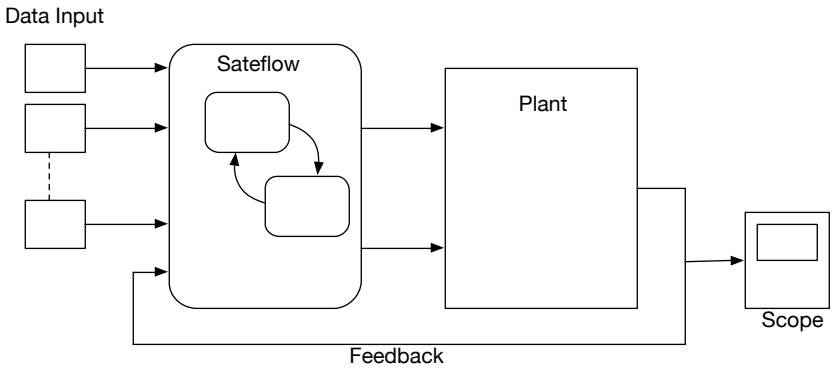

Figure 3.4: Simulink model

hybrid model developed in Stateflow and the continuous part modelled in Matlab user defined function. Our prime focus regarding the approach is to preserve the separation between the discrete and the continuous aspects.

Fig. 3.4 shows a simple simulink model with required components derived from Event-B hybrid models. The Stateflow model is made up of several states (modes) that are linked together by transitions that correspond to Event-B guards. The goal of these modes is to transition between different states. The switch is determined by the plant states detected by sensors. The discrete variables in each state can be updated using *entry*, *exist* and *during* actions, which can be identified from the Event-B model. An initial state of the stateflow model can be determined corresponding to the initial state of the Event-B model. Plant dynamics is a differential equation associated with each state describing the physical evolution of the continuous variables. In the translated model, the continuous behaviour of a plant is represented by a Simulink block: Matlab user defined function. In this block, we encode dynamic behaviour of the plant described in the Event-B model in Matlab. Finally, the output of this simulink block is connected to the Stateflow model as an input to the model, as well as a sink block, Scope, to display the simulation result. The proposed translation schema with examples is described in [36].

In the Simulink environment, we can simulate the translated model. In the simulation, we can simulate the entire behaviour of the hybrid system by using a range of input values. Moreover, the hybrid system simulation assists in validating the given invariants and determining the possible range of system input and output that satisfy the possible safety properties. The use of simulation plays a key role in the development of hybrid systems. By analysing simulation results, we can validate the discrete and dynamic behaviour of the hybrid Event-B model. This simulation emulates the required behaviour based on the results of the formal models, allowing them to be used effectively to evaluate the developing hybrid system. Moreover, the simulation results may aid in identifying potential flaws in the developed model. If an error will be discovered during simulation, we can modify the hybrid Event-B model. This process can be applied iteratively to obtain a correct hybrid model satisfying continuous and discrete behaviour. It is important to note that the generation of Simulink models is cost effective and it leads to a system implementation that can be used to deploy a real system.

Moreover, we develop an approach to use the completeness and disjointness properties of well-formed tabular expressions to aid us in establishing those properties in Stateflow models in [71]. From the Stateflow models, we generate a new kind of tabular expression that includes extended output options. We use the informal Stateflow semantics from MathWorks documentation as the basis for generating our tabular expressions. The generated tabular expressions are then used to guarantee completeness and disjointness. We provide a transformation algorithm that we are implementing in a tool to automatically generate tabular expressions from Stateflow models.

A Formal Approach to Rigorous Development of Critical Systems

In this section, we propose a framework based on formal methods for developing safety-critical systems. This framework is depicted in Fig. 3.5, which is adopted from our previous work [1][32][83][98]. In this new framework, we

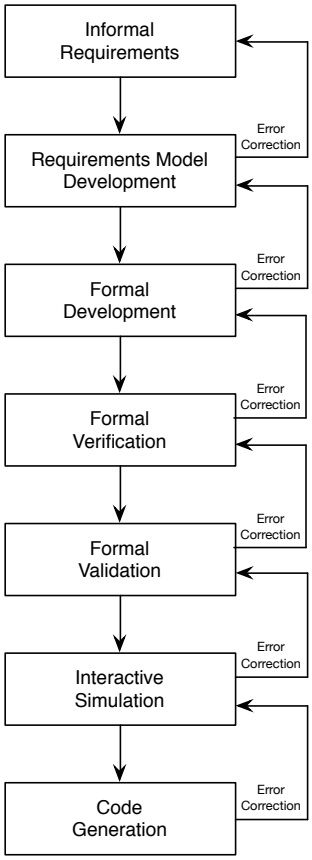

Figure 3.5: A Formal Framework for Developing Critical Systems

introduce a new step for documenting system requirements using tabular expressions that are used to automate the task of formal modelling. Note that the presented framework in [1][32][83] does not support the requirement documentation and refinement automation to produce a formal model in Event-B automatically. Further, verification, validation and interactive simulation processes can be used to check the generated formal model. Finally, source code can be produced from the corrected and proved formal specifications. All these steps of the system development are shown in Fig. 3.5. This development framework assists in automation of the development process. In addition, to automate the development process of the proposed framework, we also develop a set of required tools to support various steps from requirement analysis to implementation. A detailed description of these steps is described in the following sections.

Informal System Requirements

Software requirements specification is widely used in a restricted form of natural language that is produced by most of the industries today. A natural language representation consists of functional and non-functional requirements that is also convenient for different stakeholders, such as technical and non-technical, to comprehend the system requirements. This is the first step in our framework for developing system requirements in natural language following basic guidelines given in [Sommerville 2004] to describe essential ingredients, for example, functional requirements, non-functional requirements, system evolution, exceptional events requirements, validation criteria, assumptions, traceability, and system data, needed for building a system without considering design and implementation.

Requirements Model Development

In our proposed framework, to deal with inconsistent and incomplete requirements, we use tabular expressions to specify the system requirements from natural language informal requirements. A tabular expression presents a relation between input and output, and control variables for describing the system requirements unambiguously. The presented layout of these tables is clearly readable but yet still formal, which can help software engineers and other stakeholders to check the *completeness* and *disjointness* of the given requirements. In our work, we use horizontal condition table (HCT) to generate requirements models. For documenting the system requirements and to check the correctness of produced tables together with satisfying the required properties of *disjointness* and *completeness*, we use the TET tool [Eles 2011]. This tool is integrated with a tabular expression editor and requirements checking tools, such as a SMT solver, ATP and PVS.

Formal Development

In our framework, we propose a new refinement-based method to constructing the Event-B model for formal verification from the requirements model defined in tabular expressions. In order to produce formal models from tabular expressions, we can use refinement that allows us to construct a model progressively by traversing tabular requirements using a *correct-by-construction* approach [61][9]. The proposed strategy is suitable for any formal language that can support refinement based development. It should be noted that in this work, we use the Event-B modelling language, which supports refinement based progressive development. A detailed description of the refinement automation, including tools support, is provided in Section 3.1.

Formal Verification

Formal verification is an important step for proving and disproving the correctness of intended behaviour of a formalised system under the given safety properties and assumptions. To check the consistency and correctness of the generated formal model, we can use an automated theorem prover in the process of system verification. The formal verification ensures that the model is designed correctly, the developed formal specification has required properties and the generated models do not contain errors, oversights, or bugs. In our framework, this formal verification step allows introducing a list of safety properties in the generated models. The safety properties can be introduced either in each refinement level or in the last concrete model. Introduction of safety properties is essential to check the correctness of system behaviour. We can use the Rodin development framework [Abrial 2010b] for project management and model verification through syntactical checking and refinement checking. In addition, this tool can be used for proving the generated proof obligations. In case of finding any inconsistency in the model during the system verification process, we can modify the model obtained from the previous step of the development process. We can do it iteratively to get a correct model.

Formal Validation

Formal validation is a process to determine the formalised model is an accurate representation of the real world from the perspective of the intended user. It uses a small data set for validating and analysing the formalised requirements. The validation process ensures that the generated and formally verified models fulfil the intended requirements by the methods employed and the results obtained. Model checkers allow for verifying systems requirements clearly and to make the formal techniques easier to use while offering a high degree of automation. In our framework, we emphasise using the tools of model checking to validate the verified and developed formal models. In fact, we use the ProB [Leuschel 2003] model checker in our framework to validate the generated formal models. The selected tool ProB allows for checking the Event-B models. If this tool finds any counterexample or a model does not satisfy the desired behaviour, then the model obtained from the previous step can be modified. This iterative process can be applied by satisfying the required safety properties until it finds a correct formal model (without any error or counterexample).

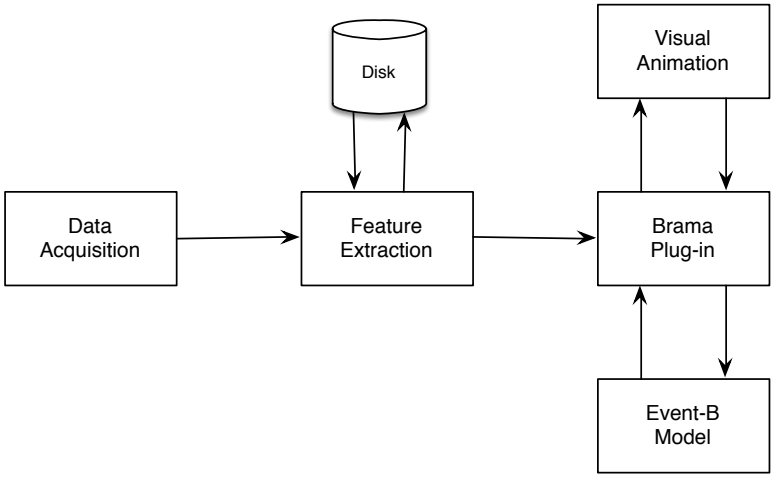

Figure 3.6: Architecture for Interactive Simulation

Interactive Simulation

The use of formal methods for rigorous reasoning and validation using a model checker is not enough for designing safety-critical systems, when stakeholders do not have an adequate understanding of mathematical reasoning, such as typical domain experts in the medical domain. In our development framework, we use an interactive simulation [93] for retrieving data from the system environment and use this retrieved data for performing the model animation of the developed formal model. This model animation can be used by domain experts to understand the desired functional behaviour without understanding the developed formal models of the given system. The simulation framework is depicted in Fig. 3.6, where *Data Acquisition* is used to measure the physical phenomena or physical property in form of input data, *Features Extraction* is used to extract the required features from input data, *Disk* can used to store the extracted features, *Brama Plug-in* is used to interface between the Event-B model and a visual animator. *Visual Animation* is a dynamic graphical animation to show system behaviour or system activities, and the *Event-B model* is a model of the formalised system that can be used to animate with the help of an interface plug-in. A detailed description of this architecture is available in [93]. Note that the same architecture can use Functional Mock-up Interface (FMI) [Blochwitz 2011, Savicks 2014] in place of Brama Plug-in for developing an interactive simulation.

This interactive simulation performs the required behaviour as per the results of the formal models so that they can be used effectively to evaluate a system and to correct the sources of faulty behaviours. Such technique can be useful when domain experts are involved in the system development. In our framework, we use the interactive simulator for checking the correctness of system behaviour according to domain experts. If any error is discovered by the domain experts, then the feedback approach allows us to modify the previous development steps. In this phase of the development, most of the errors can be discovered by the domain experts. This interactive simulation can be applied iteratively until the final correct simulation model is accepted by the domain experts.

Code Generation

This is the last stage of our proposed framework. It produces a target language source code for system implementation from the formalised, proved and validated formal models. This development phase involves the major executable components, definitions of concrete data structure, and some auxiliary structures or functions that may be assumed in a design. In our proposed framework, we use our developed tool EB2ALL[1] [88] to generate source code in a desired programming language from the verified formal specification. The code generation tool EB2ALL is a collection of

[1] http://singh.perso.enseeiht.fr/eb2all/

plug-ins that allows producing code in C, C++, Java, C# and Solidity in Rodin IDE [Abrial 2010b]. In order to produce a correct code, we need to apply pre-processing to refine the developed system to make it more concrete and deterministic by removing the abstract variables and by providing the language-dependent data range for each variable. The new refined model must be proved before applying the code generation process. A set of contexts and concrete machine can be identified from the selected project, that can be further used for code generation through parsing, and syntactical and semantic analysis of the Event-B model. The code generation process successfully translates the Event-B concrete model by supporting the defined constants, enumerated sets, functions, variables, arrays, parameters, events, guards and actions with arithmetic and logical operations. An Event-B model is a collection of events, where each event is converted into an equivalent function. A generated file contains appropriately generated constants, local and global variables, arrays, functions, and events which are generated from an Event-B model using lexical and syntactic analysis. In addition, this tool provides flexibility to choose to generate optimised or sequential code.

Supporting Tools

We provide a tool-chain to support the rigorous development framework. It should be noted that we have used existing tools and developed some new tools to support this framework. For example, with respect to existing tools, we use Rodin [Abrial 2010b] and ProB [Leuschel 2003] tools support for model development, including verification and validation, and with respect to our developed tools, we use the TET [Eles 2011] tool for documenting system requirements in tabular expressions, TX2EB [Jiang 2015] for generating Event-B models from the documented system requirements in tabular expression form, and EB2ALL [88] for code generation in various programming languages. In addition, the prime motivation for our work is to pave the way for automated tool support for the proposed framework.

F3FLUID: A Formal Framework for Developing Safety-Critical Interactive Systems in FLUID

Due to the complex nature of interactive systems, particularly safety-critical interactive systems, their engineering has resulted in the development of several notations, techniques and methods for the design of high-quality interactive systems that have been borrowed from various disciplines like psychology, cognitive science, ergonomy, and computer science. These methods and techniques are designed to address different stages of the development cycle. They are intrinsically heterogeneous, resulting in heterogeneous models due to different semantics and abstraction levels. To address the disadvantages of heterogeneity, we propose developing a modelling language that incorporates specific characteristics related to the design, verification, and validation of safety-critical interactive systems at a higher abstraction level. This work is carried out in the ANR funded project, FORMEDICIS. The goal of this project is to propose a suite for developing and designing safety-critical interactive systems. This suite consists of the integration of a unified formal framework, F3FLUID [22][42, 47], and associated tools. This framework is based on the FLUID (Formal Language of User Interface Design) core pivot modelling language, which is a domain-specific language for describing safety-critical interactive systems at a higher abstraction level. It enables the description of specific interactive domain properties by associating domain concepts with state variables and events, user scenarios recording so-called "story boards," and logic-based behavioural properties via annotation. The framework's associated formal modelling techniques and tools aid in the design of models at various abstraction levels, such as the analysis of interaction properties, domain properties, and graphical simulation of models.

The F3FLUID modelling framework supporting the *correct by construction* approach is depicted in Fig. 3.7. Formal verification and validation techniques (right part of Fig. 3.7) are linked to a FLUID model (left part of Fig. 3.7) in this framework. The results of the analyses provide feedback on the original FLUID model, which is depicted by dashed arrows. Finally, the models associated with these verification and validation techniques can be used to develop an application.

We chose Event-B [Abrial 2010a], ProB [Leuschel 2003], and Interactive Cooperative Objects [Navarre 2001a, Hamon 2013] that can support the F3FLUID for handling modelling, refinement, verification, animation, simulation, and implementation. Note that the F3FLUID is not dependent on the selected tools, but it can be used with other techniques and tools such as Smala/Djnn [Magnaudet 2018] and Electrum [Brunel 2018].

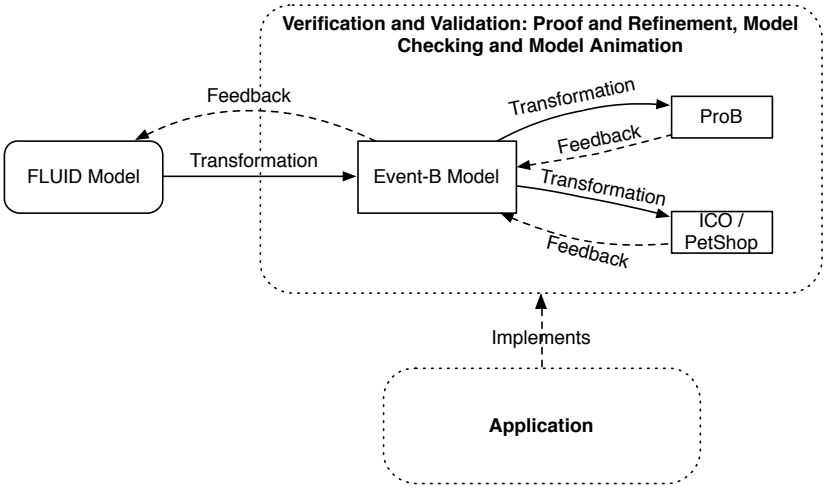

Figure 3.7: F3FLUID framework

A FLUID model is translated into an Event-B model, as shown in Fig. 3.7. To ensure the correctness of translated models as well as the defined properties, all generated proof obligations must be satisfied. On this Event-B model, two additional verification techniques are used. First, the ProB model checker, which is linked to Event-B, is used to validate other properties, specifically those expressed using temporal logic. ProB is also used for validation due to its ability to perform Event-B model animation and use this animation to check the scenarios expressed in FLUID. Second, the Event-B models are transformed into Interactive Cooperative Object Petri net models to check additional properties, as well as animating the modelled interactive system where interactive objects are linked to Petri nets [Murata 1989, Barboni 2003].

Stepwise design of the FLUID pivot model

When dealing with complex interactive systems, developing a FLUID model may necessitate multiple iterations. Starting with a basic model, it is gradually enhanced by adding design decisions and handling requirements. It is important to ensure that the enriched model retains the behaviours and properties expressed in the original model. We use the Event-B refinement mechanism to meet the requirement of preserving behaviour and properties. As a result, the framework of Fig. 3.7 has been extended to handle the stepwise FLUID model design process. The resulting framework is shown in Fig. 3.8.

In the framework of Fig. 3.8, we define a series of layers issued from Fig. 3.7. Event-B refinement is used to check FLUID model extensions as follows.

According to Fig. 3.8,

- a FLUID model *FLUID Model i* is transformed into *Event-B Model i* for which analyses are performed;
- a FLUID model *FLUID Model j* is defined by the designer as a possible enrichment or extension of the *FLUID Model i* and another *Event-B Model j* is obtained after transformation;
- the designer shall check that *Event-B Model j* refines *Event-B Model i*. If this refinement is proved, then, we can guarantee that the behaviours and properties are preserved, else, the feedback returned by the analysis of *Event-B Model j* is used to update the *FLUID Model j*.

The above process is repeated until the designed model handles the expression, verification and validation of the system requirements.

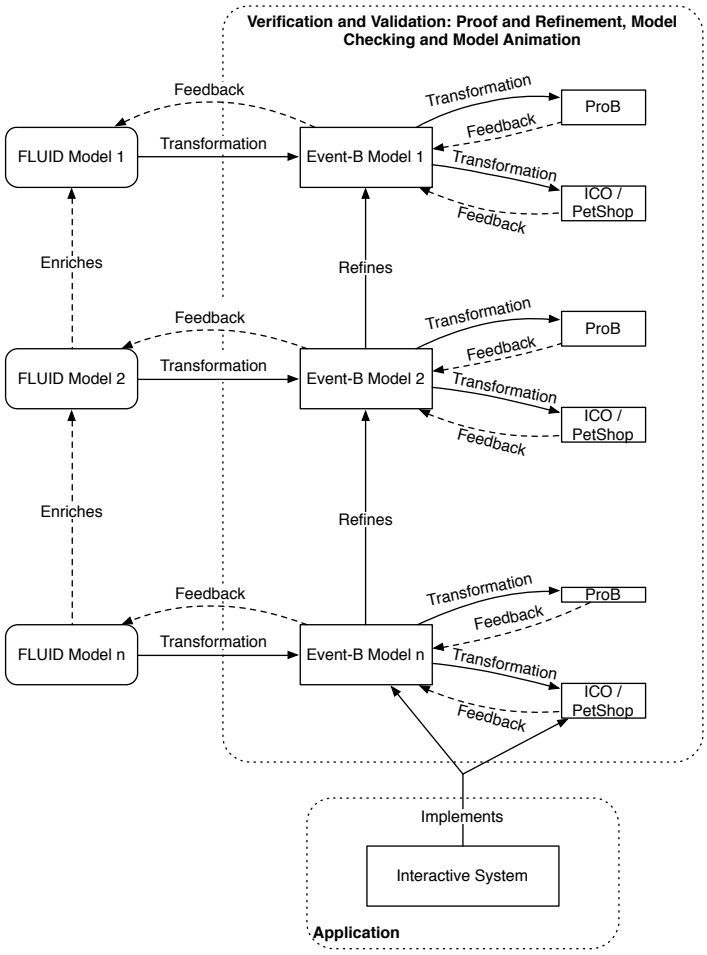

Figure 3.8: F3FLUID framework

Note that the progressive safe FLUID enrichment allows to add the required safety properties and low level system requirements related to interactive system. These refinements preserve the relations between an abstract model and its corresponding concrete model, while introducing low level details and new properties to specify more concrete behaviour of the system. Thanks to the correctness of the Event-B refinements, this incremental development guarantees the correctness of concrete behaviour of an interactive system with respect to the abstract model. The abstract and refined Event-B models are checked with the Rodin tools. The analysed models provide feedback to the original FLUID model shown in Fig. 3.8 as dashed arrows.

The generated Event-B model employs the Rodin tool to check the syntactical correctness and consistency of the modelled system under the given safety properties. The generated proof obligations (POs) must be discharged to establish the correctness of the Event-B model. These generated POs can be discharged automatically or interactively through simplification. The ProB [Leuschel 2003] tool is used for model validation and model animation. ProB supports *automatic consistency checks*, *constraint-based checks* and can also detect potential deadlocks. Note that the Event-B model produced is directly used in ProB. We use ProB to show the absence of errors (there is no counterexample) and the absence of deadlock. Furthermore, we can describe the properties of the FLUID model defined in LTL formulas to check the correctness of interaction properties of the produced model. Finally, the proven Event-B model can be used to derive the ICO model. We extract the Petri nets model of the Event-B model, which is crucial in the development of the ICO model. It should be noted that other ICO model components, such as the presentation part,

T_{decl}	:	Datatype and Constant declaration translation function
T_{st}	:	State variables translation function
T_{evt}	:	Events translation function
T_{grd}	:	Event's guards translation function
T_{act}	:	Event's actions translation function
T_{axm}	:	Axioms translation function
T_{exp}	:	Expressions translation function
T_{prop}	:	Properties translation function

Table 3.3: Transformation function

INTERACTION *Component_Name*
  DECLARATION
    T_{decl}(*SETS s*) //Sets
    T_{decl}(*CONSTANT c*) //Constant
  STATE
    $T_{st}(v)$ //Variable without $@tag$
    $T_{st}(v@tag)$ //Variables with domain-specific $@tag$
    ...
  EVENTS
    INIT
    ...
    T_{evt}(//Events
    Event $evt@tag[x]$
      where
        $T_{grd}(G(s, c, v, x, v@tag, x@tag))$ //Guard
      then
        $T_{act}((v, v@tag) :\mid (BA(s, c, v, x, v', v@tag, x@tag, v'@tag))$ //Action
      end
)
  ASSUMPTIONS
    $T_{axm}(A(s, c))$ //Axioms
  REQUIREMENTS
    PROPERTIES
      $T_{prop}(Prop(s, c, v, v@tag))$ //Safety properties
END *Component_Name*

Table 3.4: FLUID Model transformation

rendering, and activation functions, can be provided for animating the ICO model.

FLUID to Event-B Transformation

The core concepts of the transformation process from FLUID to Event-B are straightforward. Table 3.3 shows a set of transformation functions corresponding to FLUID clauses (see Table 3.4). These transformation functions are used to produce the Event-B context and machine. In the Event-B generated model, the context and machine models are produced corresponding to the static and dynamic properties of the FLUID model. Note that the supported symbols, datatypes and expressions of the transformation process is straight forward because the FLUID language operators and syntax are close to the Event-B modelling language. The FLUID language is underpinned on the basic formal syntax of Event-B modelling language extending modelling features to capture domain specific requirements and properties for critical interactive systems.

This translation process has been implemented, Fluid2EB[2], as an Eclipse plugin using the Xtext language modelling facilities and the Xtend model manipulation language [Bettini 2013]. It relies on the FLUID and Event-B language implementations done with Xtext that share parts of their expression and action sub-languages. These common parts allow to ease the implementation of the translation process and focus on the FLUID specific elements and especially the management of the domain-specific metadata (the tags).

[2]See the development forge at `https://sourcesup.renater.fr/projects/anr-formedicis`

Event-B to ICO Transformation

As both modelling techniques are state based, the Event-B notation is particularly suitable for translation into an ICO notation. There are several similarities between the two modelling languages, in fact they are based on the same kind of semantics: transition systems. Ultimately, every finite Event-B model can be represented unambiguously by a reachability graph that provides all possible states that can be reached from the original model initial state using the allowed transitions. Similarly, the underlying reachability graph of the ICO model represents the state space consisting of all the marking configurations and the transitions between states resulting from the firing of the transitions in the original Petri nets. We develop an algorithm that allows an ICO model to be derived from an Event-B model. The algorithm must produce a readable and editable Petri nets since the final result will be analysed and even modified by engineers using the Petshop environment [Barboni 2003].

The translation process is divided into three steps. To begin, identify a set of Event-B variables and create clusters based on their appearance to create a Petri subnets. Second, determine the composition relationships among the constructed Petri subnets. The guard-body relationship is a precedence relationship derived from the appearance of variables in the model events' guards and bodies. For example, if a subnet has some variables included in the guard of some events, the event will proceed to all subnets whose variables are updated by the same event. Finally, we need to set the initial marking for the entire Petri nets. The initialisation event is appropriate for this purpose. In fact, this event will determine where a token will be placed and will also provide the initial state of the Petri net. The process allows to derive an object Petri nets from an Event-B model. However, in order to fully produce an ICO model, the other components, namely the presentation part, the activation function, and the rendering function, must be developed manually. Petshop [Barboni 2003] can be used to analyse the generated ICO models.

A detailed description about this framework, including transformation strategy for producing Event-B models, ICO models, and the required tool support, is provided in [22][42, 47].

MVC Pattern for Structuring Event-B Models of WIMP Interactive Applications

Another important contribution is to propose the structuring of Event-B models of interactive systems using *model-view-controller* (MVC) [Krasner 1988] and to demonstrate the benefits from doing so [24][58]. Here, we describe a formal approach for structuring and developing an interactive application using a *correct by construction* process. This development approach is derived from the MVC architecture [Krasner 1988]. In fact, the core architecture of this developing approach is similar to MVC but each component of the MVC is defined progressively using refinement calculus [Back 1998, Abrial 2010a]. This development approach is depicted in Fig. 3.9. In this proposed architecture, we show the classical scheme of MVC with possible interaction protocol and refinements for each MVC component. In Fig. 3.9, each triangle represents possible refinements corresponding to the MVC components. Note that these triangles are overlapped with other triangles due to some shared variables and functional behaviours between the MVC components. Such refinement strategy allows us to analyse and reasoning a complex behaviour of an interactive application under the given constraints. Initially, an interactive application can be defined abstractly and then it can be refined by introducing more concrete behaviour using new state variables, events and properties.

In this development approach, first, we formalise the *model* component, which describes a very high level of abstraction of an interactive application in form of system modality. The system modality can be introduced in several refinement layers. Each refinement step introduces system level modality related to subsystem for analysing the required safety properties and for guaranteeing the correctness of modes transitions of an interactive application. The next step of the development is to introduce the *controller* component and the required controller behaviour. Similar to the previous development, the controller components can be introduced progressively for each subsystem. In this development, we define static and dynamic properties. The static properties related to the controller can be defined by extending the context of the model, while the controller components and dynamic properties can be defined by introducing a set of new events and by refining the abstract events. In each refined model, the required safety properties can be introduced to meet the desired behaviour of the interactive application corresponding to the defined controller. After introducing the model and controller components in the developing interactive system, we introduce the view component. In this refinement, we introduce all visual and graphical elements, such as buttons, radio buttons,

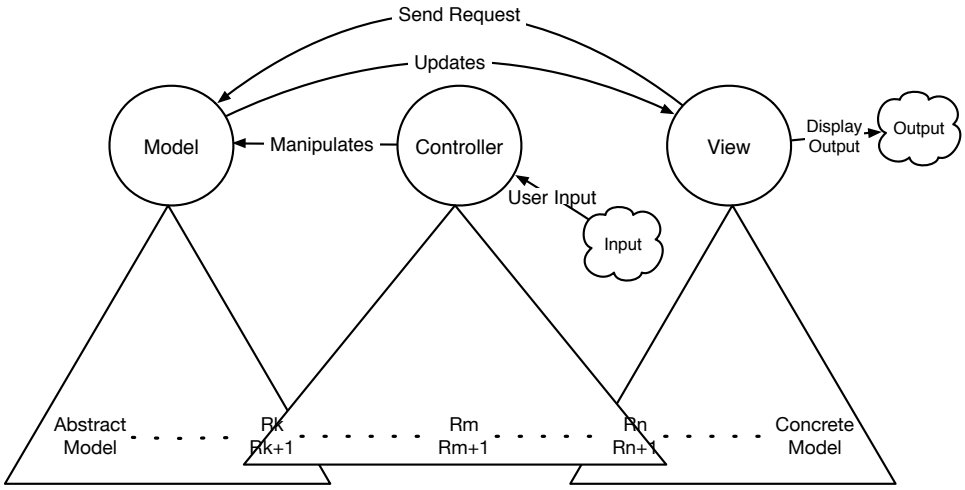

Figure 3.9: MVC structuring for Event-B models

labels, of an interactive system. These graphical elements can also be introduced in several refinements as similar to the previous developments. Note that these newly introduced elements must be linked to the mode and controller components in order to specify the interactive system. By adding the view components, we can prove the correctness of the request functions and responses of the controller. A set of required safety properties can be introduced in the progressive development to check the functional correctness of each visual elements. When all the elements are designed and integrated, the interaction properties of each component are introduced to check the correctness of the system interaction behaviour. Note that the formal development of the view is complex, and we need to add several guards in different events to meet the desired properties of interaction behaviour for each view component of the interactive systems.

We show that using architecture patterns such as MVC is of great help to define a refinement chain to support the modelling activities involved in the design of an interactive system. Supporting this activity is very important as the identification of a chain of refinements is cumbersome and has a great impact on proving systems correctness and thus on meeting system requirements and safety properties. The novel presented approach consists in defining a refinement strategy according to the MVC pattern and made of several refinement steps, each of them handling one or more requirements. Modelling a system using a refinement approach requires to iterate multiple times until the concrete model obtained meets the requirements and proofs, and safety properties. This is very time consuming and usually little guidance is proposed to identify which parts to refine, how much to refine them and when. When considering specifically interactive systems our MVC-based approach is of great support for each of these tasks. Beyond, structuring models according to the MVC pattern supports the activity of composition and decomposition of requirement as, for instance, the view and the controller components may share common required properties. To the best of our knowledge, this work goes far beyond the state of the art, as there is no standard refinement approach (directed by the defined requirements) for developing interactive systems. For non interactive systems, some related work has highlighted the complexity and difficulty of identifying efficient refinement strategies and the associated high cost (in term of resources at development time) [Su 2017]. On the contrary, the benefits of using MVC as a development pattern is well known by developers but the formal methods community is not yet exploiting it enough in the formal development of interactive systems.

A detailed description on the refinement strategy using model-view-controller (MVC) to structure and design Event-B formal models of the interactive application is provided in [24][58].

3.2.3 Applications

This section presents a list of applications developed in the context of hybrid systems as well as interactive systems, taking into account the proposed generic framework based on extended Event-B that supports continuous features via *Theory plugin*, and the F3FLUID framework, and MVC structuring for Event-B models.

The generic formal framework has been exemplified through the development of numerous complex and simple case studies, such as a car with automatic braking system [56] and signalised left-turn assist system [55] [23], different types of water tank controllers taking into account *single-to-single*, *single-to-many* and *many-to-many* architecture design patterns [20][46, 48]. Furthermore, the approximation pattern [41] is demonstrated by the development of an inverted pendulum [44] and a robot [45]. In addition, the generic framework is also used to model a complex hybrid train speed controller [53]. The development life-cycle for rigorous development of critical systems is demonstrated through the development of an Insulin Infusion Pump case study [25]. Similarly, the F3FLUID framework has been exemplified through the development of an industrial case study issued from aircraft cockpit design, complying with the ARINC 661 standard [ARINC 661 specification 2002]: Multi-Purpose Interactive Applications (MPIA) [51], and to demonstrate the effectiveness, scalability, reliability and feasibility of the MVC Pattern for Structuring Event-B Models, we use two different case studies [58][24]. Moreover, all these case studies are elaborated in Chapter 5.

Summary of our contribution for modelling & designing framework and patterns:
We propose a generic formal framework, development life-cycle, F3FLUID (Formal Framework For FLUID) formal framework, and Event-B model structuring pattern using *model-view-controller* (MVC) for modelling, designing and verifying various classes of critical systems. The proposed framework is generic and extensible, and it consists of a method and a set of tools that engineers can apply and use when developing complex systems. These frameworks and patterns includes a large set of theories that extend Event-B with mathematical features required to model discrete and continuous behaviours, and a series of patterns based on refinement that allow for easier design. To demonstrate the usability of our framework and patterns, we use several complex case studies.

Project: ANR-DISCONT, IRT-INGEQUIP, ANR-FORMEDICIS
Student supervision: Ismail Mendil (PhD, 2019 – 2023), Guillaume Dupont (PhD, 2017 – 2021), Romain Geniet (MS, 2017), Sasan Vakili (M.A.Sc, 2015)
Publications: [20, 22–24][41, 42, 44–48, 51, 53, 55, 56, 58, 71, 74, 75][15][103]
Software: Fluid2EB, Theories, domain-specific theories, approximation, modelling and designing patterns, simulation framework, and models.
Models: `https://www.irit.fr/~Guillaume.Dupont/models.php`

3.3 Meta-modelling: Reflexive Event-B

3.3.1 Context

Meta-modelling is an engineering activity that enables the description of the core abstractions and properties to which models must adhere together with model analysis techniques. It has been widely adopted in the field of software engineering, particularly in model-driven engineering. Nowadays, formal methods have adopted such meta-modelling techniques for developing theories axiomatising metamodels to represent higher level reasoning concepts used in the specification, development and verification of complex systems [Bertot 2010, Sozeau 2020, Muñoz 1999, Fallenstein 2015].

There are several modelling languages that enable abstract reasoning about model properties while also working on concrete models. It is referred to as a reflexive relationship. Several formal techniques have already addressed the reflexive relationship, such as in Abstract State Machine (ASM) [Börger 2003] as ASM-Metamodel (AsmM) [Riccobene 2004] to represent core modelling constructs and semantics; Coq [Bertot 2010] with the syntactic representation of Coq in Coq with Template Coq [Anand 2018] and the semantics in MetaCoq [Sozeau 2020].

A similar approach exists for VDM [Jones 1986, Bjørner 1978] with MURAL [Jones 1991], an interactive mathematical reasoning environment extended to support VDM [Bicarregui 1991] specifications based on meta-modelling of VDM concepts. The reflection principle [Fallenstein 2015] is also implemented in Isabelle/HOL [Nipkow 2002] to build a HOL model within HOL to analyse and reason about various modelling concepts such as infinite hierarchy of large cardinals, polymorphism, verifying systems with self-replacement functionality, etc. In PVS, Miltra et al. [Mitra 2005] proposed *strategies* concepts for proving abstraction relation between automata, based on theories and templates. Ebner et al. [Ebner 2017] described the meta-programming framework used in Lean [Moura 2021], which is an interactive theorem prover based on dependent type theory. This framework provides a mean for reflecting object-oriented expressions into a metalanguage by extending Lean object language based on Lean modelling constructs. In [Paul van der Walt 2012], the authors presented reflection in Agda [Stump 2016] in the style of Lisp [McCarthy 1959], MetaML [Taha 1997], and Template Haskell [Sheard 2002], as well as several typed programming applications.

In the case of Event-B, the formalisation of *contexts* (and only contexts) in the Event-B language has been proposed using structural embedding [Bodeveix 2021]. In the same context, the B method has been embedded in PVS [Muñoz 1999], allowing users to benefit from B modelling power while also gaining access to the proving power of the PVS theorem prover [Owre 1992]. However, this embedding is not formalised, and leads to the use of two separate methods.

Event-B [Abrial 2010a] is a state-based formal method supporting the development of complex systems following a correct-by-construction approach. It is based on set theory and first-order logic, and it uses the Rodin integrated development environment. Currently, the core modelling features of the Event-B language enable abstract system modelling as state-transitions systems, refinement based development, and interactive and automatic proofs. There are also a number of other RODIN plugins available to help with other modelling requirements, such as composition/decomposition [Silva 2012], Theory plug-in [Abrial 2009, Butler 2013], code generation [88][Fürst 2014] etc. Among these plugins, the *Theory plug-in* offers powerful means to extend Event-B enabling for the development of additional data types, theories, and operators to extend the core modelling concepts and features of Event-B [37]. For example, Dupont et al. [44][20][Dupont 2021] have developed a set of theories to integrate continuous features in the Event-B modelling language for modelling differential equations.

Currently, Event-B framework only offers standard proof obligations (POs) that are generated automatically: invariant preservation, theorems proofs, variant decreasing, simulation, event feasibility, guard strengthening, etc. For additional verifications, such as deadlock freeness, liveness, event scheduling, reachability, and domain specific properties, the designer relies on other tools based on interactive proof systems and model checkers. They require ad hoc modelling from the designer for each formalised model. This process may be fastidious and must be repeated for each model to be analysed, making it not-reusable. Bacause, there is a lack of access to and explicit manipulation of Event-B concepts, it is impossible to express generic properties at a higher order level associated with extra reusable POs in a theory that permits *automatic* generation of such POs for any designed model.

3.3.2 Our contributions

In this context, we propose the first framework for reflexive Event-B: EB4EB [39]. EB4EB is an Event-B-based modelling framework that enables the explicit manipulation of Event-B features through the use of meta modelling concepts. To cover Event-B modelling language semantics, this framework relies on a set of Event-B theories that define data-types, operators, well-defined conditions, theorems, and proof rules. It allows for the manipulation of static and dynamic aspects of Event-B modelling features, to encode new proof obligations related to other types of properties once and for all. Deep and shallow modelling approaches are proposed to exploit this framework based on the instantiation of the introduced features at the meta level. In addition, a case study demonstrating the use of our framework using the deep and shallow embedding approaches is developed. The Rodin platform, which handles Event-B models and proofs, underpins the entire framework.

The EB4EB Framework

The main objective of the EB4EB reflexive framework is to provide explicit manipulation of the Event-B components as first-order objects, making it possible to reason on these objects and define new analysis techniques. In order to develop this framework, the concept of Event-B machine is formalised as a data-type in a theory (a meta-theory), together with a set of axiomatised operators. These axiomatic definitions formalise the state-based semantics of Event-B models (contexts and machines). In addition, the meta-theory is equipped with relevant proved (once and for all) theorems useful for discharging the generated POs. There are two possible instantiations, *shallow* and *deep* embeddings, that can be used to obtain an Event-B model by instantiating the defined meta-theory to define states, events, guards, invariants, variant and related properties. Furthermore, we must ensure the correctness of the instantiated models, so during instantiation, a set of POs related to well-definedness conditions is generated. These POs ensure the Event-B machine is consistent, including invariant preservation, event feasibility, variant progress, theorems hold, etc. as defined in the Event-B semantics.

THEORY EvtBTheo
TYPE PARAMETERS
$STATE, EVENT$
DATATYPES
$Machine(STATE, EVENT)$
CONSTRUCTORS
Cons_machine(
$Event : \mathbb{P}(EVENT)$,
$State : \mathbb{P}(STATE)$,
$Init : EVENT$,
$Progress : \mathbb{P}(EVENT)$,
$AP : \mathbb{P}(STATE)$,
$Grd : \mathbb{P}(EVENT \times STATE)$,
$BAP : \mathbb{P}(EVENT \times (STATE \times STATE))$,
$Inv : \mathbb{P}(STATE)$
$Thm : \mathbb{P}(STATE))$
$Variant : \mathbb{P}(STATE \times \mathbb{Z})$,
$Ordinary : \mathbb{P}(EVENT)$,
$Convergent : \mathbb{P}(EVENT)$,

Code Snippet 3.4: Event-B Machine Datatype

Meta-Theory for Event-B

The Event-B meta-theory `EvtBTheo` introduces the STATE and EVENT type parameters in Listing 3.4 to represent machine's states and events. These type parameters are used to define an Event-B machine in the DATATYPES clause, and then a constructor *Cons_machine* is defined in the CONSTRUCTOR clause to represent each machine component, such as events (*Event*, states (*states*), initial event(*Init*), progress events (*Progress*), after-predicates (*AP*), guards (*Grd*), before-after predicates (*BAP*), invariants (*Inv*), theorems (*Thm*), variants (*Var*), ordinary events (*Ordinary*), and convergent events (*Convergent*). Further, a set of operators is defined to ensure the correctness of machines. For example, Listing 3.5 excerpts some of the operators: *BAP_WellCons* to check that all progress events are associated with defined BAP; *Grd_WellCons* to check that progress events are possibly guarded; *Event_WellCons* to check that the initialisation event belongs to machine events; *Variant_WellCons* to check that the variant states are convergent; *Tag_Event_WellCons* to check that machine events are composed of an initialisation and progress events. At last, *Machine_WellCons* operator is defined as a conjunction of the operators to ensure a machine is well-structured.

The machine data-type offers operators to access and manipulate machine concepts. To complete the definition of a consistent machine, we need to introduce the well-definedness conditions that encode the behavioural semantics and correctness criteria. The other POs associated to Event-B machines are formalised as operators in the meta-theory, as shown in Listing 3.6. In general, these operators are defined inductively on the structure of a machine (for initialisation and progress events). All the POs of an Event-B machine are gathered in a conjunction in the *check_*-

BAP_WellCons $< predicate > (m : Machine(STATE, EVENT))$
 direct definition
 $dom(BAP(m)) = Progress(m)$
Grd_WellCons $< predicate > (m : Machine(STATE, EVENT))$
 direct definition
 $dom(Grd(m)) = Progress(m)$
Event_WellCons $< predicate > (m : Machine(STATE, EVENT))$
 direct definition
 $partition(Event(m), \{Init(m)\}, Progress(m)$
Variant_WellCons $< predicate > (m : Machine(STATE, EVENT))$
 direct definition
 $Inv(m) \lhd Variant(m) \in Inv(m) \rightarrow \mathbb{Z}$
Tag_Event_WellDefined $< predicate > (m : Machine(EVENT, STATE))$
 direct definition
 $partition(Event(m), Ordinary(m), Convergent(m), Init(m) \in Ordinary(m)$
 ...
Machine_WellCons $< predicate > (m : Machine(STATE, EVENT))$
 direct definition
 $BAP\_WellCons(m) \wedge$
 $Grd\_WellCons(m) \wedge$
 $Event\_WellCons(m) \wedge$
 $Tag\_Event\_WellCon(m) \wedge$
 $Variant\_WellCons(m) \wedge$
 ...

Code Snippet 3.5: Operators of well-defined datatypes

Machine_Consistency. This operator formalises the machine's behavioural semantics and general correctness. When this operator is used in a theorem clause, a well-definedness is automatically generated, together with the PO associated to the proof of the theorem. Proving the theorem ensures the consistency of the machine, defined as an instance of the meta-theory.

Finally, in recent work [35], we extended EB4EB framework to support new analysis, possibly non-intrusive, mechanisms associated to different properties not expressed in core Event-B. In this work, we present three properties, deadlock freeness, invariant weakness analysis and reachability, to demonstrate extension of reasoning mechanism using the reflexive Event-B. For reasoning about different properties of the Event-B model at the meta level, the EB4EB framework theories can be instantiated using either a deep or shallow embedding approach. Furthermore, our reflexive framework EB4EB has been extended to formalise and operationalise the automatic generation of proof obligations associated with liveness properties expressed in LTL. Moreover, we used the trace-based semantics to demonstrate the soundness of this formalisation, and provide a set of intermediate and generic theorems to increase the rate of proof automation for these properties [34].

3.3.3 Application

We have demonstrated the approach on the clock model developed in both core modelling language and deep and shallow modelling. Both deep and shallow instantiated clock models preserve the required safety properties and functional behaviour encoded in theories. In addition, our developed theories have been applied on several case studies [38] to assess the expressiveness, effectiveness, portability, and scalability of our approach. On various case studies, we used both deep and shallow embedding approaches, as well as extended reasoning mechanisms. The use of deep and shallow modelling approaches resulted in the generation of several new POs. The most of the generated POs are associated with an extended reasoning mechanism and well-defined conditions for each operator. All the generated POs are successfully discharged. Note that most of POs are discharged interactively simplifying the complex predicate.

Note that the EB4EB modelling concepts are formalised once and for all, meaning that they are reusable and don't need to be proven again. However, these theories must be instantiated in new developments, and the generated WD POs must be discharged to check instantiation is correct. Moreover, it is possible to conduct non-intrusive analysis for Event-B models in Event-B without resorting to another formal method requiring additional proofs to guarantee

Mch_INV_Init $< predicate >$ $(m : Machine(STATE, EVENT))$
 `direct definition`
  $AP(m) \subseteq Inv(m)$
Mch_INV_One_Ev $< predicate >$ $(m : Machine(STATE, EVENT), e : EVENT)$
 `well-definedness`
  $e \in Progress(m)$
 `direct definition`
  $BAP(m)[\{e\}][Inv(m) \cap Grd(m)[\{e\}]] \subseteq Inv(m)$
Mch_INV $< predicate >$ $(m : Machine(STATE, EVENT))$
 `direct definition`
  $Mch\_INV\_Init(m) \wedge (\forall e \cdot e \in Progress(m) \Rightarrow Mch\_INV\_One\_Ev(m, e))$
Mch_FIS_Init $< predicate >$ $(m : Machine(STATE, EVENT))$
 `direct definition`
  $Inv(m) \cap AP(m) \neq \varnothing$
Mch_FIS_One_Ev $< predicate >$ $(m : Machine(STATE, EVENT), e : Event)$
 `well-definedness`
  $e \in Progress(m)$
 `direct definition`
  $Inv(m) \cap Grd(m)[\{e\}] \subseteq dom(BAP(m)[\{e\}])$
Mch_FIS $< predicate >$ $(m : Machine(STATE, EVENT))$
 `direct definition`
  $Mch\_FIS\_Init(m) \wedge$
  $(\forall e \cdot e \in Progress(m) \Rightarrow Mch\_FIS\_One\_Ev(m, e))$
Mch_NAT_One_Ev $< predicate > \ldots$
Mch_NAT $< predicate > \ldots$ **Mch_VARIANT_One_Ev** $< predicate > \ldots$
Mch_VARIANT $< predicate > \ldots$
Mch_THM $< predicate > \ldots$
  $\ldots$
check_Machine_Consistency $< predicate >$ $(m : Machine(STATE, EVENT))$
 `well-definedness`
  $Machine\_WellCons(m)$
 `direct definition`
  $Mch\_INV(m) \wedge$
  $Mch\_FIS(m) \wedge$
  $Mch\_NAT(m)$
  $Mch\_VARIANT(m) \wedge$
  $Mch\_THM(m)$

Code Snippet 3.6: Well-defined datatype operators and machine consistency

the correct embedding of Event-B. In addition, we have enriched the RODIN prover with relevant and proved rewrite rules, included in tactics, leading to a high proof automation rate.

Summary of our contribution for reflexive Event-B:
We propose the EB4EB framework that allows users to manipulate Event-B features explicitly using reflection and meta-modelling concepts. It relies on Event-B theories that define data-types, operators, WD, theorems and proof rules to formalise the semantics of Event-B. The developed theories enable manipulation of the static and dynamic properties of Event-B, including new POs associated to deadlock freeness, liveness, reachability, composition/decomposition, and so on. Deep and shallow embedding are used to instantiate the defined theories of the EB4EB framework in the Rodin development environment. In addition, we extend the core reasoning mechanism of Event-B by developing new theories based on Event-B meta-modelling concepts to express deadlock freeness, ill-events and invariant strengthening, reachability, generation of new proof obligations associated with liveness properties expressed in LTL. Finally, the reflexive EB4EB framework is evaluated using various case studies.
Project: EBRP – Enhancing EventB and RODIN: EventB-RODIN-Plus (funded by ANR)
Student supervision: Peter Riviere (PhD, 2020 – continue)
Publications: [21][37, 39][34, 35]
Software: EB4EB framework, meta-theories, models
Models: https://www.irit.fr/~Peter.Riviere/research

3.4 Environment Modelling

3.4.1 Context

Patient safety is a global challenge that requires practical knowledge and technical skills in clinical assessments, embedded systems, and software engineering including human factors and systems engineering (HFE). Many incidents related to patient safety are due to lack of attention to HFE in the design and implementation of technologies, processes, and usability [Carayon 2009].

The medical device manufacturers use an artificial environment model for simulating and testing the functionalities and effectiveness of medical devices. Such type of environment models are very expensive and critical to use for testing and validating the medical devices. These models are based on complex mathematical equations, that require high computation and large memory for simulating the environment. However, these models are not able to simulate and to check the overall functionalities of a device. For example, an IIP requires an interactive glucose homeostasis environment to verify the correctness of system behaviour. Medical devices are tightly coupled with the biological environment in which they are designed to work. They use actuators and sensors to respond to abnormal behaviours in the biological environment, and we can observe the resulting behaviour in the biological environment (by observing the behaviour of the model) to ensure that the system behaves correctly under the required conditions. This approach is clearly dependent on the fidelity of the model of the biological environment. If the model is accurate, this approach can help to provide us with assurance that the behaviour of the device is safe within that environment, and will effectively achieve its purpose. It is important to know that, these models are also not applicable to use at the early phases of the development life-cycle for testing or verifying the requirements. We need to develop the closed-loop model by using an abstract model of the virtual environment. The closed-loop model is a combined model of the medical device model and the environment model, where both models interact to each other using sensors and actuators. The designed abstract model should capture all the essential features.

Clinical models are used for identifying and predicting the various stages of diseases like diagnostics, control, progression, complication etc. Bolie et al. [Bolie 1961] presented the first mathematical model based on differential equations to model the glucose and insulin concentration, illustrating the dynamics of insulin-glucose for diagnostic purpose and evaluating several parameters of the diabetic and pre-diabetic conditions. Silber et al. [Silber 2007] proposed an integrated insulin-glucose model for analysing the diabetic condition using a bidirectional insulin-glucose feedback mechanism. Chay et al. [Chay 1985] proposed the theoretical treatment of the effect of external potassium on oscillations in the pancreatic β-cells, which can be used to demonstrate that insulin infusion may be useful for mimicking pancreatic insulin secretion. Several other models have been developed that incorporate different physiological processes associated with insulin-glucose dynamics and different variations [Ajmera 2013, Han 2012, De Gaetano 2000, Drozdov 1995].

The literature suggests that existing models, with their mathematical constraints and higher order differential equations, are not easy to express in first order logic, and thus make it difficult to express the system requirements for verification purpose. Moreover the existing models have been developed for specific purposes that cannot support desired global behaviours. We want to describe the complete system by introducing the abstract notions of possible features that can be later extended for any particular use. However, we are motivated and encouraged by our previous work on heart modelling [89][1][98] that presents an abstract notion of complex heart behaviours. We have adopted the same methodology to design an efficient and optimum environment model for the GH system using formal techniques. To our knowledge, there does not exist any environment model for homeostasis system based on formal methods that can be used for validation/verification at the early stage of system development.

3.4.2 Our contributions

In this context, we propose the development of a virtual environment model [52, 69, 77], closed-loop modelling [14], a virtual environment model for verification, simulation and clinical trials [65][8], supporting techniques and tools [61]. A virtual environment model, Glucose Homeostasis (GH), for verification, simulation and clinical trials are summarised below. In addition, several applications are also discussed to demonstrate our approaches.

A Virtual Environment Model for Verification, Simulation and Clinical trials

We propose an abstract development of virtual environment model of the glucose homeostasis for diabetes patients to analyse the patient specific medical devices, such as an Insulin Infusion Pump (IIP) [52, 65, 77][8]. This research focuses on a methodology to develop an environment model for the GH system that is based on *logico-mathematical* theory and allows for the verification and validation of system requirements [89][1][98]. The model is developed using an incremental refinement approach that helps to introduce several properties in a progressive way, and to verify the correctness of the GH model under normal and abnormal behaviours (hyperglycemia, hypoglycemia or diabetic complications).

Glucose Homeostasis System

Glucose is the major metabolic fuel of the human body. To maintain an appropriate level of glucose in the body and to provide normal functionality, we need a regular supply of glucose to the body. Failure of the glucose level causes several diseases such as diabetes mellitus, galactosemia and glycogen storage diseases [Ajmera 2013].

Fig. 3.10 depicts the normal GH system[3], which presents the structural flow of the hormones and a functional behavioural pattern of the different organs. It is vital for the body to maintain an appropriate glucose concentration, so both low and high glucose levels are serious, life-threatening problems. The body regulates its glucose concentration using the pancreas and liver. The pancreas produces two main hormones *insulin* and *glucagon* to control the GH system. The body cells use the available glucose whenever the body receives glucose from the infusion or hepatic function. There are two different type of cells that use the glucose. For instance, the brain and nervous system cells use glucose without insulin, while other type of cells like muscle and fat use glucose with the help of insulin. The glucose concentration level fluctuates in the body, and is maintained in the plasma through the pancreatic secretion of glucagon and insulin. In general, the body attempts to maintain an appropriate level of glucose in the body, but there are some natural stable oscillations that occur in the glucose and insulin concentrations [Li 2006].

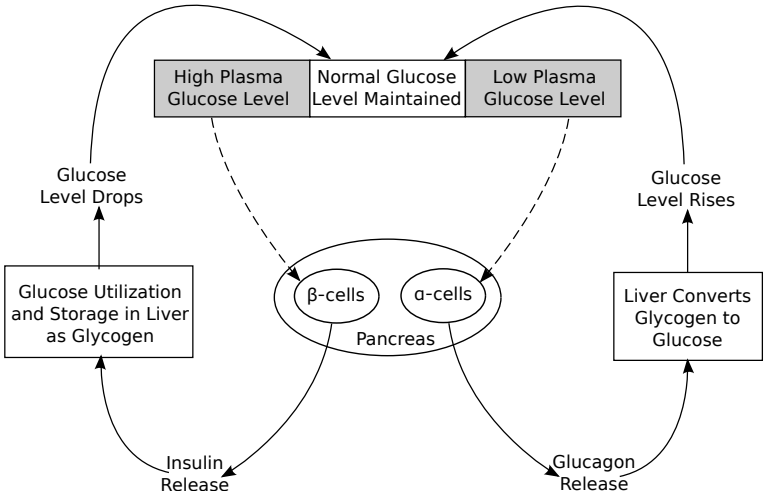

Figure 3.10: The GH System (adopted from [Ajmera 2013])

Low and high glucose levels are the two main biological responses that the body uses to maintain an appropriate plasma glucose concentration. When the glucose level drops, then the α-cells in the pancreas produce glucagon, which is transformed into glucose with the help of the liver. This process helps to increase the glucose concentration in the body. Similarly, when the plasma glucose level goes higher than expected, then the β-cells in the pancreas are stimulated to lower the glucose concentration [Ajmera 2013]. This stimulation process can be completed within 5 to 15

[3]The 'normal GH system' is when the GH system functions as it should, i.e., there are no abnormal behaviours exhibited by the system.

minutes, and during this period the insulin is produced by β-cells of the pancreas. The secreted insulin can be used by insulin dependent cells to utilize the available glucose, and to stop the natural hepatic glucose production for reducing the glucose concentration in the blood. The liver is the central organ for regulation of glucose and glycogen and behaves as a distributor of nutrients through blood to other tissues. The presence of insulin inhibits the transformation of glucagon to glucose.

Formalising GH

Our proposed method describes a GH model based on logico-mathematics to help the formal community verify the correctness of IIP models. The GH model is mainly based on the glucose regulation system of the body. This method uses advance capabilities of the combined approach of formal verification and behaviour simulation, in order to achieve considerable advantages for GH system modelling. Fig. 3.10 shows the main components of the GH system. The system comprises different states of the glucose level in the blood and biological organs, in order to control the glucose level. To formalise the GH system, we consider eight significant landmark nodes (*Hi, No, Lo, Ac, Bc, Li, St, Tr*) in the homeostasis functional network as shown in Fig. 3.11, which can control the GH system. We have identified these landmarks through a literature survey [Ajmera 2013, Li 2006, Bolie 1961, Silber 2007], and use them to express an abstract functionality of the system. We introduce the necessary elements to formally define the GH systems as follows:

Definition 4 ***(The GH System).*** *Given a set of nodes N, a transition T, is a pair (i, j), with i, j* $\in N$ *. A transition is denoted by i* $\rightsquigarrow$ *j. The GH system is a tuple GHS = (N, T,* N_0*) where:*
• *N = { Hi, No, Lo, Ac, Bc, Li, St, Tr } is a finite set of landmark nodes in the GH network;*
• $T \subseteq N \times N$ *= {Hi* $\mapsto$ *Bc, Lo* $\mapsto$ *Ac, Bc* $\mapsto$ *Li, Ac* $\mapsto$ *Li, Li* $\mapsto$ *St, Li* $\mapsto$ *Tr, St* $\mapsto$ *No, Tr* $\mapsto$ *No, St* $\mapsto$ *Hi, Tr* $\mapsto$ *Lo, Tr* $\mapsto$ *Hi, is a set of transitions to present data flow between two landmark nodes. It should be noted that the last three transitions are possible when we consider the case of failure of the GH system;*
• N_0 *= No is the initial landmark node (normal glucose level);*

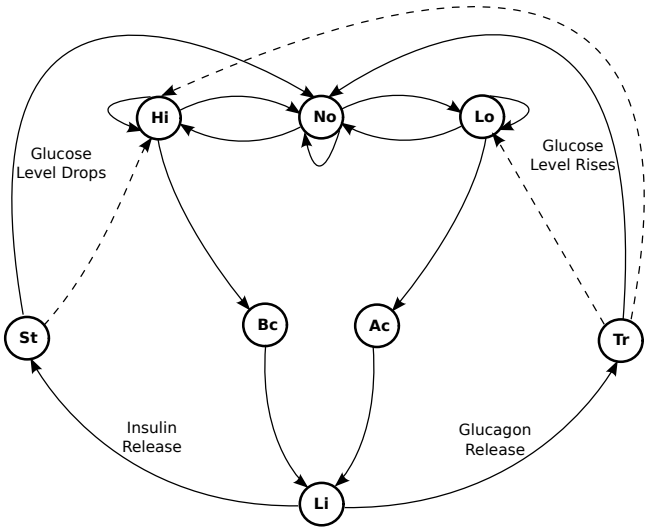

Figure 3.11: The GH Automata

The automata shows the flow of the GH system, where by default the GH system is considered to be in its normal state (No). The normal state indicates that there is an appropriate glucose level in the blood. Whenever the glucose level fluctuates in the blood, resulting in a high or low glucose level, the GH system controls the fluctuated glucose level with the help of the pancreas and liver. The high and low states are presented by Hi and Lo nodes (see Fig. 3.11).

The pancreas has two type of cells: α-cells and β-cells, which are indicated by the Ac and Bc nodes, respectively. The liver is denoted by the Li node that is used to convert the glycogen to glucose using glucagon, and to store the glucose as glycogen in the liver with the help of insulin. If the liver is well behaved, then the glucose level either rises or drops according to whether there is a low or high glucose level in the blood, respectively. Eventually, the glucose level returns to an appropriate level.

Diabetes or Abnormal Homeostasis System. Fig. 3.12 presents abnormal behaviours of the GH system. The liver plays a central and crucial role for regulating the glucose level in the blood. The main task of the liver is the continual supply of required glucose energy sources to the body. Failure of the GH system causes several diseases, and in particular, diabetes. There are two type of diabetes: *insulin-dependent diabetes* (also know as *type 1 diabetes*) and *non insulin-dependent diabetes* (also know as *type 2 diabetes*). Insulin-dependent diabetes may be caused by insufficient or no insulin secreted due to β-cells defects. In non insulin-dependent diabetes, insulin is produced, but the insulin receptors in the target cells do not work due to insulin resistance in the cells, so the insulin has no effect. In both cases there can be a very high glucose level in the blood. Low glucose level can be caused by α-cell defects or abnormal glucagon release, which can be further classified as insufficient or no glucagon secretion, excess insulin, and excess glucagon secretion. Excess glucagon secretion and defects in β-cells may also indicate a persistent high glucose level, which can be classified as hyperglycemia-induced diabetes complications [Ajmera 2013].

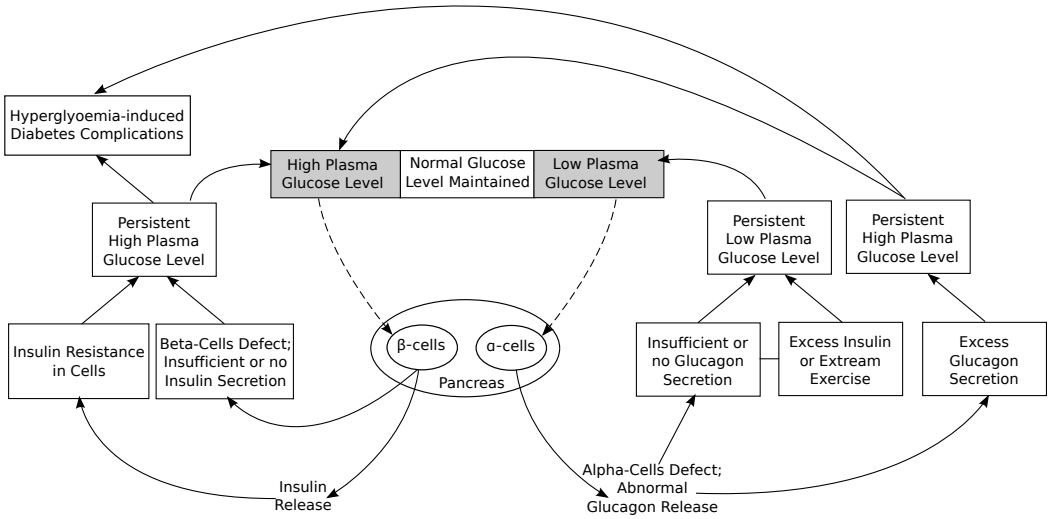

Figure 3.12: Abnormal GH System (adopted from [Ajmera 2013])

Blood Sugar Concentration. The blood sugar concentration or blood glucose level is an amount of glucose (sugar) present in the blood of the body. The body naturally regulates blood glucose levels as a part of metabolic homeostasis. The glucose level fluctuates many times in a day. In general, the glucose level is always low in the morning, and it can rise for about an hour after having a meal. There are two types of tests used to detect abnormal behaviours: FPG (Fasting Plasma Glucose) Test and the OGTT (Oral Glucose Tolerance Test) [MD 1975]. The FPG test is used to detect diabetes and prediabetes. The FPG test measures blood glucose in a person who has fasted for at least 8 hours and is most reliable when given in the morning. The OGTT can be used to diagnose diabetes, prediabetes, and gestational diabetes. This test is applied when a person has fasted for at least 8 hours and 2 hours after the person drinks a liquid containing 75 grams of glucose dissolved in water. The normal glucose level should be within the range of 70 mg/dL to 99 mg/dL for a non-diabetic person using the FPG test, while the glucose level should be within the range of 70 mg/dL to 139 mg/dL for a non-diabetic person using the OGTT [MD 1975]. In the case of low glucose level, for both FPG and OGTT tests the glucose level should be within the range of 0 mg/dL to 70 mg/dL. Similarly, for a high

glucose level, readings should be greater than 126 mg/dL in the FPG test, and greater than 140 mg/dL using the OGTT. A blood sugar level outside of the normal range indicates an abnormal glucose concentration. A high level of glucose is referred to as hyperglycemia and a low level of glucose is referred to as hypoglycemia.

Property 1 ***(Glucose level in blood).*** *The blood glucose level defines different stages, such as hyperglycemia, hypoglycemia and normal. We say that the glucose level is low (hypoglycemia) if* $FPG \in 0 \,..\, 69$ *or* $OGTT \in 0 \,..\, 69$*, and the glucose level is high (hyperglycemia) if* $FPG \geq 126$ *or* $OGTT \geq 200$*, and the glucose level is normal if* $FPG \in 70 \,..\, 99$ *or* $OGTT \in 70 \,..\, 139$*. We classify pre-diabetes to be the range where* $FPG \in 100 \,..\, 125$ *or* $OGTT \in 140 \,..\, 199$*.*

Formalisation of the GH System. To develop a virtual biological environment of GH based on formal techniques, we use the Event-B modelling language [Abrial 2010a] that supports an incremental refinement to design a complete system in several layers, from an abstract to a concrete specification. Initial model captures the basic behaviour and biological requirements of the GH system in an abstract way. The subsequent refinements are used to introduce α-cells and β-cells of the pancreas, functional behaviour of liver to convert and to store the glucose, abnormal conditions of the pancreas, diabetic conditions, and diabetes complications, and blood sugar concentration for assessing diabetes. The developed system results the dynamic behaviours of virtual GH biological environment that covers the both normal and abnormal behaviours (hyperglycemia, hypoglycemia or diabetic complications). A list of safety properties is defined at each incremental level to guarantee the correctness of designed virtual biological environment model for GH. A detail formalisation process of the virtual biological environment model is available in [77].

Model Validation and Analysis. This section presents validation of the developed model through animation, using a model checker tool ProB [Leuschel 2003], and the generated proof obligations. Validation, in this context, is a process that shows consistency between formal models and requirements. This tool enables us to validate the GH model according to the glucose fluctuation in the body. We have validated different kinds of scenarios of normal and abnormal glucose levels. In order to test the abnormal behaviour of the GH system, we have also validated the diabetics, prediabetics, and diabetics complication conditions.The ProB tool is not only used for animation, but it also verifies an absence of error, for example (no counter example exists) and no deadlocks at each level of developed model from abstraction to the final concrete model.

Model	**Total number of POs**	**Automatic Proof**	**Interactive Proof**
Abstract Model	16	16(100%)	0(0%)
First Refinement	13	6(46%)	7(54%)
Second Refinement	7	6(86%)	1(14%)
Third Refinement	25	24(96%)	1(4%)
Fourth Refinement	62	60(97%)	2(3%)
Total	**123**	**112(91%)**	**11(9%)**

Table 1. Proof Statistics

Table 1 shows the proof statistics of the development in the RODIN tool. In order to guarantee the correctness of the system behaviour, we established various invariants in the incremental refinements. This development results in 123(100%) proof obligations, in which 112(91%) are proved automatically, and the remaining 11(9%) are proved interactively using the Rodin prover. These proofs are quite simple, and can be achieved with the help of simplifying predicates. An incremental refinement of the GH system helps to achieve a high degree of automatic proof.

The main usability of environment modelling for developing an IIP as follows: Verifying patient safety in closed-loop; checking functional requirements; analysing clinical requirements; and finding essential safety properties.

Development of Glucose Homeostasis Simulator

In this modern age, computer simulations have become a standard approach for medical application, particularly for understanding the biological organ behaviour and research in medical domain. Our objective is to use the generated

code from the verified formal model to design a simulation for GH. Fig. 3.13 depicts a GH simulation framework. The proposed simulation framework may allow us to cover the basic functionalities of GH and the required physiological behaviour of biological organs, such as liver, pancreas.

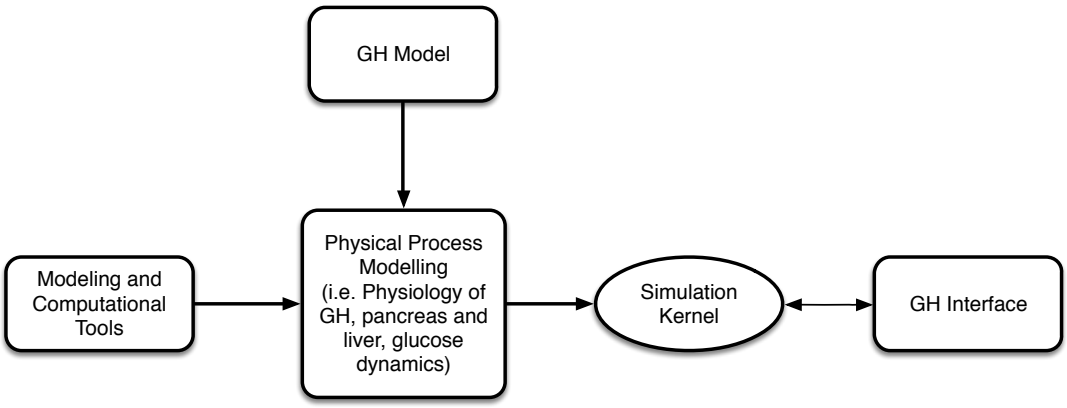

Figure 3.13: GH Simulation Framework

The liver and pancreas play a central role to control the glucose level in the blood. These are the compound organs with a unique structure, essential for digestion and hormonal regulation in the body. The muscles of the liver and pancreas are the main components of the GH simulation which can be modelled as solid and fluid mechanics models. More robust and widely accepted tools are computational fluid dynamic (CFD) [CFD-ACE+ , COMSOL Multiphysics] and finite element analysis (FEA) [COMSOL Multiphysics , Smith 2009]. The use of CFD tool helps modeller to understand and mimic the fluid flow physical system and the FEM can be used to solve the physical systems related to the physical structure, stress and strain. CFD and FEA software tools can be used together to model any physical biological system. In our simulation framework, these tools can be used to design and develop the physical model of GH. The physical process modelling is defined through the physiology of GH, physiology of the pancreas, physiology of the liver, insulin-glucose dynamics, glucagon-glucose dynamics, GH abnormality, and physiology of pancreatic α-cells and β-cells represented in the round rectangular box. This physical process model is derived from the use of modelling and computation tools, and the generated source code of GH. Note that the use of computation tools and generated code of GH can be used for deriving any specific simulation model related physical process of GH. The simulation kernel is the heart of the proposed simulation framework that plays a central role for simulating a physical model. The simulation model is designed and supervised by the simulation kernel, which arbitrates their communication between the components. The part which coordinates the top level objects is provided by the user. The user defined coordination involves possible execution order of the models and the required interfacing scenarios. It allows the user to properly interpret the semantics of the top level objects and the required interfacing scenarios based on the specific application that the simulation is being developed for. The user interface of GH allows us to display the possible simulations and possible interactions for basic user input and output operations. This user interface shows the required behaviour as per the results of the developed formal model and the given properties to evaluate the physical behaviour of GH. Note that the developed simulation using this framework can be used to simulate a complex model of GH considering various complex scenarios. These complex scenarios represent dynamic functions of the different components of GH.

Implementation of the GH System

Fig. 3.14 depicts an implementation framework of GH on a hardware platform. The modelling and implementation unit of GH is represented in the rounded rectangular box, which is derived from both the formal specification of GH and GH simulation, which contain the formal and simulated models of the physical processes in form of discrete and

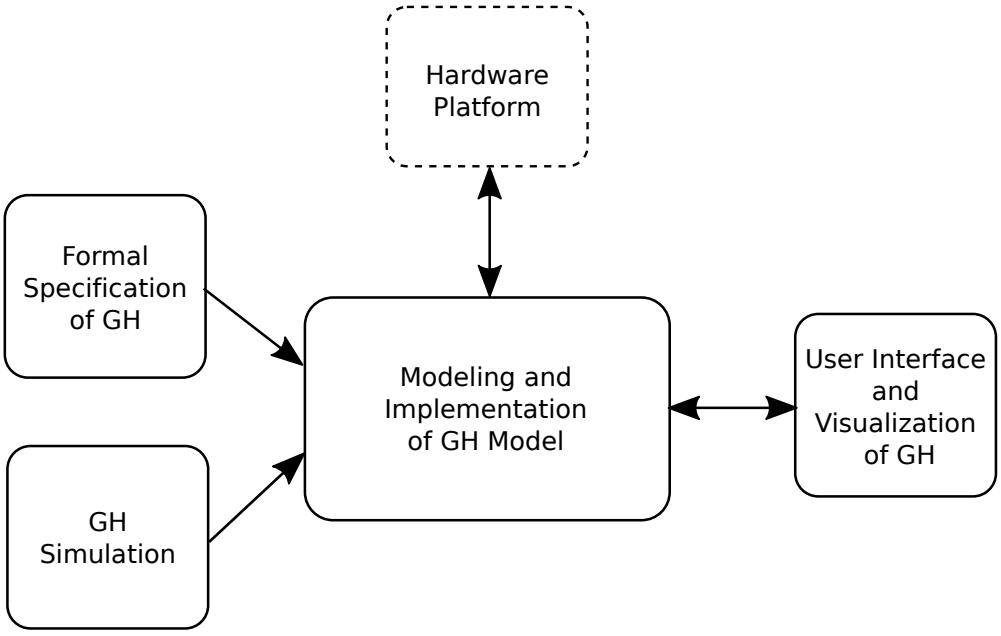

Figure 3.14: Hardware Implementation of GH

continuous behaviour. The formal models contain an abstract model and a set of refined models that can be further enriched through the introduction of complex expressions. The modelling and implementation blocks use existing tools like Matlab [MATLAB] or Labview [LabVIEW] to implement the GH, and then further it can be embedded on a hardware platform, such as FPGA [FPGA], Arduino [Arduino], Snickerdoodle [Snickerdoodle]. This block also communicates with a hardware platform, and the block of user interface. The user interface and visualisation of GH component provides an interface for basic user input through the haptic interface tools and basic output through visualisation tools according to the embedded GH model on the hardware platform.

In addition, we also need to develop a GUI interface for GH model to animate and to visualise the required functional behaviour, physiological activities of pancreas, liver, β-cells and α-cells, and dynamics of insulin-glucose and glucose-glucagon dynamics. The virtual environment model of GH implemented on a programmable hardware platform allows the user to use directly it with the medical devices, such as IIP, for the clinical trial. The virtual GH model is embedded in the micro-controller to imitate the desired patient specific conditions. The signal can output through the selected hardware platform (i.e. FPGA, Arduino) to an IIP, in which the IIP reacts similarly to the way a real medical device would react. The approach will provide the complete dynamics of an entire system that can be assessed in the testing or checking the patients' data for analysing the patient specific conditions. For instance, any patient can have diabetes for any specific cause, and we need several patients to assess the possible conditions. However, the developed virtual GH model can provide all possible scenarios in a single model. This allows comprehensive testing of the GH model to assist in the development of algorithms accordingly. Moreover, this developed virtual GH hardware platform can be used with other devices to visualise and check the required feature or behaviour of the system.

3.4.3 Applications

The GH virtual environment model is used for verifying an Insulin Infusion Pump (IIP) [25][9] developing the closed-loop model. The developed closed-loop formal model assist for validating the system requirements, finding missing requirements, validating assumptions and strengthening the existing requirements during the process of requirement engineering of IIP.

Summary of our contribution for developing medical systems:
We propose the formal development of a GH virtual environment model, simulation framework and hardware implementation. The formal model is used for checking the IIP requirements. The proposed simulation framework can be used to develop a simulation model of GH based on complex expressions using linear and nonlinear equation and the developed formal model. The hardware implementation architecture can be used to implement the GH on the hardware platform using the developed formal and simulation models. The GH virtual environment model can be embedded on the hardware platform used as a test bench for IIPs that can be used for clinical trials. This is the first computational model based on logical concepts to simulate the GH behaviour in order to analyse the normal and diabetic conditions. The developed model highlights a different aspect of the problem, making different assumptions and establishing different properties concerning the variation in glucose levels, normal and diabetic conditions, and malfunction of biological organs like liver and pancreas.
This is a promising simulated biological environment model that can be used during the development of the product life-cycle. Moreover, this developed virtual environment model may aid in certification process for the medical devices related to the homeostasis system, such as IIPs. This environment model can also be used as a diagnostic tool to diagnose or understand the patient requirements.
Project: Certification of Safety Critical Software-Intensive Systems (funded by Ontario Research Fund – Research Excellence (ORF-RE), and IBM), Centre for the Engineering of Complex Software-Intensive Systems, NECSIS (funded by Automotive Partnership Canada (APC)).
Student supervision: Mischa Geven (M.A.Sc, 2014), Nicholas Proscia (M.A.Sc, 2014)
Publications: [25][14][52, 61, 65, 69–72, 77, 84][9]
Software: models.

3.5 EB2ALL: Code Generation

3.5.1 Context

Formal methods aim to produce zero-defect software by controlling the entire software development process, from specification to implementation, by providing a solid mathematical foundation for system requirements descriptions. The ability to perform formal and automated verification of safety properties in formal models prior to code transformation has added significant value to industrial systems, including hardware and software systems.

Due to the limited size of memory for translating from formal specifications to given target programming languages, several constraining requirements exist, particularly in the embedded domain. Furthermore, due to the expressiveness of formal specifications, developing safety-critical systems is extremely difficult, as is producing equivalent source codes that can meet the required properties and the defined specifications. To overcome such issues, a balance must be struck between the expressiveness of the formal implementation language and the ease of the translation process.

Event-B [Abrial 2010a] is a correct-by-construction modelling language that allows for the progressive design of complex systems by refining an abstract model into several refined models to obtain a concrete model that is very close to source code and can easily be transformed into code. Currently, the B [Abrial 1996] and Event-B [Abrial 2010a] languages are successfully used in the core development of safety critical systems [Butler 2020], so there is an increasing demand for automatic code generation. In this direction, several code generators have been proposed. Bert et al. [Bert 2003] develop a method for producing efficient code from B models for the target domain, such as smart card applications, by adapting the B0 language to include language types and optimisations. In [Steve 2009], the authors propose the development of the B2C tool for generating C source code from Event-B models. Edmunds et al. [Edmunds 2011, Edmunds 2010] propose code generator for concurrent programs from Event-B using tasking and shared machines with the use of refinement and decomposition. This work is also extended in [Dalvandi 2019]. Ostroumov et al. [Ostroumov 2011] present an approach for generating VHDL code from Event-B models by analysing the Event-B model's core structure and hardware description language instructions in order to obtain the same be-

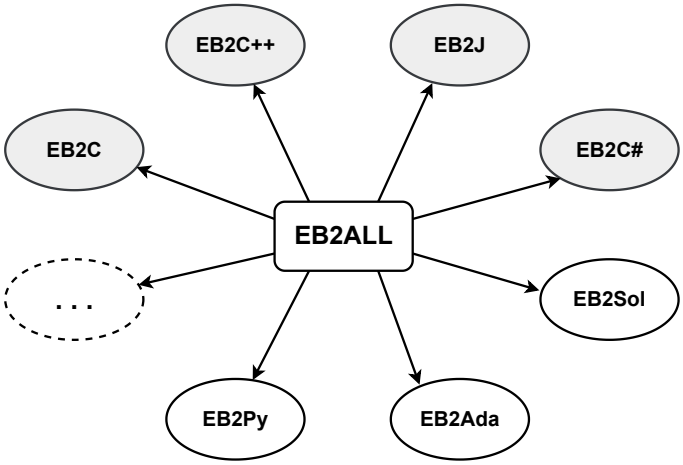

Figure 3.15: EB2ALL Extension plug-ins

haviour of generated VHDL code with respect to the Event-B model. In [Fürst 2014], the authors present an approach to generating program code from Event-B models that ensures correctness by combining well-definedness constraints, refinement, and assertions, as well as the introduction of a scheduling mechanism. In [Rivera 2017], the authors present code generation from Event-B to JML-annotated java program, including tool development.

Most of the approaches mentioned above lack tool support, or the developed tools are no longer maintainable to support the current version of Rodin IDE.

During my thesis work, I developed the EB2ALL[4] tool set, which includes the EB2C, EB2C++, EB2J, and EB2C# plug-ins for generating source codes for the C, C++, Java and C# programming languages. This is one of the most well-known and up to date plug-ins for the current Rodin IDE [88].

3.5.2 Our contributions

In this context, we propose a new extension of the EB2ALL [86, 88, 91][1] by developing a new plug-in, EB2Sol [5], for generating Solidity code from Event-B models [11] for ethereum platform, as well as extending existing plugins (EB2C, EB2C++, EB2J, and EB2C#) to integrate new modelling constructs and to keep all plug-ins up to date with respect to new Rodin IDE changes. Since 2010, I have worked as a sole developer to improve and correct reported errors. The development of EB2Sol and the integration of advanced modelling constructs are summarised below.

EB2ALL - Past, Present, and Future

In 2011, I developed the EB2ALL [86, 88, 91][1] tool as part of my thesis work. The initial version of the tool includes four main plugins: EB2C, EB2C++, EB2J, and EB2C#, which generate code in C, C++, Java, and C#. Since 2011, all of these plugins have evolved by incorporating advanced features and more modelling constructs. The current state of the EB2ALL is depicted in Fig. 3.15, where gray colour ovals represent plugins that were developed and new features and modelling constructs were added over the last ten years, and white colour ovals represent new plugins. The new plug-ins are EB2Sol, EB2Ada, and EB2Py, which generate code in the Solidity, Ada, and Python programming languages, respectively. It should be noted that EB2Sol, EB2Ada and EB2Py are fully developed and integrated into the core of EB2ALL. In addition, we are continue extending the EB2Py plugin to support remaining modelling elements of the Event-B language to support animation for Event-B models of any scale. Furthermore, we

[4] http://singh.perso.enseeiht.fr/eb2all
[5] http://singh.perso.enseeiht.fr/eb2all/eb2sol

also intend to add more programming languages denoted by "..." in the near future. The following section describes the core development of EB2Sol as well as the extension of existing plug-ins.

EB2Sol: Event-B to Solidity Smart Contract

Ethereum [Wood 2014] is an open-source computing platform, also known as the Ethereum Virtual Machine (EVM), for blockchain technology with contract functionality. The blockchain technology has been adopted successfully in different business sectors, such as e-commerce, banking, finance, insurance, energy trading, healthcare services, and asset management. Manipulation of critical transactions, as well as management of digital assets, making them attractive targets for security threats and attacks, which may result in financial losses and data leakage. EVM, on the other hand, executes bytecode of smart contracts written in Solidity [Solidity Documentation 2023, Solidity Github 2023], a JavaScript-like language, on a simple stack machine to handle and transfer digital assets, which is extremely difficult due to Ethereum's openness, allowing both programs and anonymous users to call into the public methods of other programs. In such cases, the use of trusted and untrusted code together in a large and complex application, particularly one involving financial management or privacy data, can be dangerous. For example, TheDAO is hacked by an attacker by examining EVM semantics to transfer 50 million USD in Ether [Zhao 2017].

Following several attacks [Atzei 2017, Zhao 2017] in recent years, formal methods are now regarded as first-class citizens for mitigating potential risks through formal reasoning on defined contracts. Several approaches based on formal methods for the development of smart contracts have been proposed in recent years. Palina et al. [Tolmach 2021] provided a comprehensive overview of formal models and smart contract specifications. They also highlighted some of the identified challenges and gaps in order to guide future research in the area of formal methods for developing trustworthy smart contracts. In [Hildenbrandt 2018], the authors proposed EVM semantics in the K Framework based on the ERC20 Standard Token for formalising and analysing smart contracts. Hirai et al. [Hirai 2017] defined EVM in Lem language that can be translated into many standard interactive theorem provers. In particular, they prove interesting safety properties of Ethereum smart contracts in Isabelle/HOL. The ConCert framework [Annenkov 2021] was developed for verifying smart contracts in Coq in order to detect vulnerability. In [Le 2018], the authors proposed a lazy approach to determining input conditions under which the contract terminates or not by statically proving conditional termination and non-termination of a smart contract. This is accomplished by ensuring in advance that both the current state and the contract's input satisfy the termination conditions. Wang et al.[Lahiri 2018] presented VERISOL, a formal verification tool for smart contracts verification based on semantic conformance of smart contracts against a state machine model with access-control policy. They discovered some previously unknown bugs in the published smart contracts, then fixed the bugs and would be able to perform model checking-based verification with VERISOL. In [Alt 2018], the authors described an SMT-based formal verification module integrated with the Solidity compiler for identifying potential bugs during the compile time, such as arithmetic overflow/underflow, unreachable code, trivial conditions, and assertion fails. In [Bhargavan 2016], the authors presented a framework for analysing and verifying the run-time safety properties as well as functional correctness of Ethereum contracts using the F* functional programming language. In [Grishchenko 2018], the authors showed small-step semantics of EVM bytecode in F*. They validated the executable code against the Ethereum test suite. Furthermore, they identified some bugs and defined several security properties to prevent them, such as call integrity and atomicity. In a similar vein, [Zhu 2020] presented a mechanism for translating Solidity contracts to Event-B models by defining transfer functions covering a subset of the Solidity language. Further, the produced Event-B model can be refined at different abstraction levels to verify properties associated with Solidity contracts using Rodin [Abrial 2010b].

A structured approach to smart contracts verification based on refinement in the Event-B modelling language proposed in [Banach 2020]. Our work is also in this vein, as we propose a framework based on Event-B formal methods for specifying, analysing, verifying, and implementing smart contracts through refinement by preserving the required safety properties [11]. In addition, we have developed a prototype tool, EB2Sol, to generate Solidity smart contracts from verified Event-B models. As far as we know, this is the first tool for translating Event-B models into Solidity smart contracts.

In the following section, we will go over technical details about implementation as well as design decisions made

during prototype development of EB2Sol.

Solidity

Solidity is a programming language designed specifically for developing smart contracts that compiled into byte-code executable by Ethereum platform's execution engine, often known as Ethereum Virtual Machine (EVM). Smart Contracts are programs that execute on a decentralised network without the intervention of a central authority. Solidity enables developers to create self-enforcing business logic via smart contracts, resulting in a trustworthy and authoritative record of transactions. The syntax of Solidity is quite similar to those of scripting languages like JavaScript, and it is heavily influenced by C++, Python. Solidity extensively utilises programming techniques derived from other languages. It features variables, static typing, functions, libraries, and interfaces. In addition, it offers a range of control structures such as for, while, do-while, and if-else. In case of an object-oriented programming language like Java, programmers work with classes, whereas Solidity programmers deals with contracts. Each contract can define various Solidity constructs such as state variables, functions, function modifiers, events, errors, structure types, and enum types. This subsection outlines the major Solidity constructs that are relevant to our goal. More information about Solidity programming language can be found in [Solidity Documentation 2023].

Solidity Types, Special Functions and Variables. Solidity is a statically typed programming language, which means that variable types are declared explicitly and thus determined at compile time. In Solidity, numerous elementary types exist that can be combined to form more sophisticated types. Solidity offers an extensive range of types, notably value types, reference types such as arrays and structures, mapping types, user-defined types and it also supports elementary type conversion. Solidity does not support undefined or null values, newly declared variables always have a default value based on their type. Table 3.5 summarises the most frequently used value types.

Value Types	Keyword	Description
Boolean Type	$bool$	possible values $true$ and $false$
Integer Types	int**X**	signed integers where **X** varies from 8 to 256 in steps of 8
	$uint$**X**	unsigned integers where **X** varies from 8 to 256 in steps of 8
Address Types	$address$	Ethereum address of 20 bytes
	$address\ payable$	additional members like $transfer$ and $send$
Byte Arrays (Fixed Size)	$bytes$**X**	**X** varies from 1 to 32
Byte Arrays (Dynamic Size)	$bytes$	similar to $bytes1[]$ but skips padding
	$string$	similar to $bytes$, but don't allow $length$ or index access
Enumerated Type	$enum$	default value is the first member

Table 3.5: Solidity Value Types

In the global namespace, special variables and functions exist at all times and are primarily used to relay information about the blockchain or to perform general-purpose utility operations. Table 3.6 lists some of the most frequently used variables and functions from the global namespace of Solidity. Additionally, solidity supports various denomination of Ethereum cryptocurrency such as wei, $gwei$, and $ether$, as a suffix to number literals.

Errors are handled in Solidity using state-reverting exceptions. This exception nullifies any state changes made during the execution of the code and notifies the caller of an error. The assert and require functions enable programmers to check for conditions and throw exceptions if they are not met. The assert function should be preferred exclusively to check for internal errors and invariants. The require function supports optional error message and should be used to ensure the existence of valid conditions that must be identified during contract execution. This includes conditions on input values or the return values of external contract calls. The revert function is another mechanism available in the global namespace for reporting errors and undoing state changes during contract code execution. Additionally, this function accepts and returns an optional message containing details about the error to the caller.

Type	Variable/Function	Description
$address$	$msg.sender$	sender of the message (current call)
$uint$	$msg.value$	number of wei sent with the message
-	$assert(bool)$	used for internal errors
-	$require(bool, [message])$	used for checking condition on input
-	$revert([message])$	abort execution and revert state changes
$uint$	$< address > .balance$	balance of the $address$ in wei
-	$< address\ payable > .transfer(uint)$	send given amount of wei to Address
$bool$	$< address\ payable > .send(uint)$	send wei to $address$, returns $false$ on failure
Contract	$this$	refers to the current contract
$uint$	now	current block timestamp
$address\ payable$	$tx.origin$	sender of the transaction

Table 3.6: Frequently Used Special Variables and Functions

Function Modifiers. Modifiers in Solidity are analogous to the Object-Oriented Programming decorator patterns. A modifier controls how a function behaves at run-time. In the example provided in Code Snippet 3.7, the modifier $validate(int)$ restricts the execution of the $increment$ function if the parameter value of increment function is less than one.

```
pragma solidity >=0.6.0;

contract ModifierExample {

    int public number;

    modifier validate(int value){
        require(value>0, "value of incrementBy must be greater than 0");
        _;
    }

    function increment(int incrementBy) validate(incrementBy) public{
        number += incrementBy;
    }

}
```

Code Snippet 3.7: Modifier Example

Multiple modifiers can be applied to a function or constructor by specifying them in a whitespace-separated list and are evaluated from left to right. Modifiers are inheritable properties of contracts and may be overridden by derived contracts. The symbol $'\_;'$ in modifier body returns the flow of execution to the original function code. It applies to various contexts, such as providing an easy-to-understand approach to express certain guards, confirming specific conditions after the function execution.

EB2Sol development

In this section, we describe the development architecture for our developed tool EB2Sol, which is an extension of EB2ALL [EB2ALL 2022], developed as a new plugin for generating Solidity smart contracts from Event-B models. Fig. 3.16 depicts the overall architecture of the EB2Sol automatic code generation tool. The given boxes show different steps of the code generation process. The first block is associated with the Event-B model, which serves as an input to the code generation tool.

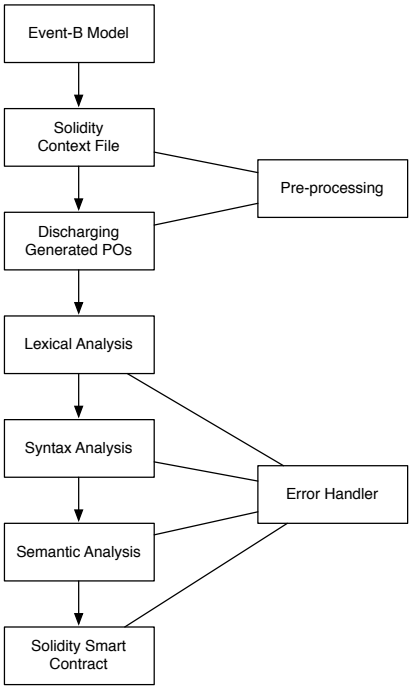

Figure 3.16: Development architecture of EB2Sol

Pre-processing and generated POs. Most of the time, systems fail due to a run-time error. Overflow and underflow of bounded integers, for example, are types of run-time errors that should be checked before producing the Solidity code. There is a pre-processing step in the code generation process that allows for the introduction of a context file based on the Solidity language to provide a deterministic range for data types in order to make the Event-B model deterministic [Sites 1974]. To obtain the deterministic model, we can use vertical refinement to refine the previous concrete model, which provides deterministic definitions of constants and variables. For this, we can introduce a *Solidity context file* which contains bounded integer data types. Table 3.7 shows a map between Event-B and Solidity types for signed and unsigned integers. Note that the Solidity context file is an Event-B context file, which is required to generate correct code. Adding a new context file may generate new set of POs that must be proved before generating the Solidity code as well as to verify the specification in order to ensure the system's consistency.

Event-B type	Formal Range	Solidity type
tl_int8	$-2^7..2^7-1$	int8
tl_int16	$-2^{15}..2^{15}-1$	int16
tl_int24	$-2^{23}..2^{23}-1$	int24
...	...	...
tl_int256	$-2^{255}..2^{255}-1$	int256
tl_uint8	$0..2^8-1$	uint8
tl_uint16	$0..2^{16}-1$	uint16
tl_uint24	$0..2^{24}-1$	uint24
...	...	...
tl_int256	$0..2^{256}-1$	uint256

Table 3.7: Signed and Unsigned Integers

Translation of Event-B to Solidity. The main objective is to translate the Event-B model into a *semantically observationally equivalent* standard Solidity smart contracts. The developed translator is based on a set of transformation rules that map between the Event-B and Solidity constructs. In fact, the transformation functions allow to produce code in target language applying a set of rules when matching the inputs from the source language. We define a set of transformation functions to generate Solidity code from the Event-B model. These transformation rules are given in Table 3.8.

T_{decl}	:	Datatype and Constant declaration translation function
T_{st}	:	State variables translation function
T_{axm}	:	Axioms translation function
T_{pred}	:	Predicate translation function
T_{thm}	:	Theorem translation function
T_{evt}	:	Events translation function
T_{grd}	:	Event's guards translation function
T_{act}	:	Event's actions translation function
T_{exp}	:	Expressions translation function

Table 3.8: Transformation function

A set of supported symbols of EB2Sol tool is given in Table-3.9. This table shows a subset of Event-B syntax to equivalent Solidity smart contracts. All constants defined in a model's context must be replaced with their literal values. We consider Event-B formal notations available at [Abrial 2010a] and capture the Event-B grammar in a Abstract Syntax Tree (AST). This translation tool accepts conditional, arithmetic, and logical formal model expressions. Event-B model that contains machines and contexts are translated to Solidity contracts. Below we provide the translation process for dealing with context and machine models.

Context models. The context of an Event-B model consists of *sets*, *enumerated sets*, *constants*, *arrays* and *functions*, all of which are associated with their respective type. For translation purposes, the translation tool supports all types of context components. The observational equivalence is based on the equivalence of Event-B values and the values of Solidity smart contracts. This equivalence on values is naturally extended to context instances. In Table-3.10, the observational equivalence between Event-B sets and Solidity smart contracts types is given.

Constants, sets and enumerated sets of the Event-B model are translated into constants, type declaration and enumerated sets of the Solidity contracts using the translation function T_{decl} and T_{axm}. An Event-B enumerated sets are semantically equivalent to Solidity contracts enumerated types, thus it is simple to translate.

For the efficiency of the generated code and the correctness of the translation, the link between Event-B and Solidity contracts for integer values have been regarded significant. We recommend to use preprocessing step by introducing Solidity context file. Similarly, the Event-B constant are also directly translatable due to direct typing correspondence between Event-B integer types and Solidity language integer types. For example, if Event-B use a data type given in Table 3.7.

The translation for the declaration of Event-B array type and the Solidity array type is not straightforward. In Event-B, an array can be defined as a total function, whereas in Solidity, they relate to a contiguous memory zone (coded as the beginning address of the array and its size). The semantical correspondence between an array element *arr(i)* in Event-B and the value at the position *arr[i]* in Solidity, can be easily translate.

The translation for the Event-B function into Solidity function is also very complex. Only the Event-B total function is supported by the current prototype tool. Solidity function input and output arguments can be easily identified by looking at the left and right sides of the total function symbol ($\rightarrow$) in the Event-B function specification. The Event-B function definition can be translated in a Solidity function structure.

The context model elements are declared global in the generated code. The type information for context elements is derived from the context axioms used for type definition, such as it can be used to express as integer ranges, specifically supported bit-map types, or arrays of the defined mapping functions.

Event-B	Solidity	Comment
const_x ∈ ℕ ∧ cons_x=120	int256 constant const_x = 120	Constant declaration
x ∈ ℤ	int256 x	Signed integer variable declaration
x ∈ ℕ	uint256 x	Unsigned integer variable declaration
x ∈ tl_int16	int16 x;	Signed integer variable declaration
b ∈ BOOL	bool b;	Boolean variable declaration
x ∈ n..m → ℤ	int [m+1] x;	Array declaration
x = y	if(x==y) { ... }	Conditional statement
x ≠ y	if(x!=y) { ... }	Conditional statement
x < y	if(x<y) { ... }	Conditional statement
x ≤ y	if(x<=y) { ... }	Conditional statement
x > y	if(x>y) { ... }	Conditional statement
x ≥ y	if(x>=y) { ... }	Conditional statement
(x>y) ∧ (x≥z)	if ((x>y) && (x>=z) { ... }	Conditional statement
(x>y) ∨ (x≥z)	if ((x>y) ‖ (x>=z) { ... }	Conditional statement
¬x<y	if(!(x<y)){ ... }	Logical not
X⇒Y	if(!X ‖ Y){ ... }	Logical Implication
X⇔Y	if((!X ‖ Y) && (!Y ‖ X)){ ... }	Logical Equivalence
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 := 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
fun ∈ ℕ × ℕ → ℕ	function fun_name(uint256 $arg1$, uint256 $arg2$) public returns(arg) { ... }	Function definition

Table 3.9: Event-B to Solidity

Machine models. A machine model consists of *variables*, *invariants*, *events*. The Event-B variables can be used to generate Solidity attributes, and the Event-B invariants can be used to extract typing information. All the generated variables or attributes have default property *public*. Note that the Event-B machine may also contain function and array declaration that can also translate similarly to translation rule for function and array given in context model. The required typing information can be extracted from the Event-B invariants.

There are two types of variables in the Event-B specification: global variables and local variables. Global variables are produced directly from the variable declarations, and all of these variables have global scope. Local variables are derived from any clause of an event and are completely local to the corresponding event. All local variable declarations are placed as a list of input arguments of the function when the function structure is generated.

The translation tool uses a recursive method to generate Solidity contracts for each event of the Event-B specification. The translation tool always checks for the *'null'* event (i.e. the guard of a false condition), never generates the source code for such event, and inserts a relevant note into the generated code for traceability. For example, if an event has a single guard with a $false$ condition, the Solidity code does not produce for that event. This automatic reduction occurs to avoid the production of inaccessible run-time code.

The initialization event of Event-B machine is translated as a constructor, and all variables are initialized with default values in the constructor body which are directly derived from the action predicates of the Event-B initialization event.

Event-B types	Solidity language
Enumerated sets	Enumerated types
Basic integer sets	Predefined integer types
Event-B array types	Solidity array type
Function	Solidity function structure

Table 3.10: Equivalence between Event-B and Solidity smart contracts

Guard handling in Event-B is extremely ambiguous due to different meanings, such as local variable type definition, assignment of a value to a local variable, condition statements using negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$), implication ($\Rightarrow$), and equivalence ($\Leftrightarrow$) operators. We build a recursive technique for parsing and identifying different elements of the Event-B guard for translation purposes. For example, an implication ($\Rightarrow$) and equivalence ($\Leftrightarrow$) operator, the translator tool automatically rewrites the predicate in an equivalent form using conjunction ($land$), disjunction ($\vee$), and negation ($\neg$) operators, an equal relation may signify an assignment or equality comparison, and the precise meaning (and thus the resulting translation) deduced from the type and scope of implication ($\Rightarrow$).

Another interesting point to discuss is the definition of a functional-image relation, which can be used to represent a data array or an external function. Once the guards of an event have been classified, the guards that confer local variable type information are utilized to generate variable declarations in the function, while the remaining guards are used to generate local assignment and conditional statements. In addition, local variable type information is derived in the same way as global variables from guard information.

Event-B events are translated into functions, and event parameters are passed as function and modifier arguments. In order to effectively call each function, modifiers are added to each generated function. These associated modifiers are also defined as a function using required/assert statements derived from Event-B guard predicates.

Actions are triggered concurrently in Event-B, and any state modification in the actions is only valid in the whole event post-condition. As a result, dependency checks must be conducted to guarantee that no state variable used as an action assignee has been updated to its post-condition before to usage. As a guard statement, a similar kind of parsing is used on the Event-B action statement. At last, the event actions are directly translated in the form of assignment statements in the function body. Assignments to scalar variables, override statements acting on array-type variables, and arithmetic complicated expressions are all supported through an action translation.

Code generation using EB2Sol

In this section, we will use our developed tool EB2Sol to generate Solidity smart contracts from the formal Event-B model of smart purchase. EB2Sol is an Eclipse-based plug-in for code generation in the Solidity language for the Rodin platform. A screen shot of the EB2Sol in the Rodin environment is shown in Fig. 3.17. After installing this plug-in successfully, the menu *Translator/EB2Sol* and a tool button on the toolbar will appear. To generate Solidity source code for any formal model, a user can select it from the EB2Sol menu or tool button, and a dialog box will appear. This dialog box displays a list of currently active projects. Any project can be chosen by the user to generate Solidity contracts, including a log file containing information about the code generation process.

In our case study, we generate Solidity source code from the proven smart purchase model using the EB2Sol plug-in. Before using this tool, we refine our concrete model by introducing a new context containing Solidity type definitions (see Table 3.7) and removing the abstract operations and non-supported symbols via data refinement. By defining some glueing invariants, this refinement makes the smart purchase model deterministic. All new generated POs must be discharged before generating smart contracts.

Smart contract generation from an Event-B model is straightforward. The EB2Sol tool generates Solidity smart contract files from the concrete model. Constants, type definitions, variables, modifiers, and functions are all included in the generated smart contracts that are extracted from the smart purchase model. The translated constants, type definitions, and variables are taken from the generated code shown in code snippet 3.8.

```
1 enum ProductState {InStock,Booked,Packed,Transite,ReturnReq,
```

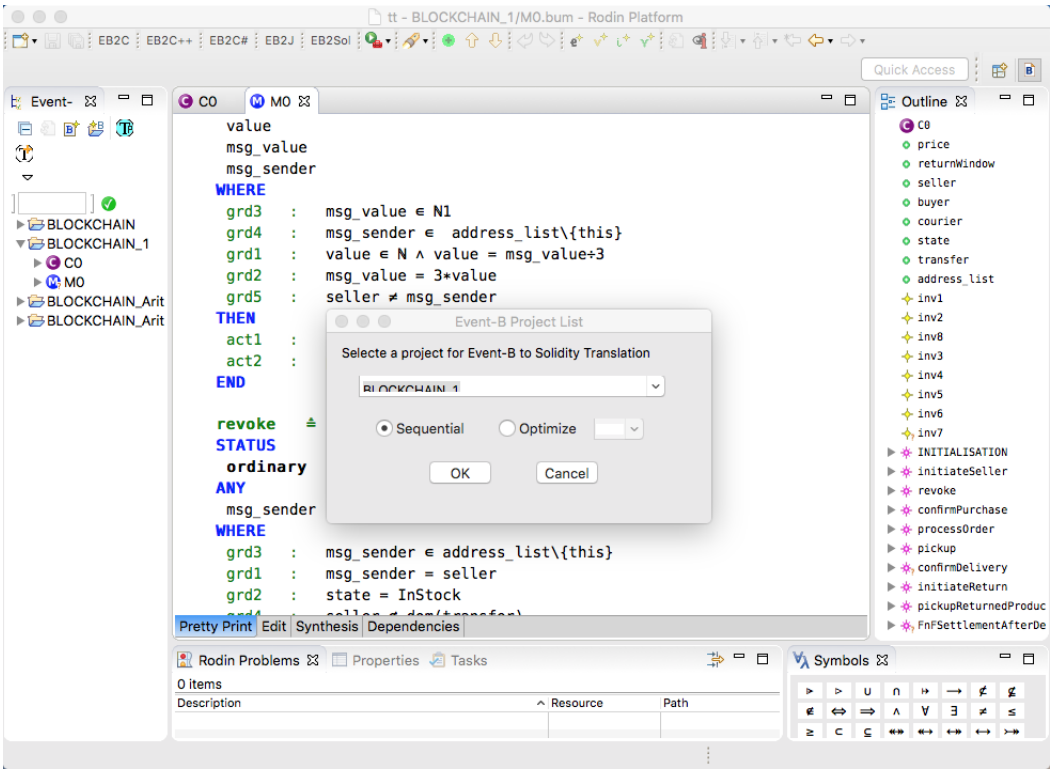

Figure 3.17: EB2Sol plug-in in the Rodin IDE

```
 ReturnAck,Delivered,Closed} // Enumerated  definition

 int256 constant _timeInSec = 10;
//Solidity datatype declaration when constant is given
 int256 constant balance = 500;
//Solidity datatype declaration when constant is given
 ...
 ...
 uint256 price;
//Solidity datatype declaration when variable is given
 uint256 returnWindow;
//Solidity datatype declaration when variable is given
 ...
```

Code Snippet 3.8: Generated type declaration

The formalised smart purchase model yields a set of functions. These functions are generated from events by analysing various elements such as local parameters, guards, and actions. Event-B events are converted into functions, and event parameters are used as function arguments. Modifiers are added to each generated function in order to effectively call it. These associated modifiers are defined as a function as well, using required/assert statements derived from Event-B guard predicates.

The event actions are directly translated equivalent to Solidity assignment expressions. To execute a set of actions in any function, all of the given modifiers must be TRUE. If the given modifiers do not satisfy, the function body statements are skipped. The only excerpt from the generated code equivalent to the given events is given in code snippet 3.9.

```
...
modifier MOD_initiateSeller {
```

```
        uint256 value = msg.value/3;
        require((msg.value==3*value)&&(seller!=msg.sender));
        _;
}
modifier MOD_confirmPurchase{
        require((state==InStock)&&(msg.value==3*price));
        _;
}
...
...
function initiateSeller() MOD_initiateSeller public{
    // Actions
    seller = msg.sender;
    price = msg.value / 3;
    returnWindow = _timeInSec;
}
function confirmPurchase() MOD_confirmPurchase public{
    // Actions
    state = ProductState.Booked;
    buyer = msg.sender;
}
...
```

Code Snippet 3.9: Generated modifiers and functions

Improvement and Extension in EB2ALL

This section describes the previous development and integration of new modelling constructions into EB2ALL.

EB2ALL code refactoring and compatibility with new Rodin IDEs. The first version of EB2ALL was developed in 2011. Since then, the Rodin IDE has greatly improved by adding several new features and making them compatible with new Java releases and other dependent repositories. Because of the new Rodin IDE release, we must also update the EB2ALL plugins to avoid errors and make them compatible with the new Rodin release. As, I am the sole developer of EB2ALL plugins, I refactor the source codes of all plug-ins, compile them, and then test their compatibility with new Rodin releases. This is a time-consuming and iterative process that I engage in on a regular basis. In addition, new errors reported by the Event-B community and active users are also fixed.

Integration of quantifiers ($\exists$ and $\forall$). EB2ALL plug-ins were initially developed without regard for quantifiers. The quantifiers `exist` ($\exists$) and `forall` ($\forall$) are now carefully implemented in the revised version of the EB2ALL plug-ins. For implementing the both `forall` ($\forall$) and `exist`($\exists$) quantifiers, we use for loops (determining the fix number of steps) to check the given predicates. The equivalent code generation from the quantified Event-B predicate is shown in Table 3.11. The provided pseudo-code is applicable to any target programming language. The current version of EB2ALL plug-ins are now capable of generating code for quantified predicates.

Event-B BAP assignment. The first version of EB2ALL generates code for event actions in sequential order, assuming no side effects. It means that no action expression is dependent on any previously computed actions of the same event. However, event actions of Event-B are atomic, thus we modify the core algorithms of code generation for event actions in order to support the atomic action of Event-B. Before handling any action predicates, each event introduces a set of new temporary variables to store all values. All actions are then rewritten to preserve the semantics of atomic action, and all core state variables are finally updated at the end of event actions. This has been successfully implemented in the most recent version of the EB2ALL plug-ins.

Event-B	Pseudo-code for any target language

$\forall i \,.\, i \in 1 \,..\, 100 \Rightarrow fun(i) < 50$	...
	FOR i = 1 to 100
	  IF NOT (fun(i) < 50)
	  return FALSE
	  END IF
	END FOR
	...

$\forall p, k, l . p \in 1 \,..\, 500 \wedge l \in 1 \,..\, 100 \wedge k \in 1 \,..\, 50 \Rightarrow ((p + k) \div l) = 5$	...
	FOR p = 1 to 500
	  FOR k = 1 to 100
	  FOR l = 1 to 50
	  IF NOT ((p + k)/l = 5)
	  return FALSE
	  END IF
	  END FOR
	  END FOR
	END FOR
	...

$\exists i \,.\, i \in 1 \,..\, 100 \wedge fun(i) < 20$	...
	BOOL exist_tf = FALSE
	FOR i = 1 to 100 AND NOT exist_tf
	  IF fun(i) < 20 AND NOT exist_tf
	  exist_tf = TRUE;
	  ELSE IF i = 100 AND NOT exist_tf
	  return FALSE
	  END IF
	END FOR
	...

$\exists i, j . i \in 1 \,..\, 100 \wedge j \in 1 \,..\, 50 \wedge \wedge fun(i) > 10 \wedge fun(i) < fun(j) + 2$	...
	BOOL exist_tf = FALSE
	FOR i = 1 to 100 AND NOT exist_tf
	  FOR j = 1 to 50 AND NOT exist_tf
	  IF fun(i) > 10 AND fun(i) < fun(j) + 2 AND NOT exist_tf
	  exist_tf = TRUE;
	  ELSE IF i = 100 AND j = 50 AND NOT exist_tf
	  return FALSE
	  END IF
	  END FOR
	END FOR
	...

Table 3.11: Quantifiers transformation

Future planned extensions. In the near future, I plan to expand the development of the code generation tool, EB2ALL, to support a wider range of programming languages. As previously illustrated in Fig. 3.15, two planned extensions are currently under development: EB2Ada and EB2Py. Moreover, our goal is support the code generation for hybrid systems developed in Event-B.

Certified code generation. Code generation is an essential component in the design of safety-critical systems. Our long-term goal is to develop an industrial version of EB2ALL that can generate certified code that meets certification standards for industrial applications. In this direction, we will concentrate on generating Frama-C [Cuoq 2012] code from proven Event-B models.

3.5.3 Applications

EB2ALL [86, 88, 91][1] plugins are used in a variety of applications to generate source code in various languages. We produce source code for cardiac pacemaker [1][88], cardiac pacemaker resynchronization therapy [70], insulin infusion pump [25], cruise controller [24], multi-purpose interactive application [22, 24] and so on. Furthermore, these plug-ins are downloaded and used for research as well as evaluation purposes when compared to other code generation tools. A new plugin, EB2Sol, has recently been added to the family of EB2ALL, which has been successfully used to generate smart contracts in Solidity[11]. The use of EB2ALL plugins are elaborated in Chapter 5.

Summary of our contribution for EB2ALL: code generation: In 2011, We developed the EB2ALL [86, 88, 91][1] tool that that includes four main plugins: EB2C, EB2C++, EB2J, and EB2C#. Since 2011, all of these plugins have evolved by incorporating advanced features and more modelling constructs, such as integration of quantifiers, BAP assignment and so on. Several new plug-ins, EB2Sol [11], EB2Py, and EB2Ada, are introduced to generate code in the Solidity, Python, and Ada programming languages, respectively. Note that EB2Sol, EB2Ada and EB2Py are fully developed and integrated into the core of EB2ALL. Furthermore, we intend to add more programming languages to EB2ALL in the near future, as well as develop an industrial version of EB2ALL that can generate certified code that meets certification standards for industrial applications. To demonstrate the usability of EB2ALL, we use several complex case studies from a range of domains.
Project: ANR-EBRP, ANR-FORMEDICIS
Student supervision: Ismail Mendil (PhD, 2019 – 2023), Romain Geniet (MS, 2017), Sasan Vakili (M.A.Sc, 2015)
Publications: [22, 24][70][11][1]
Software: Revised EB2ALL: EB2C, EB2C++, EB2J, EB2C#, EB2Sol, EB2Ada, EB2Py.
Models: `http://singh.perso.enseeiht.fr/eb2all`

CHAPTER 4

Certification and Assurance Case Templates

This chapter covers the work in papers [26, 28][49, 50, 68][16]. This work was done in collaboration with Yamine Aït-Ameur (INPT-ENSEEIHT, France), Morayo Adedjouma (McMaster University, Canada), Valentin Cassano (McMaster University, Canada), Abderrahmane Feliachi (RATP, France), Mark Lawford (McMaster University, Canada), Thomas S. E. Maibaum (McMaster University, Canada), Julien Ordioni (RATP, France), Joannou, Paul (McMaster University, Canada), Hao Wang (Norwegian University of Science and Technology GjÃ¸vik, Norway), and Alan Wassyng (McMaster University, Canada); and with following students: Anas Charafi (Master student at INPT-ENSEEIHT/IRIT, France under my supervision), Mischa Geven (Master student at McMaster University, Canada under under co-supervision of Alan Wassyng, Mark Lawford and myself), Silviya Grigorova (PhD student at McMaster University, Canada, whom I informally advised for the work presented here), Alexandra Halchin (PhD student at INPT-ENSEEIHT/IRIT, France under co-supervision of Yamine Aït-Ameur, Julien Ordioni, Abderrahmane Feliachi and myself), and Nicholas Proscia (Master student at McMaster University, Canada under under co-supervision of Alan Wassyng, Mark Lawford and myself).

This chapter summarises our work on certification and the development of assurance cases for safety-critical cyber-physical systems. Throughout this work, our main focus was to work on developing integrated verification framework that can be used to aid in the certification process. As we all know, a minor flaw in a safety-critical CPS can lead to catastrophic failure and even death. Many people believe that formal methods have the potential to develop safety-critical cyber-physical systems that are also more amenable to certification with required features. The work presented here addresses the problem of integrated verification of system design models for transportation systems, in particular railway systems. It has been achieved in context of the B-PERFect project of RATP (Parisian Public Transport Operator and Maintainer) aiming at applying formal verification using the PERF approach on the integrated safety-critical models of embedded software expressed in a single unifying modelling language: High Level Language (HLL).

In the past few years, the certification bodies introduced a recommended practice asking manufacturers to submit an Assurance Case if they want to market their products. The reasoning behind this was that it would help manufacturers develop safer and more reliable systems, and that the certification bodies would have a better foundation for evaluating these submissions. We have been exploring the use of Assurance Case Templates that can be used to drive development of a software-intensive critical system. Such a template will also provide explicit guidance on an effective assurance case for a specific product within the template's identified product scope. We believe that a product-domain specific template can serve as a standard for development and certification of a safety-critical CPS in that specific product-domain.

In the remainder of the chapter, we sequentially describe our contributions to the development of integrated verification framework and template development for assurance cases.

4.1 Integrated Verification Framework for Certifying Critical Systems

4.1.1 Context

Nowadays, complex critical systems include both hardware and software components that must be developed using high-quality development processes. In fact, when working with critical applications such as transportation, aviation, and medicine, such systems must establish robust testing and confirmation protocols. Moreover, when developing such systems, several stakeholders are involved in a single task or multiple tasks associated with different development

processes. Note that each development process includes a number of development activities and models that are shared and distributed among the other stakeholders, resulting in heterogeneity. In fact, each stakeholder may use a variety of modelling techniques, programming languages, design processes, validation and verification procedures, and so on to deliver the hardware and software components, but prime issue of ensuring the global correctness is the foremost and challenging task.

To address the issue of heterogeneity, RATP developed the PERF (Proof Executed over a Retro Engineered Formal Model) [Benaissa 2016] verification methodology, which allows the evaluation of any software system regardless of its development processes or languages. The approach enables the development and analysis of all product component models in a single shared PERF pivot modelling language with efficient formal verification procedures. The PERF pivot language, HLL [Ordioni 2018], is a synchronous data-flow language, close to Lustre [Halbwachs 1991], allowing to specify both system behaviour and safety properties together in the same model. In fact, the source model must be translated to an HLL model. This translation must be sound and semantically correct. Once model translation is complete, the obtained shared models can be used for (integrated) verification and validation. By using the source code of the developed software as the verification target, it ensures complete language agnostic and non-interference with the software supplier, which drastically reduces any bias. Furthermore, maintaining multiple validation techniques in different domains can be costly, especially when automated assistance is not available.

When the source and target languages have different semantics, one of the most difficult problems is verifying the transformation process. Several researchers have proposed formal verification and certification of translators. Many compiler verification strategies have emerged to reduce the difficulty of the verification processes, such as translation validation, certified compilers, and transformation by proofs. In [Blech 2007], an automated generation of correct translation of the program is defined. Source and target code semantic equivalence is demonstrated using a simulation-based proof. The CompCert compiler [Leroy 2009] is a formally certified translator that uses Coq proof assistant [Bertot 2010] to produce assembly code from the C language. The generated code is extracted from the theorem prover directly. The formal verification of the compiler is provided using LUSTRE in [Bourke 2017, Biernacki 2007]. [Strecker 2002] uses Isabelle/HOL to systematically confirm the transformation of Java program to Java byte code.

In [Besnard 2009], the authors presented a formal approach for translating source of imperative programming languages, such as C and C++, into synchronous language Signal [Gamati 2009]. Model-checking is used in this work to check the required properties. A transformation validation technique capturing the clock semantics in the models is shown in [Ngo 2015]. There, a refinement relation between the source specification and the generated code is proved using SMT solvers. Pop et al. [Pop 2009] presented non-standard denotational specification of the SSA form, including translation from imperative languages to SSA, and vice versa. A similar approach to SSA formalisation is provided in [Blech 2004]. Synchronous versus sequential code validation based on the proof strategy is presented in [Ryabtsev 2009, Pnueli 1999].

Many other approaches [Tatibouët 2003, Storey 1994] concentrate on code generation particularly from B specification to different programming languages like C, Ada and Java. [Bert 2003] shows an optimised transformation from B specifications to C executable code to meet hardware constraints. The general architecture of the transformation process is presented together with optimisation techniques. However, they do not address the formal certification of the transformation. For example, a set of translation rules is presented in [Mammar 2006] to generate Java/SQL code from B models for designing and analysing database systems. A tool B2Jml [Cataño 2012] is developed to generate JML specifications from B models. Bonichon et al. [Bonichon 2015] have developed a tool, *b2llvm*, for generating LLVM executable code from B models. A transformation based on SSA register assignment from a functional to an imperative language is shown in [Schneider 2015]. The transformation is validated via a bi-simulation relation proven in COQ. In [Ning Ge 2017], the authors proposed a set of translation rules for generating HLL models from Event-B models. In fact, the main objective of this work is to use an intermediate HLL representation to produce C code from the Event-B specification. They use equivalence proof to check the correctness of the code generation process. To our knowledge, the proposed translation approach from Event-B to HLL is not automated yet. In [Petit-Doche 2015], the authors reported a posteriori approach for applying formal methods to the developed software by translating SCADE code into HLL code.

As previously stated, RATP employs the PERF framework for verifying and validating various software systems

obtained from different manufacturers. As HLL is the pivot language of the PERF framework for verification and validation, they require a standard framework for dealing with all types of source programming and modelling languages. They have currently developed several transformations to handle various programming languages, but they are not certified. Furthermore, in a number of RATP projects, the B method is used to develop complex software systems using the correct by construction approach, and the developed specifications are manually checked with respect to the given documented informal requirements. This manual process is based on either an analysis of the manufacturer's documentation (who is responsible for delivering the system and software) or a critical reading of code to evaluate the software developed in B. Currently, there is no mechanism for integrating B methods in PERF framework.

Our primary goal is to develop a certified framework for PERF as well as to integrate B methods into PERF.

4.1.2 Our contributions

In this context, we address the problem of integrated verification of system design models in the context of transportation systems, in particular railway systems. It has been achieved in context of the B-PERFect project of RATP (Parisian Public Transport Operator and Maintainer) aiming at applying formal verification using the PERF approach on the integrated safety-critical models of embedded software related to railway domain expressed in a single unifying modelling language, HLL [49]. In particular, we use the B method [Abrial 1996]. It presents a certified translation of B formal models to HLL models [26]. The proposed approach uses Isabelle/HOL as a unified logical framework to describe the formal semantics and to formalise the transformation relation between both modelling languages. The developed Isabelle/HOL [Nipkow 2002] models are proved in order to guarantee the correctness of our translation process. Moreover, we have also used weak-bi-simulation relation to check the correctness of each translation step [50]. We also show that, when models are translated into this unified modelling language, it becomes possible to handle the verification of properties expressed across different models. The main results and our formal framework for certified translators are described below; all details are available in the student thesis [Halchin 2021].

Formal framework

In this study, our goal is twofold: first, we want to certify PERF, and second, we want to integrate B methods into the PERF framework. *A posteriori* and *non-intrusive* verification are achieved in the context of integrated development of safety-critical software by expressing high-level properties in an independent language like HLL. To support this task for B software, we use the B2HLL tool [Charafi 2017], which is a prototyping approach that transforms B into HLL. A crucial part of our approach is demonstrating that the translation in HLL preserves the semantics of B. Fig. 4.1 depicts a formal framework of certified translators that is aligned with other approaches [Blech 2007, Blech 2004, Leroy 2009] in terms of checking semantic preservation and/or semantic equivalence using simulation relationship defining an observational equivalence. This relation compares states of the two models at each execution steps. To our knowledge, there is no work related to verification and certification for heterogeneous system meeting our stated objectives. Our work is the first integrated verification framework for modelling and verifying large complex heterogeneous systems under given requirements formalised as logic properties in a non-intrusive manner. Moreover, our method is transparent compared to the other methods, it does not rely on any specific modelling language. Below, we describe important steps of our framework.

- First, we provide a formal semantics description of the source and target modelling languages in a denotational style using Isabelle/HOL (see upper part of Fig. 4.1). State transitions systems formalise and unify the semantics of the B and HLL models. As new state variables and new transitions emerge in the HLL state transitions system, new properties can be expressed to observe these new variables and transitions.
- An equivalence relation enabling to semantically compare B programs with HLL models is formally defined (see middle part of Fig. 4.1). In our case, two programs are semantically equivalent if their output traces are identical. We establish an equivalence criterion and prove that variables and data flows in B and HLL models have corresponding values during each execution step (transition). An equivalence theorem is stated and proved once and for all.

- B and HLL models are checked to be equivalent. Both B and HLL specific models are defined as instances of the formal semantic models (Instance of relation on Fig. 4.1). Then, the equivalence theorem has to be checked for these two instances by discharging the associated proof obligations successfully. Further, we use the animation approach, including proof steps to validate the formalised models. The associated proofs related to equivalence checking certify that functional representation in modelling languages satisfies the translator specification of the original model.
- Finally, an export tool (lower part of.Fig. 4.1) generates Isabelle/HOL models for the specific input B models and HLL models generated by the B2HLL tool.

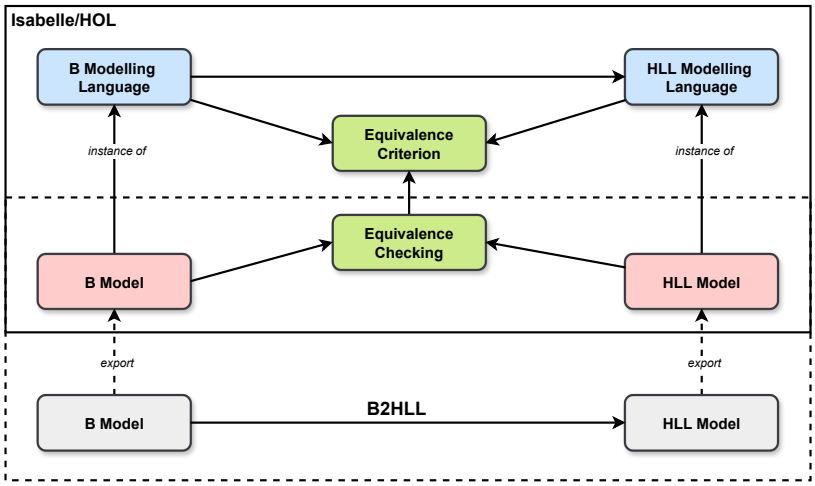

Figure 4.1: A formal framework of certified translator

In summary, the developed framework allows for the definition of a set of sound transformation rules from B to HLL, the guarantee of semantic preservation of the transformation of B models to HLL that can be used for tool certification, and the development of a tool that implements the desired transformation from B to HLL.

Translation of B to HLL

In this section, we describe the general transformation strategy that allows us to obtain an equivalent HLL code from concrete formal models based on B language. The detailed transformation rules can be found in [26][50][Halchin 2021]. The transformation of B models to HLL models is difficult due to the semantic mismatch. The imperative style is used on the B side, while the data flow paradigm with single static assignment form (SSA) is used on the HLL side. In our translation rules, we address variables to data streams transformation, sequential to SSA form transformation, variables tracing, and semantic preserving transformation. We begin by introducing the transformation environment, which will be used to store correspondences (mappings) between B variables and HLL flows, and then we define a set of transformation functions to handle various modelling components. For transformation environment, we define a mapping function, $Mapping : Var_B \rightarrow (Label_{HLL} \times Label_{HLL})$, to maps B variables to pairs of HLL (read and write streams). Moreover, we define a composition operator $(\otimes)$ to compose two environments as $\otimes : Mapping \times Mapping \rightarrow Mapping$.

A set of transformation functions is defined based on a set of transformation rules mapping source language constructs to target ones. We define a general transformation function $T_s(.)$ with a B model and a transformation environment $Mapping$ as input parameters and computes the corresponding HLL model and the updated environment $Mapping$. Specific transformation functions are defined on the syntax tree structure.

Let $model\_B$ and $model\_HLL$ be the syntactic constructs of B and HLL modelling languages, respectively. Then, for each syntactic B construct in $Synt \in model\_B$, we define a transformation function T_{Synt}, that associates

a B construct to its corresponding HLL construct as $T_{Synt} : model\_B \times Mapping \rightarrow model\_HLL \times Mapping$. This function takes into account the previously defined translation environment for variables and streams.

The following notation $T_{Synt}(S_B)_{\mathcal{M}_B} \doteq (S_{HLL}, M_{HLL})$ is used to describe applications of the transformation function T_{Synt} with B and HLL environments $\mathcal{M}_B$ and $\mathcal{M}_{HLL}$. We defined several transformation functions to cover different components of B model, such as states, data definitions, sets, properties, variables, invariants, expressions, predicates, instructions etc. Some of the transformation rules related to static and dynamic clauses are given in Tables 4.1 and 4.2. Complete formal description of each transformation rule is provided in [Halchin 2021].

B Construct	HLL Construct
$T_{cst}(CONSTANTS\ c)_{\mathcal{M}} \doteq$	**Let** $CreateFresh(c, \mathcal{M}) = c_{HLL}$, GetType(c) = type and GetValue (c) = value **in** (Constants: type $c_{HLL} := value;$, $\mathcal{M}[c \mapsto c_{HLL}]$)
$T_{prop}(PROPERTIES\ R)_{\mathcal{M}} \doteq$	**Let** $T_P(R)_{\mathcal{M}} \doteq (R_{HLL}, \mathcal{M})$ **in** (Constraints: $R_{HLL};$, $\mathcal{M}$)
$T_{set}(SETS\ A)_{\mathcal{M}} \doteq$	**Let** $T_S(A)_{\mathcal{M}} \doteq (A_{HLL}, \mathcal{M})$ and GetType(A) = type **in** (Types: type $A_{HLL};$, $\mathcal{M}$)
$T_{var}(CONCRETE\_VARIABLES\ x)_{\mathcal{M}} \doteq$	**Let** $CreateFresh(x, \mathcal{M}) = x_{HLL}$ and GetType(x) = type **in** (Declarations: type $x_{HLL};$, $\mathcal{M}[x \mapsto x_{HLL}]$)
$T_{inv}(INVARIANTS\ I)_{\mathcal{M}} \doteq$	**Let** $T_P(I)_{\mathcal{M}} \doteq (I_{HLL}, \mathcal{M})$ **in** (Proof Obligations: $I_{HLL};$, $\mathcal{M}$)

Table 4.1: Rule: Static Clauses Transformation

B Construct	HLL Construct
$T_{init}(INITIALISATION\ v := E)_{\mathcal{M}} \doteq$	**Let** $T_V(v)_{\mathcal{M}} \doteq (v_{HLL}, \mathcal{M})$ and $T_E(E)_{\mathcal{M}} \doteq (E_{HLL}, \mathcal{M})$ **in** (Definitions: I(v_{HLL}) := $E_{HLL};$, $\mathcal{M}$)
$T_{ops}(OPERATIONS\ x_{out} \leftarrow opName(y_{in}) = S)_{\mathcal{M}} \doteq$	**Let** $T_I(S)_{\mathcal{M}} \doteq (S_{HLL}, \mathcal{M}')$ and $Name_{HLL}(opName, \mathcal{M}) = opName_{HLL}$ **in** (Namespaces: $opName_{HLL}$ { S_{HLL} } , $\mathcal{M}'$)

Table 4.2: Rule: Dynamic Clauses Transformation

Based on this transformation rules, we developed a prototype tool B2HLL [Charafi 2017]. The current version of the B2HLL tool handles the implementation level of B models including imperative programming constructs corresponding to 5000 lines of B code. This approach allowed us to realise a first proven tool before its transfer. Indeed, industrialisation of B2HLL is ongoing at RATP targeting the translation of the entire B language based on our results. In order to automatise the entire verification chain, automatic export tools from B and HLL languages to Isabelle/HOL are under development. We also provide proof of the correctness of the transformation rules implemented by the B2HLL tool.

Certified Model Transformation of B to HLL

This section presents a certified translation from source modelling language to target modelling language, in which we use Isabelle/HOL as a unified logical framework to describe the formal semantics of both languages and formalise the translation relationship between them. The Isabelle/HOL models developed are validated to ensure the correctness of our translation process. We present the weak-bisimulation relation for validating translation steps and the proof process. In our work, we use the B as a source modelling language and HLL as a target modelling language to realise the proposed idea.

Both B and HLL semantics are formalised in a big-step semantics style. Syntactic constructs are defined as datatypes. State changes are recorded using interpretation functions associated to each syntactic construct in a current state. Different types of variables for both modelling languages are implemented using datatype definition. Is-

abelle/HOL data-types modelling features and constructs of B and HLL (states, flows, expressions, modelling statements) are defined. A state, accessed using an environment env function, is defined as a total function that maps variable names to variable values. Primitive types, like integer and boolean are defined as $Tval$. $varname = name \times Tval$ associates a variable to its type (possible values). The val datatype defines values for B and HLL datatypes. The relation between B variables and HLL variables is given by the mapping type.

B Semantics in Isabelle/HOL. For defining B Semantics in Isabelle/HOL, we use a deep embedding approach where B models are manipulated as first class objects. This work rely on the work of [Badeau 2004]. The semantics of B models is described using a semantic function. This function is defined on the structure of the B models. Each syntactic B construct is interpreted by this function. For example, specific data-types for arithmetic expressions $aexp$, boolean expressions $bexp$ and B statements $instruction$ (a block of instructions in sequence, skip, assignment, and conditional) are defined. The semantics of B constructs is defined using primitive recursive functions encoded in Isabelle/HOL. Listings 4.1 shows the formalisation of this semantic function, where B expressions are interpreted by the total function $meaning\_exp \in exp \rightarrow env \rightarrow val$. An expression is evaluated in the environment env. A B expression is interpreted in a given state and denotes a value in val. Two intermediate meaning functions are introduced for arithmetic ($meaning\_a$) and Boolean ($meaning\_b$) expressions defined on their type constructors. Literal values are directly interpreted by their corresponding Isabelle/HOL values. The semantics of binary expressions (+, -, *, =, <, >) is defined from the interpretation of their operands. Similarly, State changes are formalised by the state transition function $meaning\_instruction \in instruction \rightarrow env \rightarrow env$ that produces the next state after execution of a given B statement. It updates the environment env with the effect of the interpreted instruction.

```
fun meaning_exp :: "exp ⇒ env ⇒ val" where
  "meaning_exp (Bexp ex) σ = B o meaning_b ex σ"
| "meaning_exp (Aexp ex) σ = I o meaning_a ex σ"

fun meaning_a :: "aexp ⇒ env ⇒ int" where
  "meaning_a (Value i) _  = i"
| "meaning_a (AVar vname) σ = (case σ vname of (I v) ⇒ v)"
| "meaning_a (Plus aexp1 aexp2) σ = (meaning_a aexp1 σ + meaning_a aexp2 σ)"
| ...
fun meaning_b:: "bexp ⇒ env ⇒ bool" where
  "meaning_b (Value b) _                  = b"
| "meaning_b (Bvar vname) σ               = (case σ vname of (B v) ⇒ v)"
| "meaning_b (Not bexp1) σ                = (¬ meaning_b bexp1 σ)"
| "meaning_b (And bexp1 bexp2) σ          = (meaning_b bexp1 σ ∧ meaning_b bexp2 σ)"
| ...
```

Code Snippet 4.1: Semantics of B expressions

HLL Semantics in Isabelle/HOL. HLL is a declarative and synchronous language with a SSA form. Several formal models of synchronous languages with single state assignment [Schneider 2001, Yang 2013] have been proposed. We rely on the SSA based semantics to define HLL semantics. A HLL model can be defined as a set of order independent flow (stream) assignments. In HLL, each sequence is defined as a total function mapping a natural number to a polymorphic datatype, 'a [Bourke 2017]. HLL variable names are defined as $(name \times Tval) \times nat$. Each variable is uniquely identified. Uniqueness indexing of variables ensures that once translated, no B model identifier is assigned twice in the obtained HLL model (i.e. single assignment property). As for B, where we define the values and state variables as Isabelle/HOL datatypes, we proceed with HLL in the same way by defining HLL flows (streams) as total functions mapping naturals on a polymorphic data-type. HLL stream variables are defined as $vname = (name \times Tval) \times nat$. Each variable is associated to a unique identifier defined by a natural number. Similar to B, specific data-types for arithmetic expressions $aexp$, boolean expressions $bexp$ and statements $instruction$ are defined. Since, HLL conditional is an expression, a particular attention is paid to the flows resulting from conditional expressions. HLL instructions are blocks of assignments. The Isabelle/HOL definitions of these constructs are given in Listings 4.2. Like for B, the HLL semantics is given by semantic functions defined structurally on the corresponding syntactic constructs. The semantic rules for evaluating expressions are defined by interpretation function $meaning\_exp \in$

$exp \rightarrow env \rightarrow val$. As for B, it is defined for arithmetic expressions with $meaning\_a$ and boolean expressions with $meaning\_b$ (see Listing 4.2). The semantics of *if* expression in a state σ produces stream values resulting from the recursive evaluation of branch expression depending on the given condition.

```
fun meaning_a :: "aexp ⇒ env ⇒ int stream" where
 "meaning_a (Value i) _                    = i"
|"meaning_a (AVar vname) σ  = (λi. (case σ vname of (I v) ⇒ v i))"
|"meaning_a (Plus aexp1 aexp2) σ = (λi.meaning_a aexp1 σ i + meaning_a aexp2 σ i)"
...
fun meaning_b :: "bexp ⇒ env ⇒ bool stream" where
  "meaning_b (Value b) _ = b"
| "meaning_b (Bvar vname) σ = (case σ vname of (B v) ⇒ v)"
| "meaning_b (Not bexp1) σ = (λi.¬ meaning_b bexp1 σ i)"
| "meaning_b (And bexp1 bexp2) σ = (λi.meaning_b bexp1 σ i ∧ meaning_b bexp2 σ i)"
| ...
fun meaning_exp :: "exp ⇒ env ⇒ val" where
  "meaning_exp (Bexp ex) σ = B (meaning_b ex σ)"
| "meaning_exp (Aexp ex) σ = I (meaning_a ex σ)"
| "meaning_exp (If c b1 b2) σ = (let (val1, val2) =
   ((meaning_exp b1 σ), (meaning_exp b2 σ)) in (case (val1, val2) of
     ((I b1), (I b2)) ⇒ I (λi.(if meaning_b c σ i then b1 i else b2 i))
   | ((B b1), (B b2)) ⇒ B (λi.(if meaning_b c σ i then b1 i else b2 i))))"
```

Code Snippet 4.2: Semantics of HLL Expressions

Certification of the translation. Once the B and HLL semantics are encoded in Isabelle/HOL, the specification of the B2HLL translation [60] shall be defined in Isabelle/HOL. Semantic preservation by defining an equivalence relationship is defined later. The transformation function from B to HLL is defined on the syntactic constructs identified for both B and HLL. First, we address the mapping of B state variables to HLL flows, and then we provide mapping between B modelling components to HLL modelling components.

At this stage, it is possible to define an equivalence relationship on states and flows. This relation, namely $\cong$ is defined on state variables using an observational relation [Sangiorgi 1998] between states of a B model and corresponding HLL flows obtained after transformation. We prove that a variable and a data-flow stream have the same value by checking that each stream value of the HLL model is equal to the value of corresponding variable of the B model. This definition defines the basic property to check semantic preservation. It relates states to flows. A bi-simulation relationship can be defined to relate B models to HLL models. All the ingredients to write the equivalence theorem are available. This theorem states that the two state transition systems (for B and for HLL) are bi-similar.

```
theorem Equivalence:
1. fixes codeB::"b.instruction" and σ_B::"b.env"
2. and codeHLL::"hll.instruction" and σ_HLL::"hll.env"
3. and n m::mapping
4. assumes * : "(codeHLL, m) = Transformation codeB n"
5. and # : "σ_B ≅_n σ_HLL"
6. and $ : "finite (dom n)"
7. and @ : "well_defined codeB n"
8. and ♣: "well_defined_mapping n"
9. and ~ : "well_defined_state σ_HLL"
 shows
10. "(b.meaning_instruction codeB σ_B) ≅ _m (hll.meaning_instruction codeHLL σ_HLL)"
```

Code Snippet 4.3: Equivalence Theorem

Listing 4.3 describes the global equivalence theorem defining the semantic preservation property. Informally, this theorem states the bi-simulation relation between two state transitions systems. The proof of the equivalence theorem of Listing 4.3 is performed using the Isabelle/HOL theorem prover. The powerful tactics available in this prover allowed us to complete the whole proof of this theorem. Most of the proofs are interactive (semi-automatic), they are completed through user interaction with the theorem prover of Isabelle/HOL. In addition, we perform animation, offered by the Isabelle/HOL models animator, to check and validate the generated HLL models from the B models. We perform model execution combined with formal proof to link the B2HLL tool with the certified

translation rules as defined in Isabelle/HOL. More details on certified model transformation approach can be found in [26][49][Halchin 2021].

4.1.3 Applications

The certified translator has been exemplified by developing simple as well as complex models. We have considered simple examples, such as a room reservation model, simulating pixel movement and so on. Further, our approach is applied to several case studies provided by RATP. We are especially interested in the development of the CBTC (Communications-based train control) system [IEEE-Std-1474.1 1999], specifically the TRPL (Train Reference-Point Localisation) function. This case study is described in [Halchin 2021], including identified unitary requirements and safety requirements. The B model associated to the TRPL is composed of several machines and refinements. All the properties verified in this B model are safety properties to ensure the correct functionality of TRPL function. We have used this model to produce HLL model using our tool B2HLL. All the given properties in B model are successfully expressed in the generated HLL model. Further, we use our developed certification framework to ensure that the transformation of the B model to HLL model is semantic preserving using the defined equivalence theorem. We instantiate the formal specification defined and proven in Isabelle/HOL theorem prover. We show that the state of the obtained HLL model is equivalent to the original B state with respect to the specified model in both B and HLL. The evaluation of expressions is achieved using term substitutions in the proof goal using the Isabelle's simplifier built-in term rewriting engine. At this level, it becomes possible to observe step by step states evolution (i.e. traces) after expanding the corresponding definitions. Moreover, we also use model animation to show that the output HLL model computed by the B2HLL tool is equivalent to the source B model in terms of the defined transformation rules.

Summary of our contribution for integrated framework for certifying critical systems:
Our interest for developing this framework is twofold. First, it offers an independent verification approach to B models that does not rely on B tools and, second, it supports an integrated verification framework with HLL in a single modelling language. In addition, the certification process summarised here asserts the correction of the transformation in terms of semantic preservation. The work presented here addresses the problem of integrated verification of system design models for transportation systems, in particular railway systems. It has been achieved in context of the B-PERFect project of RATP aiming at applying formal verification using the PERF approach on the integrated safety-critical models of embedded software expressed in a single unifying modelling language: HLL. We propose an integrated verification framework that can be used to aid in the certification process. Our work define a formal technique, related to model translation, to verify and to validate the safety critical software developed using the B modelling language. HLL language is used as a basis for safety properties verification in order to bridge the gap between the software specification, such as the formal development in B, and the verification techniques on system level. In addition, our work propose a formal framework to guarantee the correctness of the translation from B models to HLL models. The correctness of the translation rules is proven in Isabelle/HOL theorem prover. A proof of equivalence between B and HLL semantics based on a bi-simulation relationship has been set up. It guarantees that the translation rules implemented in the B2HLL tool are correct i.e. semantic preserving according to the defined equivalence relation. The formalisation and the associated proofs presented in this work can be easily extended to other transformation from state-based language to HLL.
Project: CIFRE-RATP
Student supervision: Alexandra Halchin (PhD, 2016 – 2021), Anas Charafi (MS, 2017)
Publications: [26][49, 50][Halchin 2021, Charafi 2017]
Software: B2HLL, HLL and B semantics in Isabelle/HOL, and models

4.2 Assurance Case Templates

4.2.1 Context

Cyber-Physical Systems are extremely complex systems that combine components with both physical and cyber interfaces and potentially complex interactions between these parts. They are also often both security and safety-critical if the physical system being controlled can harm people. It is imperative that these systems be developed and certified to be safe, secure and reliable – hence the focus on safety-critical cyber-physical Systems. Current safety-critical or high-integrity standards primarily set out objectives on the process, as is typical in much of software engineering. Thus, the acceptance criteria in these standards apply to the development process much more than to the product being manufactured. Having manufacturers use these "good" processes is, indeed, advantageous. However, their use does not guarantee a "good" product, except in a statistical sense. We need to evaluate the quality of the product, not only the process by which it was built [Maibaum 2008].

Regulators, for instance the U.S. Food and Drug Administration (FDA)[1] and the National Highway Traffic Safety Administration (NHTSA)[2], have been dissatisfied with the frequency of the recalls of many of the products evaluated in such a process based regime. Of course, we can make our standards specify the product-focused evidence that is required, as well as acceptance criteria for this evidence. However, software engineering does not have a good track record in this regard. One approach we can take is to identify critical properties of a system that are necessary in order to achieve tolerable risk regarding the safety, security and reliability of that system. Assurance cases have been gaining traction as a way of documenting such claims about these critical properties of a system, together with evidence and associated reasoning as to why the claims are valid. Not surprisingly, the FDA have turned to assurance cases to help improve the quality of products submitted for approval [Keatley 1999]. Assurance cases provide one way of documenting a convincing argument regarding the trustworthiness of the resulting system – built on the identification of specific items of evidence, and the satisfaction of explicit acceptance criteria involving both product and process.

An assurance case [Kelly 1998] provides a structure in which the developer of a product makes a claim regarding critical properties of the product (e.g., safety, security, reliability), and then presents an argument that validates that claim through the decomposition of that claim into sub-claims that are eventually supported by evidence. There are a number of notations and tools for assurance cases, the most popular notation being Goal Structuring Notation (GSN), developed by Tim Kelly [Kelly 1998]. On the one hand, there are many benefits of assurance cases, such as the explicit, detailed documentation of the 'case', to the traceability of evidence to a specific claim facilitated by the structure of the 'case', and a well-structured assurance case can facilitate the identification of gaps between claims, arguments, and evidence in the constructed 'case'. On the other hand, there are some drawbacks, such as the argument linking claims to sub- claims and eventually to evidence is not explicit [28][16], every assurance case is structured differently that is extremely difficult for evaluation [Wassyng 2011], development of assurance cases for regulators after system development [Kelly 2001], and a lack of support for the incremental development of assurance cases [68].

A number of solutions to limiting the variety of assurance cases have been proposed. Two of these are closely related – *safety case patterns* [Kelly 1998], and *assurance case templates* [Wassyng 2011]. Safety case patterns were originally described as "A means of documenting and reusing successful safety argument structures" [Kelly 1998]. (Again, the structure may be present, the argument is not.) The idea here is that an assurance case could be composed primarily of well-known (decomposition) patterns. This is reminiscent of *design patterns* [Gamma 1995], which are widely used in software design. An extension of this idea is an assurance case template, which is an almost complete assurance case structure that can be determined before development starts, in which missing details are provided during development, and some elements may be modified during development.

4.2.2 Our contributions

In this context, we address the problem of developing assurance cases for aiding regulators as well as manufactures for developing and evaluating the assurance cases for certifying safety-critical cyber-physical systems. In addition

[1] https://www.fda.gov/
[2] https://www.nhtsa.gov/

to assurance case researchers exhorting developers to build the assurance case early in the development cycle, our interest in using the assurance case in the way we describe in [28] [16], was inspired by the work of John Knight and colleagues on *Assurance Based Development (ABD)* [Strunk 2008]. Interestingly enough, after working on using assurance case templates and exploring their potential to replace standards, we were informed of a talk given by John Knight [Fitzgerald 2008], in which he outlined the use of an assurance case as an alternative to the avionics standard DO-178B. We do believe our assurance case templates are somewhat different from the *fit for purpose assurance case patterns* used in ABD and other assurance case development. We have been exploring the use of Assurance Case Templates that can be used to drive development of a software-intensive critical system. Such a template will also provide explicit guidance on an effective assurance case for a specific product within the template's identified product scope. We believe that a product-domain specific template can serve as a standard for development and certification of a safety-critical CPS in that specific product-domain.

To use an assurance case template as a standard, we have to develop the template in much the same way we do standards. We need contributions from all stakeholders – industry, academia, and regulatory bodies. Each assurance case template has to reflect the expectations of the domain, and in particular, must result in compliance with the requirements imposed by the appropriate integrity level. It must also result in compliance with ALARP. There are real benefits in having the template developed as a community effort. Templates evolved through community development should benefit from input from a larger group of experts. Members of industry can provide valuable input on practical development processes, members of academia can enhance the templates with the latest ideas and research, and members of regulatory agencies could use the vast data collected from adverse events to further enhance them. Flaws found in a template can be promptly corrected to prevent multiple occurrences of similar mistakes. Every developer, from a multi-national to a small entity, would use the latest approved template. The principal benefits are thus similar to those that apply to community development of standards, namely that communal expertise can be more than the expertise of a chosen few, and that industry is more likely to 'buy-in' to the effort required to comply as well as recognise the benefits of compliance. Specifically for assurance case templates, there is important and much needed consensus to obtain as to what evidence to produce and the acceptance criteria to be used.

Our observation of assurance case patterns is that they are assurance case design artefacts that can be instantiated for a specific situation, and that are used within an overall assurance case. The assurance case template, is a complete assurance case, in which claims are specialised for the specific situation, and the evidence for terminal claims is described together with acceptance criteria for the evidence, and as development progresses the evidence is accumulated and checked against its acceptance criteria. The concepts are, indeed, very similar but these templates are not the same as current patterns. Patterns could prove to be useful within assurance case templates.

Probably the best way of describing an assurance case template is to show an overview example of the process that led us to believe that assurance case templates could be a productive and important step in being able to develop reliable, safe and secure CPS.

To start, let us assume we have successfully developed a product (system) and also documented an effective assurance case for the product. This product is creatively called "Product 1", and the assurance case decomposition structure is shown in Fig. 4.2. In the interests of clarity, assumptions, contexts, strategies, etc., are not included in the figure, and the "1" in the top-level claim box simply indicates that this is the top level-claim and assurance case for Product 1. Now let us further assume that we follow up our success by developing another product, which we call "Product 2". Product 2 is a product different from Product 1, but is related in that it is within the same product domain as Product 1. For instance, perhaps Product 1 is an insulin pump, and Product 2 is an analgesic infusion pump. Or, perhaps Product 1 is a car and Product 2 is a mini-van. We again document an assurance case for Product 2, and we are so expert at developing assurance cases that the new assurance case differs from that for Product 1 only where absolutely necessary. The assurance case for Product 2 is shown in Fig. 4.3.

Fig. 4.3 also highlights the differences between the two assurance cases by explicitly showing which components have been added, removed or modified in the assurance case for Product 2 as compared with the assurance case for Product 1. Now, consider what the figure would look like if we developed Product 2 before Product 1 and then highlighted the differences in the assurance case for Product 1. It should be clear that components added in the case for Product 2 would be shown as components removed in the case for Product 1. In other words, the difference between

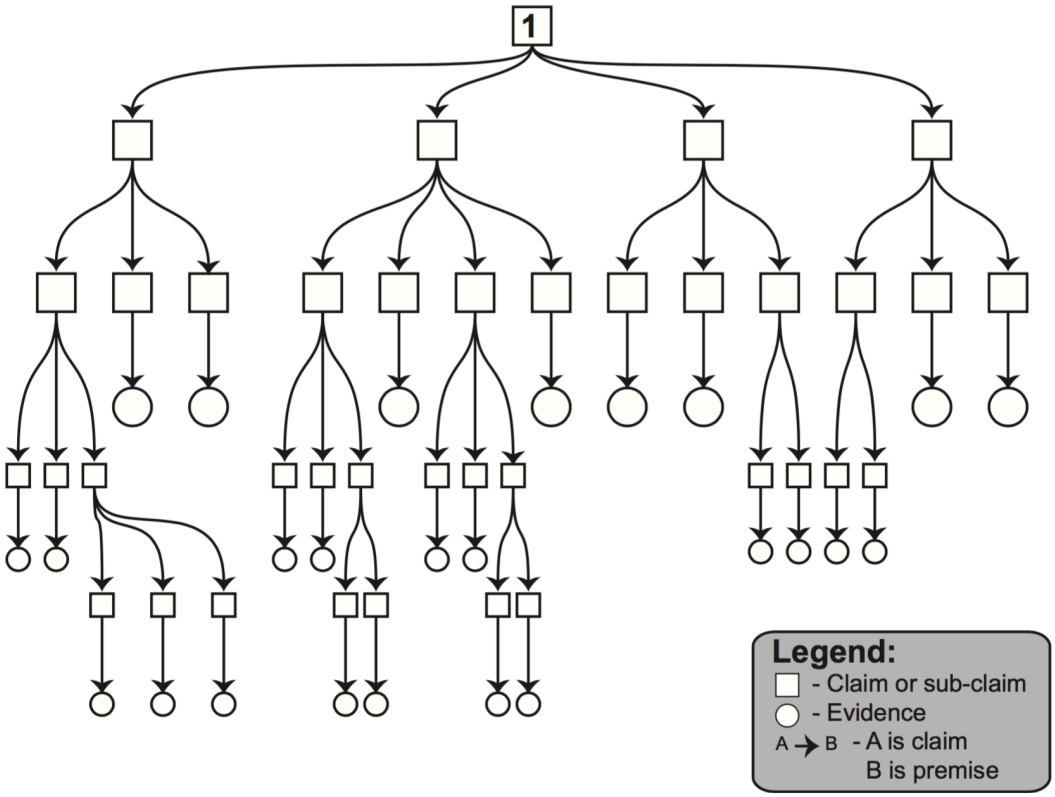

Figure 4.2: Assurance Case Structure for Product 1

“added" and “removed" is one of time – it depends on whether (the assurance case for) Product 1 is developed before or after (the assurance case for) Product 2.

So now we get to the heart of the idea of an assurance case template. We want to develop an assurance case for both Product 1 and Product 2 – before we actually develop the products themselves. Of course, since an assurance case must be product/system specific, we cannot actually build a single assurance case that precisely documents assurance for both products. What we can do is build a template that has a structure that will cater for both products, but in which the content of components will need to be different for the two products. If our template has to handle only the two products, Product 1 and Product 2, then a template that may achieve this is shown in Fig. 4.4. The idea here is that claims and sub-claims can often be parameterised, similarly as is done in assurance case argument patterns [Denney 2013, Yamamoto 2013], so that the claim can reflect the specific product the assurance case deals with. The details are not visible in the figure, but we will discuss this aspect of the template later in this chapter.

Fig. 4.4 now needs some explanation. There are two specific aspects of the template we discuss at this stage.

- *Optional paths:* The paths shown in grey in the figure are optional, depending on the specific product for which the assurance case is being instantiated. The numbers next to the optional paths show the multiplicity of the paths.
  - *Optional 0-1:* This is a single path that may or may not be required for a specific product.
  - *Exclusive-Or 1:* One of the paths (there can be more than 2) must be instantiated for a specific product.
  - *Non-exclusive-Or 1-n:* One or more of the paths can be instantiated for a specific product.
- *Evidence nodes:* The actual evidence for products will differ from product to product. That is why all the evidence nodes in Figure 1.4 are shaded. If this were not true we would not need to develop an assurance case

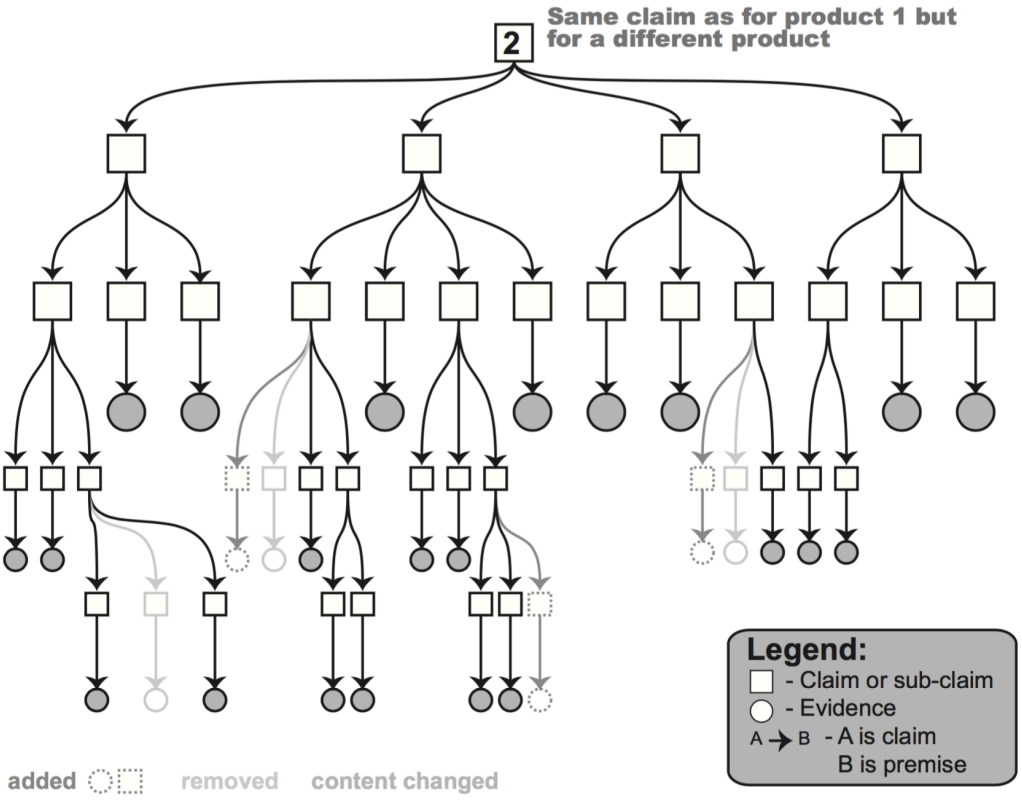

Figure 4.3: Assurance Case Structure for Product 2

at all! Since this is a template, the content of the nodes in the template will not be the actual evidence. What will it be? The answer is simple, and reflects one of the benefits of building an assurance case template. Each evidence node must contain:

- A description of the required evidence.
- Acceptance criteria for that item of evidence, i.e., what must be true of that evidence to raise the level of confidence that the critical properties that the system must have are true.

Characteristics of an Assurance Case Template

We believe the following characteristics are essential for any state of the art assurance case: *explicit assumptions*, *explicit context*, *explicit argument*, *explicit strategies*, and *explicit justification*. Assurance Case Template makes explicit all essential components of assurance cases, allowing us to evaluate the soundness of each component more easily and accurately, allowing us to build high-quality systems. In addition, the Assurance Case Template must meet the following characteristics: (1) *adequate descriptions of evidence to be provided*; (2) *acceptance criteria for evidence*; and (3) *arguments that cope with optional paths*. The template we envisage will be product-domain specific and support for the different development processes. Moreover, it may help us to to design assurance case structures that are robust with respect to change in order to address the problem of incremental development of assurance cases.

Finally, we believe that the developed templates can be used to replace some existing standards because they provide explicit, consistent, and understandable guidance, the opportunity for incremental safety, the ease of developing expertise in evaluating safety, security, and reliability, valid arguments built by experts, avoid confirmation bias, and publicly available examples of good assurance cases.

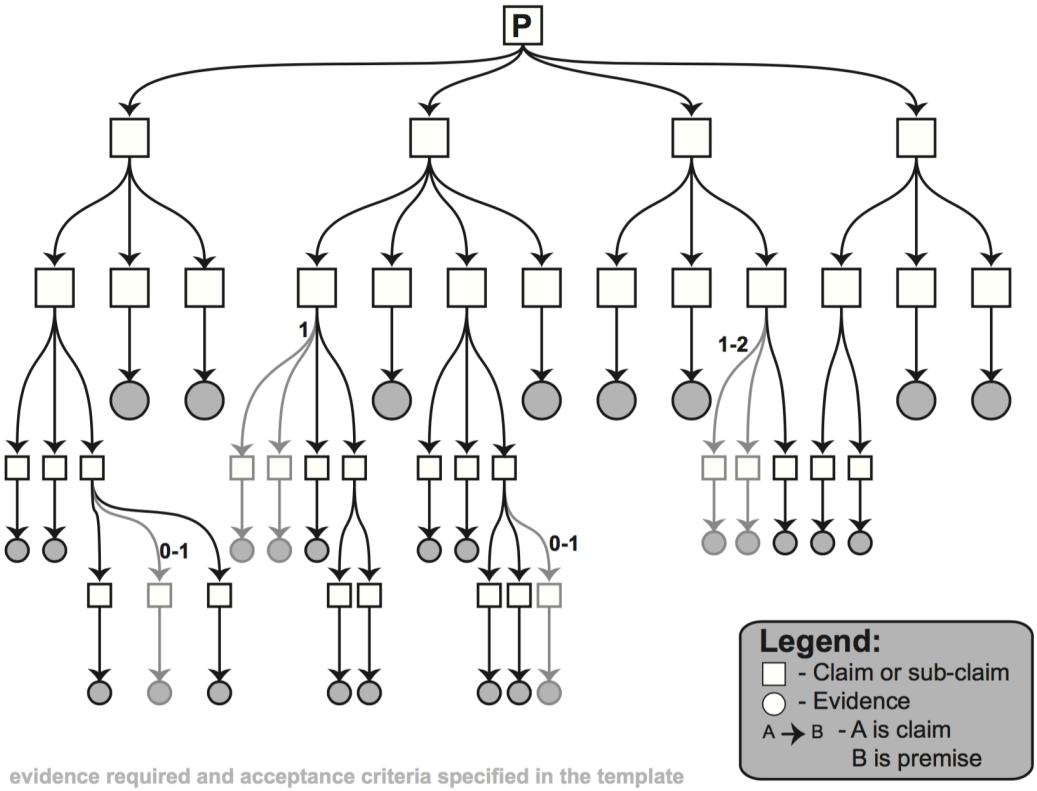

Figure 4.4: Assurance Case Template Structure

An in-depth explanation of the development of assurance case templates, as well as their benefits, is provided in [28][16].

4.2.3 Applications

We used different types of insulin pumps to demonstrate the development of the assurance case templates. In order to illustrate the different kinds of optional paths we foresee for assurance case templates, we have three brief examples drawn from our experience with insulin pumps. Here, we have presented three cases identified for optional paths as follows:

- *Optional 0-1:* Some insulin pumps include a built-in light that helps the user read the pump's screen in the dark. This has impact on the assurance case in a number of ways, including access to data from the pump, and battery usage. These paths in the assurance case would not exist at all if the pump did not have this feature.
- *Exclusive-Or 1:* Some pumps use a standard Luer connector for their infusion sets, while others do not. The pumps definitely need to use some sort of connector for the infusion set, and there are different pros and cons depending on what connector is used. Use of a Luer connector is not mandatory. The assurance case has paths that depend on the connector since it affects both the safety and effectiveness of the delivery of insulin to the patient. However, these paths are different because of the different pros and cons of the connectors. The template therefore will include a number of paths, depending on the number of connectors likely to be used in commercial pumps. Only one of those paths will apply for a particular instantiation of the template.
- *Non-exclusive-Or 1-n:* Some pumps allow you to input glucose readings directly from an associated meter, or to input those readings manually. The assurance case then has to handle the situation where both options are

present in a single pump, as well as the situations where only one of the options is available for a specific pump. Note that this is not a situation where redundancy is being used to increase reliability. The pump that allows both modes of input does so for ease of use – only one mode is used at any one time. The arguments for safe, secure and effective are different in the two modes. An easy difference to note is that there is (probably) less chance of a security problem arising when the manual mode is used.

Summary of our contribution for developing assurance case templates:
We propose using a product/system-domain specific assurance case template as a standard for the development and certification of products/systems within that product domain. Assurance cases have been growing in acceptability as an excellent way of determining the confidence we have that a system fulfils the need for which it was intended, and possess the critical properties it is required to posses. The assurance case template we envisage is the structure of a complete assurance case with (optional) sub-claims and requirements on generated evidence. The template is to be produced, much like a standard, through consensus of experts in the relevant domain. Such an approach promises to deliver significant benefits. Moreover, this work is to be able to cope with the complex, multi-disciplinary, connected nature of all class of critical cyber physical systems. Developing and certifying such systems, so that they are dependably safe, secure and reliable, seems to call out for a significant change in the way we guide people through the development and certification of these systems. We believe that a concerted effort in understanding how to produce effective assurance case templates for this purpose will be one effective step in meeting the challenges of the (near) future. Luckily, they should be just as useful for less complex systems as well.
Project: Certification of Safety Critical Software-Intensive Systems (funded by Ontario Research Fund – Research Excellence (ORF-RE), and IBM), Centre for the Engineering of Complex Software-Intensive Systems, NECSIS (funded by Automotive Partnership Canada (APC)).
Student supervision: Mischa Geven (M.A.Sc., 2014), Nicholas Proscia(M.A.Sc., 2014)
Publications: [28][68][16]
Software: Assurance Case Templates

CHAPTER 5

Deployment of Safety-Critical Cyber-Physical Systems

This chapter covers the work in papers [20, 22–25, 30][36, 41, 42, 44–48, 51–53, 55–58, 61, 65, 67, 69–72, 74, 75, 77, 80, 84][14, 15][8, 9]. This work was done in collaboration with Yamine Aït-Ameur (INPT-ENSEEIHT, France), Alan Burns (University of York, UK), Michael J. Butler (University of Southampton, UK), Ana Cavalcanti (University of York, UK), Arnaud Dieumegard (IRT Saint Exupéry, France), Alexei Iliasov (Newcastle University, UK), Fuyuki Ishikawa (National Institute of Informatics, Japan), Eric Jenn (IRT Saint Exupéry, France), Tsutomu Kobayashi (Japan Science and Technology Agency, Japan), Mark Lawford (McMaster University, Canada), Thomas S. E. Maibaum (McMaster University, Canada), Dominique Méry (University of Lorraine, France), David Navarre (Toulouse III - Paul Sabatier University, France), Philippe Palanque (Toulouse III - Paul Sabatier University, France), Marc Pantel (INPT-ENSEEIHT, France), S. Ramesh (Global GM R&D, Warren, USA), Alexander B. Romanovsky (Newcastle University, UK), Manoranjan Satpathy (Indian Institute of Technology Bhubaneswar, India), Colin F. Snook (University of Southampton, UK), Paulius Stankaitis (Newcastle University, UK), Hao Wang (Norwegian University of Science and Technology Gjøvik, Norway), Alan Wassyng (McMaster University, Canada), and Andy J. Wellings (University of York, UK); and with following students: Guillaume Dupont (PhD student at INPT-ENSEEIHT, France under co-supervision of Yamine Aït-Ameur, Marc Pantel and myself), Romain Geniet (Master student at University of Rennes 1, France, under co-supervision of Yamine Aït-Ameur and myself), Mischa Geven (Master student at McMaster University, Canada under co-supervision of Alan Wassyng, Mark Lawford and myself), Yanjun Jiang (Master student at McMaster University, Canada under co-supervision of Thomas S. E. Maibaum and myself), Ismail Mendil (PhD student at INPT-ENSEEIHT, France under co-supervision of Yamine Aït-Ameur, Dominique Méry, Philippe Palanque and myself), Nicholas Proscia (Master student at McMaster University, Canada under co-supervision of Alan Wassyng, Mark Lawford and myself), Sasan Vakili (Master student at McMaster University, Canada under co-supervision of Mark Lawford and myself).

This chapter summaries various applications pertaining to safety-critical cyber-physical systems. Our main goal is to demonstrate the usability, reliability, maintainability, portability, efficiency, correctness and scalability of our proposed methods, techniques, and tools for domain knowledge engineering, system modelling, and certification and assurance case on a variety of complex safety-critical cyber physical systems. We focus on three types of systems in particular: *Hybrid Systems*, *Interactive Systems* and *Medical Systems*.

Table 5.1 provides a summary of methods and techniques used in the development of safety-critical cyber-physical systems from various domains. The table's first column contains a list of methods and techniques described in various chapters, while the first and second rows are dedicated to the system domain and selected case studies, respectively. If the developed case studies used the listed methods and techniques, the table grids are filled with a 'X,' otherwise the grid is empty.

In the remainder of the chapter, we sequentially describe our contributions to the deployment of Hybrid Systems, Interactive Systems and Medical Systems.

Methods and Techniques	Hybrid Systems			Interactive Systems		Medical Systems	
	Train Controller	ARP	LGS	MPIA	TCAS	IIP	CRT
Implicit and Explicit modelling and Refactoring	X	X	X	X	X		X
Event-B Theories for Handling Domain Knowledge	X			X	X		
Automatic Refinement						X	
Modelling & Designing Frameworks and Pattern	X	X	X	X	X	X	X
Meta-modelling: Reflexive Event-B					X		
Environment Modelling						X	X
Automatic Code Generation-EB2ALL		X	X	X		X	X
Integrated Verification Framework for Certifying Critical Systems	X						
Assurance Case Template						X	X

Table 5.1: A summary of the methods and techniques used in the development of safety-critical cyber-physical systems.

5.1 Hybrid Systems

5.1.1 Context

Today, we are surrounded by various hybrid systems that play a pervasive role in our daily activities. They include, for example, avionics, transportation, nuclear, and medical applications. A hybrid system is a complex dynamic system that exhibits both continuous and discrete behaviour while interacting with the physical environment via sensors and actuators. The continuous behaviour can be represented by differential equation and the discrete behaviour can be represented by a state machine or automaton. The main feature of hybrid systems is that they rely on feedback mechanisms on actual plant behaviour, where sensors, controllers, and actuators play an important role in ensuring proper system functionality. A key challenge in modelling hybrid systems is that physical plants exhibit continuous behaviour, whereas the software controller defines discrete computations. Furthermore, the interleaving nature of hybrid systems' continuous and discrete behaviour is an open problem in case of developing them for safety-critical domains. Verification and validation activities can be carried out to aid in the certification process, ensuring that the developed product is safe for use in the real world.

Formal methods play an important role for designing and checking hybrid system requirements. There are several modelling techniques and tools proposed by the formal methods community to handle different issues related to designing safe hybrid systems. These methods are divided into two categories: *model checking* and *proof-based* approaches. Hybrid automata [Alur 1995, Henzinger 2000] and model checking tools such as HyTech [Henzinger 1997], d/dt [Asarin 2002], PHaVer [Frehse 2008], Flow* [Chen 2013], iSAT [Fränzle 2007], SpaceEx [Frehse 2011], and iSAT-ODE [Eggers 2011] have been used to characterise linear and non-linear differential equations of hybrid systems. However, model checking approaches always suffer from the classical state explosion problem. Proof-based approaches, on the other hand, are based on proof techniques and symbolic verification and can be used to characterise any type of hybrid system. The main proof based approaches for modelling hybrid systems are KeYmaera [Platzer 2008, Quesel 2016], continuous action systems [Back 2000], Event-B [Su 2014b, Butler 2016], Hybrid Event-B [Banach 2013, Banach 2015], HybridCSP [Jifeng 1994, Liu 2010], and Coq [Bertot 2010].

Despite several approaches for dealing with the complex hybrid system design problem, designing safe hybrid systems in a system engineering context remains a major challenge and looking for better solutions. Rigorous verification and validation techniques must be enabled in the engineering process of designing complex and safety-critical hybrid systems. we believe that two major requirements, compositional and incremental design and verification, and validation techniques, must be addressed in order to handle the complexity of such systems. Such requirement refers

to the definition of a reusable framework for the development of hybrid systems. A systematic procedure that allows a designer to produce correct hybrid system designs must be established.

In this context, our proposed the reusable formal framework for designing and verifying hybrid systems [20, 23][41, 44–46, 48, 53, 55, 56] is used for developing complex hybrid systems. The proposed framework is generic and extensible, consisting of method and a set of tools that engineers can apply and use when developing such systems. This framework consists of a large set of theories that extend Event-B with mathematical features required to model continuous behaviours (e.g., differential equations), a generic model that encodes a generic hybrid system, complete with its controller and continuous plant, and a series of patterns based on refinement that allow for easier design. In particular, we define three formal patterns, inspired by general practice in hybrid system designs: approximation, centralised control with multiple plants, and distributed hybrid systems. This work is illustrated in greater detail in Guillaume's thesis [Dupont 2021]. Several case studies related to hybrid systems are developed using our formal framework and modelling patterns. Some of them are summarised below.

5.1.2 Our contributions

This section presents a list of applications developed in the context of hybrid systems, taking into account both the generic framework based on extended Event-B that supports continuous features via *Theory plugin* and the refinement approach in classical Event-B.

Note that we chose three complex hybrid system applications to demonstrate our approach. The first case study is a cyber-physical railway signalling system[53], which demonstrates how we applied our approaches related to implicit and explicit modelling (see Section 2.1), Event-B theories for dealing with domain knowledge (see Section 2.2), continuous behaviour (ODE) modelling and designing generic framework (see Section 3.2), and certification framework (see Section 4.1). The second and third case studies are automatic rover protection [67] and aircraft landing gear systems [74, 75], respectively, to demonstrate our developed approaches related to implicit and explicit modelling (see Section 2.1), modelling and designing framework, and refinement pattern for discrete behaviour (see Section 3.2), and automatic code generation (see Section 3.5).

In addition, several other complex applications, such as stop sign controller [56], signalised left-turn assist [55], water tank[20][46, 48], inverted pendulum [44], plannar robot [45], and stop-and-go adaptive cruise control [15], are also developed using the Event-B and theory plugin extension.

Hybrid train speed controller [53]

Requirements. The signalling system is comprised of trains, communication centres, interlocking boxes and field elements. The former are continuously communicated a safe distance they are allowed to travel, also known as the end of a movement authority (EoA). The speed controller of the train must ensure that at all times the train remains within the movement authority. The other sub-systems of the signalling system must ensure that the communicated EoA guarantees a safe train separation and prevents train derailment by passing over unlocked/moving railway track switches.

The total rolling stock resistance is comprised of the mechanical and air resistances, and commonly expressed as a second-order polynomial (Davis Resistance equation $R_{tot.}(t)$ in Equation 5.1), where A, B, C are fixed parameters and $v(t)$ is the speed of a train at time t [Rochard 2000].

$$\begin{cases} \dot{tv}(t) &= f - (A + B \cdot tv(t) + C \cdot tv(t)^2) \\ \dot{tp}(t) &= tv(t) \end{cases} \tag{5.1}$$

The train speed controller we consider is continuously issued with the end of movement authority (EoA) which is updated discretely in the time by the communication centre. We assume that the speed controller is able to sense its distance to EoA and, in particular, determine if with a given current speed and acceleration it can stop before EoA. The stopping distance calculus is generally done by a complex algorithm on the on-board computer, whereas in our train model, we abstract the algorithm by a stopping distance function ($StopDist$) which takes the current acceleration

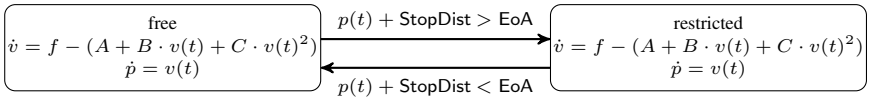

Figure 5.1: Hybrid automaton model of rolling stock speed controller

MACHINE TrainMach **REFINES** Generic
VARIABLES $t, tp, tv, ta, f, \mathsf{EoA}$
INVARIANTS
...
inv4 : $f_{min} \leq f \wedge f \leq f_{max}$
saf_1 : $\forall t^* \cdot t^* \in [0, t] \Rightarrow tp(t^*) \leq \mathsf{EoA}$
saf_2 : $\forall t^* \cdot t^* \in [0, t] \Rightarrow tv(t^*) \geq 0$

Code Snippet 5.1: Excerpt of Train model invariants and safety properties

and speed as parameters, and returns the distance needed by the train to stop, together with necessary assumptions, provided as axioms.

The train speed controller has two modes: free and restricted. If the stopping distance of the train is shorter than the EoA, then the train is said to be in a free mode and it can choose arbitrary values for f. Once the stopping distance of the train becomes shorter than the EoA, the train enters a restricted mode in which it is required to provide values for f such that it can stop before the EoA. The train speed controller hybrid automata model is visualised in Figure 5.1. The railway signalling model is based on radio-based communication and in-cab signalling systems, which typically contain three types of objects: trains, interlocking boxes, and communication centres. The communication centre is the only subsystem that assigns EoA to rolling stock based on information received from trains (e.g., position) and interlocking boxes (e.g point locking and direction status). A centre contains and continuously updates an internal railway network map with junction locations (also their status: free or locked) and rolling stock positions.

The objective of the railway signalling model is ensuring a safe spatial separation of trains and preventing train derailment by guaranteeing that only locked switches are crossed by train.

Formal development. The railway signalling system is developed using the generic framework presented in Chapter 3, and we use Event-B's refinement to instantiate it. We describe the modelling and verification of the railway signalling system Event-B model, and then, discuss simulation of the train speed controller model.

Modelling the railway signalling system starts by formally defining static information. Common properties of the train are gathered in the Train domain Event-B theory. This theory defines the coefficient a, b and c for a traction force of f, with initial condition $p(t_0) = p_0$ and $v(t_0) = v_0$. This equation corresponds to Equation 5.1. In Event-B Context, we define the Davis coefficients (a, b, c) as well as some bounds on the train's traction power (f_{min}, f_{max}), including the minimum traction power for deceleration (f_{dec_min}). Moreover, the train stopping distance function StopDist is introduced as a function of the current speed and acceleration with associated function constraining axioms. Finally, we introduce train controller modes free_move and restricted_move by refining the STATES set with an enumerated set.

The abstract model of the railway signalling system is obtained by refining the generic hybridised Event-B model. Two refinement steps are defined. The first one models the speed controller where the end of the movement authority is regularly updated. At this refinement level, it is left abstract and under specified. Furthermore, we introduce several new events by instantiating generic events to capture the hybrid automata depicted in Figure 5.1. The train is modelled using its position, speed and acceleration (tp, tv and ta respectively), as well as its traction power (f). Additionally, the end of authority is modelled by a real variable, EoA, and the required safety properties are expressed in invariants (saf_1 and saf_2).

Several events are introduced to specify the restricted move and free move of trains. The Transition_restricted_move event models the change in the speed controller by adjusting trains traction effort when the train is in the restricted

EVENT Transition_restricted_move REFINES $Transition$
WHERE
$grd1 : x_s = restricted\_move$
WITH
$s : s = \{restricted\_move\}$
THEN
$act1 : f :| tp(t) + \mathsf{StopDist}(f' \mapsto tv(t)) \leq \mathsf{EoA}$
END

EVENT Actuate_move REFINES $Actuate$
ANY t'
WHERE
$grd1 : tp(t) + \mathsf{StopDist}(ta(t), tv(t)) \leq \mathsf{EoA}$
$grd2 : t < t'$
WITH
THEN
$act1 : ta, tv, tp :\sim_{t \to t'} \langle \dot{tv} = ta = f - (a + btv + ctv^2), \dot{tp} = tv \rangle$
$\& \; tp + \mathsf{StopDist}(ta, tv) \leq \mathsf{EoA} \wedge tv \geq 0$
END

Code Snippet 5.2: Excerpt of Train model

move mode. The event is guarded by a single predicate which enables the event if and only if the status variable $\mathsf{x_s}$ is set to restricted_move. To control train's speed we introduce Variable f denoting the traction force. It is modified by the action such that the stopping distance would not overshoot the end of the movement authority. Then, one must prove an open proof obligation that such traction force value can be found.

The Actuate_move event is the main continuous event of the model. It models the dynamics of the train, using the CBAP operator together with the Davis equation defined in the Train theory. The proposed evolution domain ensures that 1) the train remains before the end of authority, and 2) the train's speed remains positive, in accordance with the system's safety invariants.

The second refinement step extends TrainMach machine by introducing other signalling sub-systems: interlocking, communication centres and field elements and their communication protocol. At this step, we are interested in proving safety of the cyber-physical railway signalling system, more specifically, proving that the issued EoA ensures safe rolling stock separation and prevents derailment. Based on Event-B communication modelling patterns [Stankaitis 2019] new events and variables are introduced to model message channels and capture message exchanges between different sub-systems.

To prove safety properties of the hybrid train speed controller, we strengthen local invariants of the actuation event which in turn allows to automatically prove invariant preservation of invariants $saf_{1,2}$. The resulting $CBAP$ feasibility proof obligations requiring proof of solution existence, are translated to JuliaReach [Bogomolov 2019]. The model is proved by discharging all the generated proof obligations. Corresponding proofs statistics are summarised in Table 5.2, where 55 POs for the speed controller model and 85 POs for the communication model. Most of POs are related to discrete behaviour and the available automated theorem proving tools (e.g. [Iliasov 2016, Déharbe 2014]) are able to discharge them automatically. The introduced gluing invariant linking the first and second refinement models preserves the already proven proof obligations of the first refinement model.

Model	**Total number of POs**	**Automatic Proof**	**Interactive Proof**
Speed Controller Model	55	36(65%)	19(35%)
Communication Model	85	71(83%)	14(17%)
Totall	**140**	**107(76%)**	**33(24%)**

Table 5.2: Proof statistics for the cyber physical railway signalling Event-B model

Train Model Simulation and Validation. We describe the Simulink/Stateflow model translated from the train's hybridised Event-B model. Discrete and continuous parts of the train model are generated in form of Stateflow block and a user defined matlab function block, respectively. The Stateflow model contains two modes: *restricted* and *free*. These modes can be switched between based on various parameters such as end of authority (EoA), stopping distance (SD), engine power (f), position (p) and speed (v). Several Matlab functions are defined within the Stateflow model to calculate EoA, engine power and SD. For calculating SD, we use the equation 5.2, where U is the speed of the

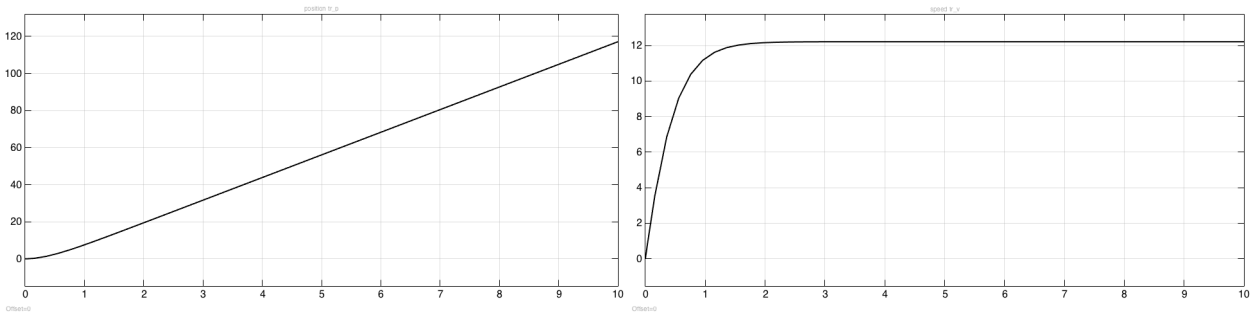

Figure 5.2: TGV train simulation (engine power $f_{max} = 50$, Davis equation coefficients for TGV: $a = 25$, $b = 1.188$ and $c = 0.0703728$, moving authority (MA) $= 10$, time delay t_d= 2 second.

train when the break command was issued; a is the acceleration provided by the braking system; b is the acceleration provided by gravity; and t_d is the train's brake delay time [Barney 2001].

$$SD = -(U + b * t_d)^2/2(a + b) - U * t_d - b * t_d^2/2 \tag{5.2}$$

In each state, we use the *entry* and *during* actions to update the concrete variables. Similarly to Event-B models, the *restricted* mode is chosen as an initial state in the Stateflow model. The dynamic part of the train model is represented by a user defined matlab block in which we encoded the Davis equation 5.1 to calculate the train's acceleration, speed and position. The output of this Simulink block are connected as input to the Stateflow model. We use two scopes to display the train's position and speed. A step block is connected to the Stateflow model as input to define the power engine (f).

The train simulation results show the evolution of the train position and speed in Fig. 5.2. For this simulation, we use the standard coefficients for the Davis equation collected from [Rochard 2000], to simulate the dynamic behaviour of TGV. Moreover, we use a range of values for different parameters to analyse the dynamic behaviour of the train system. We simulate the train model using other standard passenger train coefficients to test scalability and coverage of other classes of trains. The train simulation results ensures the correctness of train dynamic behaviour as well as animation allows to validate the abstract functions of the hybrid train model.

Discussion. This case study demonstrates how the hybrid Event-B framework provides a rigorous and comprehensive formal system development approach by integrating a generic modelling framework, designing and approximation patterns, domain theories for continuous functions, and system simulation techniques. This generic model designs and patterns enable the integration of model features (variables and parameters) by refining its many ways to design different classes of controller-plant hybrid systems (i.e, distributed systems). Furthermore, the development of a simulation model is important for validating the developed hybrid model by running simulations with a wide range of possible values. The proposed methodologies have simplified modelling complexity, proof strategies, integrating domain specific properties in the development of hybrid systems, and may aid in the certification of the respective systems by meeting safety and certification standards.

The instantiation method enables us to create any class of hybrid systems that can always guarantee the core requirements of a hybrid system as well as preserving the required properties. Furthermore, such frameworks encapsulate domain-specific physical properties that can be reused in the future for the development of other systems, and such models only need to be proven once. Moreover, at the generic model level, the proof obligations associated with the systems are realised once and for all, leaving only proofs relating to model instantiation. The generated proof artefacts, and simulation results can be used as evidence in the development of safety cases.

Note that our modelling and designing framework guide novice and expert users to developing complex hybrid systems using in progressive manner. In summary, our frameworks have several advantages, including ease of use, reduced proof efforts, support for the correct by construction approach, encapsulation of domain specific properties,

re-usability of domain theories, integration of continuous and discrete features, and ease of maintenance for developed theories and models,model validation using both proof and simulation, and assist in certification by providing evidences like proofs and simulation results.

Automatic rover protection [67]

Requirements. The Automatic Rover Protection (ARP) is a self-adaptive system that implements a collision avoidance function for *TwlRTee* [67]. TwlRTee is a three-wheel rover developed by IRT Saint-Exupery for the INGEQUIP project, specifically for evaluating various types of hardware and software techniques and tools, such as formal modelling, system design, requirement engineering, simulation and co-simulation, code generation, and hardware integration. ARP is a self-adaptive autonomous system comprised of a collection of small rovers that are supervised by a single supervision station, with each rover acting autonomously by controlling the required speed and brake. Each rover is associated with a priority and is surrounded by three dynamic zones: *warning*, *caution* and *info*(see Fig. 5.3). These areas can change depending on the speed of the rover. All of the rovers must be separated in time and space. If they are not separated, they are in *conflict*, which can be resolved by taking a *stop* or *break* action based on their priority. In an emergency, the supervisor may take over control of any rover.

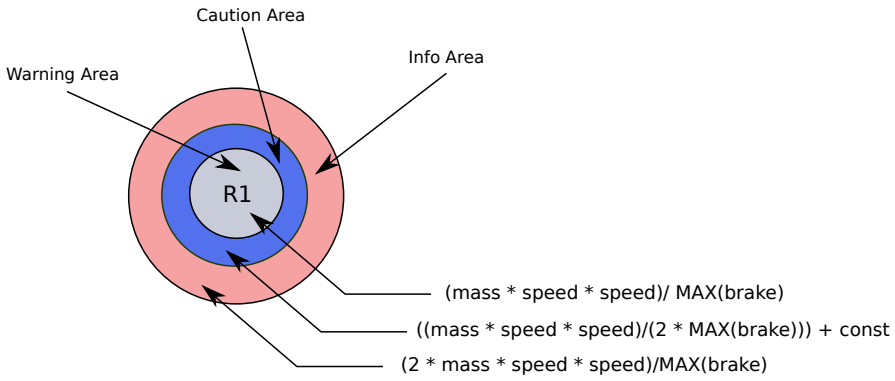

Figure 5.3: Rover zones (warning, caution and info)

The primary goal is to ensure safe rover operations by following a set of protocols in the case of an emergency situation. The unique supervision station supervises regularly each and every tasks of each rover, including position and speed. Moreover, it also maintains a global information of overall system that can be used by any other rovers, and every rover has partial view of the state of the other rovers. In addition, this supervision station can also override any task that can be performed by any rover locally. For example, it can perform stop or brake event on any rover in case of any emergency or any technical difficultly. [67] provides a detailed description of the ARP informal requirements.

Formal development. The ARP self-adaptive system is formalised in classical Event-B using a correct-by-construction approach. We have one abstract model and seven refinements in this development. The first abstract model describes controller operating modes by modelling each rover's possible changing states/modes using mode automata (see 5.4). The first refinement introduces space segregation and rover positioning. This refinement, in particular, abstractly models a set of spatial ranges and positions for each rover, including the necessary safety properties, such as warning area is a subset of caution area, and caution area is a subset of info area. The following four refinement levels are used to model an ARP controller using physical properties (such as mass, brake, and speed), domain modelling, dynamic controller, and a clock. These refinements provide possible dynamic behaviour of the rovers, in which it covers complex calculations of physical equations of required spatial range based on a rover's given speed, mass, and brake in order to stop safely within the given time interval. The final two refinements are used to model rover conflict detection and resolution, in which time domain and space domain separation are used to detect conflicts among the rovers and priority

order is defined for conflict resolution. We incrementally introduce the safety properties in the stepwise refinements to ensure the correctness of the system behaviour.

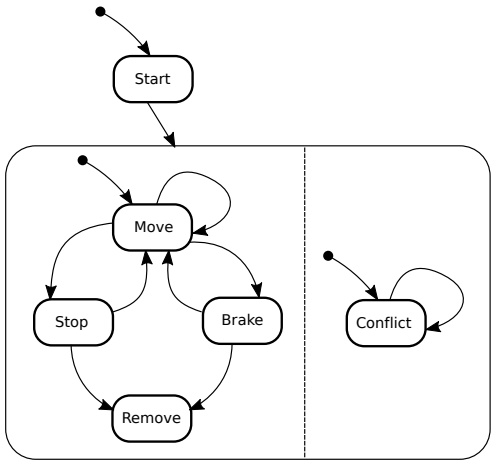

Figure 5.4: Rover operating modes

Rodin tools are used to formalise, verify, and validate the ARP requirements. The development produced 723 (100%) POs, 444 (62%) of which are proved automatically, and the remaining 279 (38%) are proved interactively using the Rodin prover and SMT solvers. These interactive proof obligations are mainly related to refinement and complex mathematical expressions, which are simplified through interaction, providing additional information for assisting the Rodin prover. Some proofs are quite simple that are achieved by simplifying the predicates. Furthermore, we use ProB model checker [Leuschel 2003] to identify the desired behaviour and ensure the deadlock freedom of ARP models in various scenarios. The complete formal development is available for download at[1].

Model	**Total number of POs**	**Automatic Proof**	**Interactive Proof**
Abstract Model	242	153(64%)	89(36%)
First Refinement	108	50(47%)	58(53%)
Second Refinement	70	70(100%)	0(0%)
Third Refinement	111	62(56%)	49(44%)
Fourth Refinement	22	22(100%)	0(0%)
Fifth Refinement	163	82(51%)	81(49%)
Sixth Refinement	2	0(0%)	2(100%)
Seventh Refinement	5	5(100%)	0(0%)
Totall	**723**	**444(62%)**	**279(38%)**

Table 5.3: Proof statistics for the automatic rover protection Event-B model

Discussion. Stepwise refinement is always important in designing a complex and large system systematically and incrementally by gradually providing the system requirements and required safety properties. It is always interesting to practitioners to know the decisions for introducing system behaviours in each new refinement level when applying the refinement steps. In fact, there is no specific rule of thumb or correct pattern to follow during system development. However, when developing a new system, previous experience always helps to make better decisions, improve the quality of the developed models, and reduce manual proof efforts.

[1]http://singh.perso.enseeiht.fr/Conference/ICECCS2016/ARPModels.zip

Initially, we use relation and modes-based approaches to develop ARP models using classical refinements. We discovered that the relation-based approach is very simple for expressing system requirements, but proofs were difficult, and more interactions with proofs were required to discharge the generated proof obligations. On the other hand, the modes-based ARP models approach had sufficient proof automation, and we needed fewer human interactions to prove the generated POs. Note that these manual proof interactions are very simple and can be obtained simply by simplifying predicates. During the development of ARP, we identified the following generic refinement chain: 1) possible controller operating modes, 2) space segregation and position, 3) introduction of physical quantities such as mass and speed, 4) space and time domain separation, 5) introduction of controller, 6) introduction of clock, 7) conflict detection, and 8) conflict resolution. This obtained refinement chain can be used to develop other similar autonomous systems such as autonomous vehicles, swarms, robots, and so on, as well as to guide us to model by introducing required behaviour and safety properties. Moreover, the final concrete model is closed to source code that can be used to generate code for target platforms.

Aircraft landing system [74, 75]

Requirements. A landing gear allows an aircraft to land safely and supports the entire weight of the aircraft during landing and ground operations. The basic engineering and operational behaviour of the landing gear are complex. [FAA 2012]. The landing system controls the maneuvering landing gears and associated doors. Figure 5.5 depicts the architecture of a landing gear system. A landing gear system is made up of three different landing sets that correspond to the front, left, and right of the aircraft. A landing gear system is composed of three major components: 1) mechanical system, 2) digital system, and 3) pilot interface. The mechanical system consists of three landing sets, each of which includes a landing gearbox and a door with latching boxes. The landing gears and doors move with the assistance of cylinders. The cylinder position is used to identify the various door states and landing gear positions. The cylinders are controlled by hydraulic power via electro-valves. A digital system controls these electro-valves (see Fig.5.6).

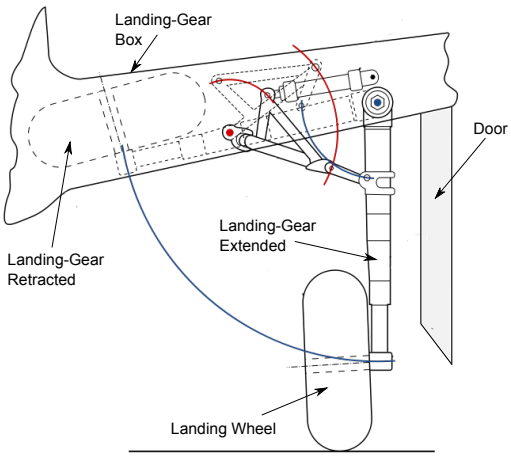

Figure 5.5: Landing Gear System

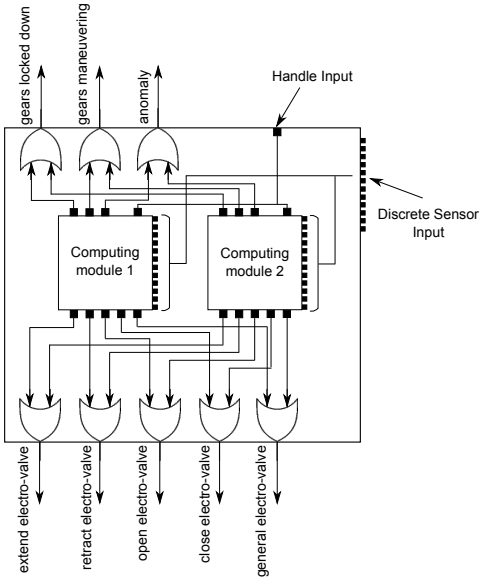

Figure 5.6: Digital Architecture

The digital system is made up of two identical computing modules that run concurrently. The digital system is only in charge of controlling mechanical components such as gears and doors and detecting anomalies. A set of light indicators and an Up/Down handle are included in the pilot interface. The pilot uses the handle to extend and retract the landing gear sequence. A set of light indicators displays gear and door positions, as well as other system states. The landing gear system is described in detail in [FAA 2012, Boniol 2014].

The landing gear is an embedded complex critical system, with each operation dependent on the physical device's state and temporal behaviour. The main challenge is to model the system behaviour of the landing gear and to demonstrate the safety requirements while taking physical behaviour of hydraulic devices into account.

Formal development. The formal development is gradually designing a landing system by analysing core system requirements described in [Boniol 2014]. The aircraft landing system [Boniol 2014] is formally developed in Event-B modelling language using correct-by-construction [74, 75]. The general process starts by an abstract model by capturing elementary behaviours (see Fig. 5.7) of the landing system. The abstract model formalises the up and down movement of the landing system, while the following two refinements describe a general automaton that models the possible dynamic behaviour of doors and gears, respectively. In the following refinement, we model the sensor reading, computing module, and abstract failure detection based on digital architecture (see Fig. 5.6). We introduce sensors to collect sensing values from various components. These sensors are used to detect current activities or states of various components of the landing system, such as the handle, analogical switch, gear extended, gear retracted, gear shock absorber, door open, door closed, and circuit pressurised. In the fifth refinement, we investigate a method for formalising the behaviour of physical mechanical systems such as electro-valves used to control the movement of cylinders. Then, in the following refinement, we integrate the cylinders' behaviour according to the electro-valves circuit and control the system process by computing the system's global state using sensor values. The next refinement simulates the detection of various potential failures or anomalies, such as analogical switch monitoring, pressure sensor monitoring, door motion monitoring, gear motion monitoring, and so on. The timing requirements and health monitoring process of the landing system are introduced in the following refinement by enriching abstract events with timing constraints. The pilot interface is introduced in the final refinement of our development. The pilot has a set of lights that indicate the current positions of the gears and doors, as well as the system's health and the required inputs for these lights are provided by the computing module that monitors the system's health.

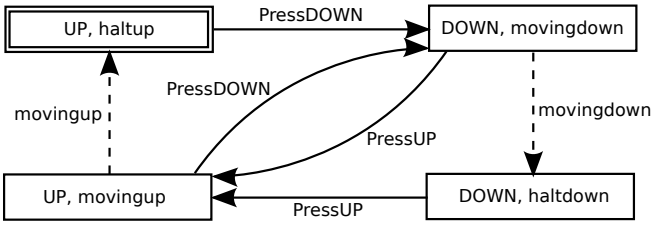

Figure 5.7: Abstract operations of gears and doors

At each refinement level, the required safety properties are introduced. The whole development is done in the core Event-B language, and the Rodin prover is used for verifying and validating the formalised specifications. The complete development of the landing system resulted in 495 (100%) proof obligations, of which 414 (84%) are automatically proven. The Rodin prover is used to prove the remaining 81 (16%) proof obligations interactively. Many proof obligations are generated in the models as a result of the introduction of new functional and temporal behaviours. The majority of proof obligations are automatically discharged, and the remaining proof obligations are discharged interactively by simplifying complex predicates or assisting the Rodin prover by providing additional information. Moreover, the ProB [Leuschel 2003] tool is used for model checking and animating the abstract and refined models under some constraints. These constraints include parameter selection for testing the given models and avoiding state explosion. More detailed formal development is provided in [103].

Discussion. The landing gear system is also developed in classical Event-B using the correct by construction approach, which allows for the progressive design of a system by introducing system requirements and the required safety properties. Our main goal in this development is to formalise and reason about the operational and temporal behaviour of a landing system. We obtained the model of the landing gear system after several iterations. In

Model	Total number of POs	Automatic Proof	Interactive Proof
Abstract Model	10	10(100%)	0(0%)
First Refinement	33	33(100%)	0(0%)
Second Refinement	44	44(100%)	0(0%)
Third Refinement	264	252(96%)	12(4%)
Fourth Refinement	19	19(100%)	0(0%)
Fifth Refinement	49	20(41%)	29(59%)
Sixth Refinement	11	10(90%)	1(10%)
Seventh Refinement	56	23(41%)	33(59%)
Eighth Refinement	9	3(33%)	6(67%)
Total	**495**	**414(84%)**	**81(16%)**

Table 5.4: Proof statistics for the landing gear system Event-B model

our first attempt, we proposed a series of refined models that were closest to the description of requirement documents [Boniol 2014]. In this development, the proof was difficult. Furthermore, we attempted to provide a global view of the system and a very abstract model, but we soon discovered an error in further refinements of not accounting for counter orders. We revised our abstract model based on abstract automation to capture the essential behaviours, and then we refined the abstract model to include the systematic operations or counter orders of door movement, such as locking and unlocking operations. We introduced concrete elements such as gears, sensors, computing modules, and other necessary components based on this model. The timing requirements were added in the seventh refinement, which was then equipped with lights as the pilot interface in the final refinement. The developed model can be refined further to reach an implementation level.

The approach is concerned with separation: first, it proves the basic behaviour of the landing system at an abstract level, and then it introduces the peculiarity of the specific properties. We demonstrated in the beginning that the fundamental properties, namely retraction, extension, doors opened and closed, and solution uniqueness, were maintained during the refinement process (provided, of course, the required proofs are done). Later refinements address the system's complex behaviours in order to obtain a final implementable concrete model. However, the developed model is complex, and the domain-specific knowledge is fully modelled within the system model. We may create a domain-specific theory to simplify the modelling and proof process. Furthermore, certain physical properties are not encoded in the model that can be introduced in theory.

5.2 Interactive Systems

5.2.1 Context

The Interactive Systems (IS) are the primary interfaces through which users interact with complex critical systems. Today's critical systems employ highly sophisticated interactive systems made of both hardware and software components. To ensure the safety of operations, the intended behaviour of these interactive systems must comply with usability, dependability, and security. In such systems, any component failure or operator error may lead to catastrophic consequences. Developing an interactive application was known to be a difficult and time-consuming task in the 90s [Myers 1993] and this has become even more complex due to complex system characteristics and user requirements capturing key aspects of human behaviour, system components and operating environment. Beyond that, the complexity lies in gathering data about operators such as requirements, needs, and tasks [Maiden 1993], particularly because different application domains necessitate different methods and techniques [Sutcliffe 2020]. Finally, the development process of interactive systems differs from traditional software development processes [Shneiderman 2016], and necessitates dedicated phases such as task analysis and modelling, usability evaluation, or training program design and construction [Martinie 2012].

Since a long time, formal methods play an important role for the modelling and analysis of interactive systems [Harrison 1990, Palanque 1997, Campos 2008, Bolton 2011, Aït-Ameur 1998b], and it is widely accepted that they are the only way to integrate interactive systems in critical systems [Gray 1999]. Moreover, formal techniques are being introduced as part of the standard development process for designing interfaces, such as Ansys SCADE Display[2]. In particular, formal methods have been used to check functional requirements and safety requirements by developing models for interactive systems [Harrison 2017]. As interactive systems become more complex, their formal description faces scalability issues [Massink 2012].

Several techniques for modelling, designing, verifying, and implementing interactive systems have been developed in recent years. Many methods have been successfully used, such as Petri net [Palanque 1996], process algebra [Eijk 1989] and model checking [Abowd 1995], to test HMI's intended behaviour. Petri nets and finite automata are used to describe and analyse mouse behaviour related to clicking and dragging actions [Henry 1990]. Palanque et al. [Palanque 1994, Palanque 1996] proposed the development and implementation of HMI using the formalism of Interactive Cooperative Objects (ICO) and PetShop [Palanque 2009]. In [Navarre 2001b], the authors also proposed a framework for analysing interactive systems by combining user task models and system models, to test whether the system model supports a user task. In [Campos 2008], the authors proposed a framework that supports a model checking approach to analyse a set of generic properties for interactive systems that can aid in visualisation and implementation of usability requirements. Bolton et al. [Bolton 2011, Bolton 2009] proposed a method for evaluating human errors and system failures using formal techniques by integrating task models and erroneous human behaviour. LIDL (LIDL Interaction Definition Language) is a modelling language proposed in [Lecrubier 2016] to describe interactions of an interactive system by defining the static nature of the HMI with interfaces and the dynamic nature of the HMI with interactions. In [Ge 2017], the authors proposed a formal development process for designing interactive systems in the LIDL modelling language, followed by formal verification of the required interaction behaviour using the S3 solver [Breton 2016]. The CHI+MED [Curzon 2014] project proposed a set of integrated model-based engineering methods that support formal and semi-formal techniques to aid in the certification process for critical medical devices. In [Harrison 2014], the authors presented an approach for modelling medical interactive systems in Modal Action Logic (MAL) for specifying interaction behaviour, and the PVS theorem prover [Owre 1992] was used to verify the modelled system. In [Harrison 2014], the authors presented a methodology for designing a user interface that complies with use-related safety requirements using formal methods. To check the required properties of executable model of interactive software in the *Djnn* framework presented in [Chatty 2015], interactive components are described hierarchically, with descriptions of low levels such as graphics, behaviours, computations and data manipulations to exploit HMI low-level properties.

In formal modelling, refinement plays an important role for handling this complexity by developing models incrementally, in which each incremental step can be used to introduce new functionalities while preserving the required safety properties [Abrial 2007] within and between incremental models. An incremental development of an interactive system using B methods presented in [Aït-Ameur 1998b, Aït-Ameur 1998a] to describe interaction requirements and the properties of reachability, observability and reliability. In [Aït-Ameur 2000, Aït-Ameur 2006], a development lifecycle was proposed to produce source code from a formalised model of an interactive system for implementation purpose.

All of the approaches discussed above face a number of challenges, including a lack of abstraction or formal design patterns for dealing with various aspects of interactive systems, validation of possible nominal and non-nominal scenarios, expressing domain specific required properties, and domain-specific information related to widgets. Nevertheless, the main contribution of the aforementioned research and studies is to address specific problems of an interactive system, such as interaction management, task analysis, various types of HMI properties, and so on. To our knowledge there is no work related to modelling, refinement, designing patterns, simulation, domain knowledge integration and management, scenarios, task analysis together for developing interactive systems.

In this context, our proposed framework, such as F3FLUID (Formal Framework For FLUID) and structuring of Event-B models using *model-view-controller* (MVC), is used for designing, verifying as well as tackling the core challenges of interactive systems [22, 24][42, 47, 51, 58]. The proposed framework is generic and extensible, consisting

[2]https://www.ansys.com

of method and a set of tools that engineers can apply and use when developing interactive systems. This framework consists of theories that extend Event-B with modelling features required to model domain specific behaviour, ARINC 661 standard conformance, widget properties for an interactive system, and modelling patterns based on refinement that allow for easier design as well as structuring of Event-B models. Several case studies related to interactive systems are developed using our formal framework and modelling patterns. Some of them are summarised below.

5.2.2 Our contributions

This section presents a list of applications developed in the context of interactive systems, taking into account both the F3FLUID framework, and MVC structuring for Event-B models, including standard conformance and Event-B theory extensions.

Note that we chose two complex interactive system applications to demonstrate our approach. The first case study is Multi-Purpose Interactive Applications (MPIA) [22, 24], which demonstrates how we used our approaches related to implicit and explicit modelling (see Section 2.1), Event-B theories for dealing with domain knowledge and standard conformance (see Section 2.2), MVC structuring and F3FLUID modelling and designing generic framework (see Section 3.2), and automatic code generation (see Section 3.5). The second case study is Traffic Collision Avoidance System (TCAS) [19] to demonstrate the use of our proposed approaches related to implicit and explicit modelling (see Section 2.1), Event-B theories for dealing with domain knowledge and standard conformance (see Section 2.2), F3FLUID modelling and designing generic framework (see Section 3.2), and meta-modelling reflexive framework (see Section 3.3).

Multi-Purpose Interactive Applications (MPIA) [22][51]

Requirements. The Multi-Purpose Interactive Applications (MPIA) [51] an industrial case study issued from aircraft cockpit design, complying with the ARINC 661 standard [ARINC 661 specification 2002]. Fig. 5.8 depicts a real User Application (UA) of the MPIA interactive system that handles many parameters of the flight. This system provides a tabbed panel with three buttons, WXR for controlling Weather Radar information, GCAS for Ground Collision Avoidance System parameters and AIRCOND for handling air conditioning settings. The crew leader can turn to either mode (see Fig. 5.8) using tabs. These tabs have three separate programs that can be operated by the pilot and the co-pilot utilising any input system. The MPIA window in each tab consists of three major parts: *information area*, *workspace area* and *menu bar*. The information section is the top bar of every tab that splits into two sections to show the current status of the task on the left hand side and the error notices, activities in progress or incorrect modification as appropriate on the right. The workspace area depicts adjustments to the chosen virtual control panel. For example, the WXR workspace displays all the modifiable parameters of the weather radar system, the GCAS workspace displays some of the operating modes of the GCAS, and the AIRCOND workspace displays the selected temperature within the aircraft. The menu bar section includes three sections for accessing the WXR, GCAS and AIRCOND digital control panels.

Formal development. The MPIA system is developed using the F3FLUID framework as well as the MVC structuring pattern presented in Chapter 3. We describe the modelling and verification of MPIA models in FLUID and Event-B, and then, discuss interactive simulation.

This development demonstrates the effectiveness of our proposed framework to the development of such interactive systems. Consequently, we elaborate system requirements of MPIA in natural language, which are used to develop an initial FLUID model of MPIA that consist of functional behaviours, states, assumptions, expectations, interactions, properties and scenarios. In[43], we present the formalisation of standard concepts and rules as an ontology, as well as the formalisation of an engineering domain, using an Event-B theory consisting of data types and a collection of operators and properties. The ARINC 661 standard is formalised as an Event-B theory and the conformance checking is accomplished by annotating the system model with typing conditions. Moreover, this development involves the development of the FLUID model and its Event-B model generation using refinement for the formalisation and

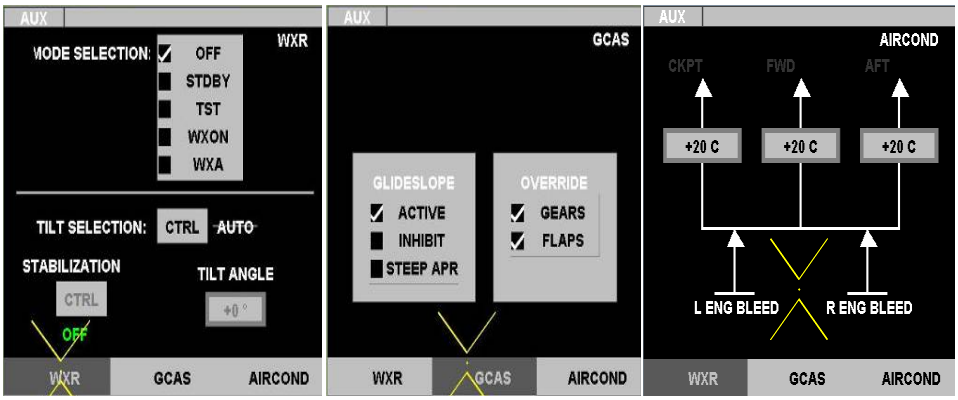

Figure 5.8: Snapshots of the MPIA (from left to right: WXR, GCAS and AIRCOND)

reasoning of the interaction behaviour of MPIA; interactive graphical simulation of the MPIA model in ICO; and validation of the model using ProB using the verified and proven formal MPIA model. We have used the proposed formal framework and associated tools to model and design of MPIA using several refinement layers. Each refined model has been proven to ensure that the required safety properties are preserved. The initial model captured the basic behaviour of MPIA and subsequent refinements were used to formalise the concrete behaviour of the resulting MPIA, which includes a detailed design of interactive systems. The generated Event-B model uses the Rodin tool [Abrial 2010b] to check the syntactical correctness and consistency of the modelled MPIA with respect to its safety properties. Several proof obligations (POs) are generated and all of them have been proved to establish the correctness of the Event-B MPIA model. Here, we summarise the POs of WXR model provided by the Rodin prover. The complete formal specification of MPIA contains 1 constant, 6 enumerated types, 7 axioms, 9 variables, 21 invariants and 9 events for specifying the system requirements. The stepwise development results in 67(100%) proof obligations, in which all the generated POs are proved automatically using the Rodin provers, such as SMT solvers and standard B prover (see Table 5.5). Note that the progressive development using MVC increases the proof automation.

Model	**Total number of POs**	**Automatic Proof**	**Interactive Proof**
Abstract Model	5	5(100%)	0(0%)
First Refinement	11	11(100%)	0(0%)
Second Refinement	21	21(100%)	0(0%)
Third Refinement	8	8(100%)	0(0%)
Fourth Refinement	22	22(100%)	0(0%)
Total	**67**	**67(100%)**	**0(0%)**

Table 5.5: Proof Statistics of MPIA

Model validation and model animation are carried out using the ProB [Leuschel 2003] model checker. ProB supports *automatic consistency checks*, *constraint-based checks* and can also detect potential deadlocks. Note that the Event-B model produced is directly used in ProB. In this study, we use ProB to show the absence of errors (there is no counterexample) and the absence of deadlock. We describe the properties of FLUID model defined in LTL formulas to check the correctness of interaction of the produced model. Model animation allows to investigate the traces of the Event-B models with respect to the given properties. We also use ProB for animating the models to validate the built MPIA model. This validation approach means gaining confidence that the models built are consistent with the requirements. The ProB animation helps to identify the desired behaviour of the model in different scenarios.

Animation with PetShop. Each Event-B model is used for producing ICO models for task analysis, animation and validation of expected visual properties for each defined widget. The concrete model of Event-B and ICO model can be used for implementing a desired user interface satisfying the required properties. Fig. 5.9 presents the PetShop execution of the MPIA models. Note that, the Petri nets models can be modified at runtime and these modifications are directly rendered on the associated user interface. This support prototyping activities when the correctness of models is validated with stakeholders. Detection of models not behaving as expected can result in immediate modification that can be checked right away.

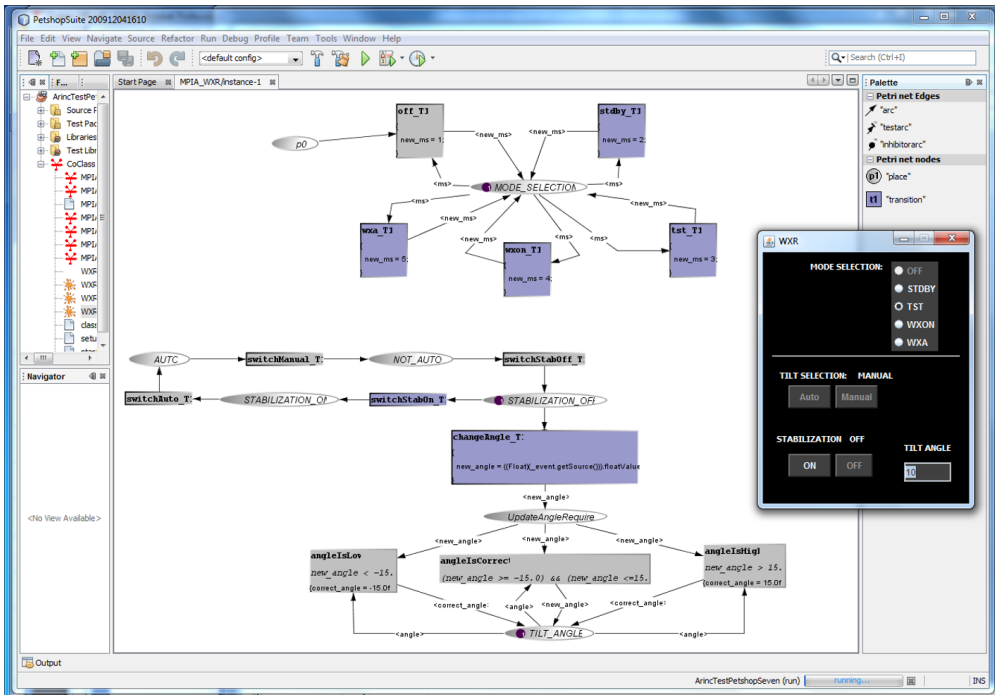

Figure 5.9: Screenshot of the Petshop main user interface

A complete formal development of the MPIA case study can be found on website[3], as well as the MPIA ICO models are available on [4].

Implementation. We also use EB2J [5] tool, a code generation plugin for EB2ALL [6] [88][1], to generate Java code from the Event-B model. EB2J generates Java files corresponding to the concrete models of interactive system. The generated constants and variables are extracted from the context and machines sections of the Event-B model, respectively. Similarly, the datatype for each variable is also extracted from the invariant section of the model. All the events of the formalised model are translated into equivalent Java functions, in which guards are represented as a nested *'if'* structure in a separate *'if'* statement, and action predicate of an event is translatable into Java assignment expression. A detailed description about code generation is given in [88][1], and these code generation plugins are improved by adding new functionalities, new operators, and support for a new programming language.

Discussions. The outcomes of MPIA development using the proposed F3FLUID framework and MVC structuring patterns can aid the HMI community in the progressive development of interactive systems. Our proposed framework

[3] http://singh.perso.enseeiht.fr/Journal/JSEP2021/MPIA_Models.zip
[4] https://sites.google.com/view/mpia-application-ico-models/home
[5] http://singh.perso.enseeiht.fr/eb2all/eb2j.html
[6] http://singh.perso.enseeiht.fr/eb2all

has the potential for improving system safety and quality, including to aid in the process of system certification for interactive systems. F3FLUID is the first integrated formalised framework for formal development of interactive systems. The prime benefits of these approach are to improve the process for specifying interactive system requirements abstractly, separation of concerns, integrating HMI specific domain knowledge and domain theories, formal development using correct by construction, improving error detection through rigorous analysis and interactive simulation, certifying interaction behaviour, and user interface implementation. Applying iterative process for developing an interactive system from specification to user interface implementation allows us to carry out rigorous analyses that can verify the important properties of consistency, deadlock-freedom, interaction behaviour, nominal and non-nominal scenarios, and domain-specific properties. Furthermore, the proof obligations associated with the theories are realised once and for all, which may reduce overall proof effort. Moreover, the generated proof artefacts and simulation results may aid in gaining confidence, which may aid in the certification process.

In order to design a complex interactive application, it is always essential to identify a good abstraction and a sequence of refinements. There may be no 'correct' pattern to follow universally. However, we identified the development order for designing complex interactive applications using MVC patterns. The identified MVC patterns for structuring Event- B models allows us to systematically build a complex interactive application by analysing interaction behaviour. Changing orders of MVC components in the identified progressive development pattern is not suitable for developing complex interactive systems. It leads to complex modelling structure and complex proofs. Moreover, it allows easy maintenance and re-usability of the developed models.

Traffic Collision Avoidance System (TCAS) [19][42]

Requirements. The Traffic Alert and Collision Avoidance System (TCAS) is an airborne avionics system that operates independently of ground-based air traffic control (ATC) to reduce the risk of mid-air collisions. It monitors aircraft in the surrounding airspace, using position data from their transponders to detect potential collisions. TCAS issues a Resolution Advisory (RA) to the flight crews of affected aircrafts when an impedent collision is detected. These advisories instruct them to climb or descend at a specific vertical rate to avoid collisions [EUROCAE 2013, EUROCONTROL 2017].

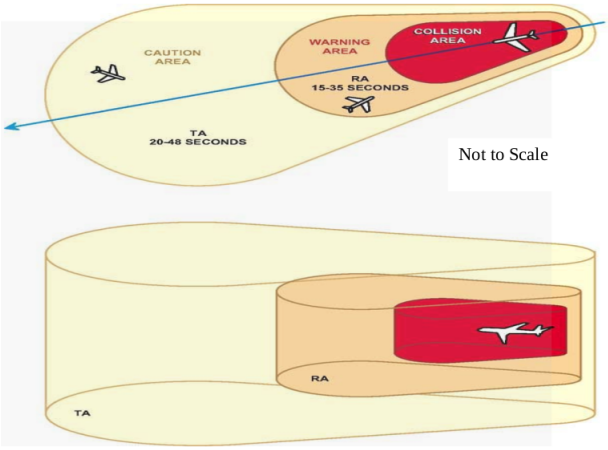

Figure 5.10: TCAS protected volume

TCAS creates a virtual protected volume (Fig. 5.10) that includes the positions of nearby aircraft. This volume is subdivided into three layers: Caution Area (CA), Warning Area (WA), and Collision Area (CO). The volume of the aircraft is affected by its speed and trajectory. Some volume-related information is displayed on a cockpit screen for use by the flight crew. Any critical aircraft detected in CA, WA, or CO must be displayed on the screen, while non-critical aircraft cannot. If the aircraft is within range, it must display on the grid; otherwise, it may display on the screen edge. There are some important safety requirements, such as: every detected aircraft must be either within or

THEORY $DisplayabilityTheory$
IMPORT $OntologiesTheory$
AXIOMATIC DEFINITIONS $IOOntologyAxiomatisation$:
TYPES $IOClasses, IOProperties, IOInstances$
OPERATORS
isIOOntologyWD $< predicate >$
$(o : Ontology(IOClasses, IOProperties, IOInstances))$
visible $< expression >: IOInstances$
hidden $< expression >: IOInstances$
critical $< expression >: IOInstances$
safe $< expression >: IOInstances$
hasVisibility $< expression >: IOProperties$
hasCriticality $< expression >: IOProperties$
visibility $< expression >: IOClases$
criticality $< expression >: IOClases$
isWDSetCriticali $< predicate > ...$
setCriticali $< expression >$
$(o : Ontology(IOClasses, IOProperties, IOInstances),$
$ipv : \mathbb{P}(IOInstances \times IOProperties \times IOInstances),$
$i : IOInstances)$
well-definedness $isWDSetCriticali(o, ipvs, i)$
...
...
AXIOMS
IOProperties: $\{visibility, criticality\} \subseteq IOClasses$
IOInstances: $\{visible, hidden, critical, safe\} \subseteq IOInstances$
isVisibleWD: $\forall o, ipv, i \cdot o \in Ontology(IOClasses, IOProperties, IOInstances) \land$
$ipv \in \mathbb{P}(IOInstances \times IOProperties \times IOClasses) \land i \in IOInstances \land$
$isIOOntologyWD(o) \Rightarrow$
$(isVisibleWDi(o, ipv, i) \iff$
$isVaribaleOfOntology(o, ipvs) \land i \in dom(; dom(ipvs)))$
...
...
THEOREMS
setCriticaliThm : $\forall o, ipvs1, ipvs2, i \cdot o \in Ontology(IOClasses, IOProperties, IOInstances) \land$
$ipvs1 \in \mathbb{P}(IOInstances \times IOProperties \times IOInstances) \land$
$ipvs2 \in \mathbb{P}(IOInstances \times IOProperties \times IOInstances) \land$
$i \in IOInstances \land isIOOntologyWD(o) \land isWDSetCriticali(o, ipvs1, i) \Rightarrow$
$(ipvs2 = setCriticali(o, ipvs1, i) \Rightarrow$
$(\forall j \cdot j \mapsto hasCriticality \mapsto critical \in ipvs2 \Rightarrow$
$j \mapsto hasVisibility \mapsto visible \in ipvs2))$
...
...

Code Snippet 5.3: Displayability Theory

outside of the current range, but never both; every detected aircraft must be either displayed or hidden, but never both; critical aircraft must be displayed on the screen edge if they are out of current range; and critical aircraft must always be visible regardless of the pilot's range level.

Formal development. Initially, we develop an `OntologiesTheory` using Event-B theory plugins. Further, this is used for developing the first domain theory, `DisplayabilityTheory` (see Listing 5.3), which axiomatises a collection of specific operators with WD conditions to characterise the displayability properties of interactive critical systems. Several operators are defined with WD condition to express interactive components properties like `isIOOntologyWD`, `visible`, `hidden`, `critical`, `safe`, `hasVisiblity`, `hasCriticality`, `visibility`, `criticality`, `setCriticali`, and `isWDSetCriticali`, and so on. Additional axioms, such as `IOProperties`, `IOInstances`, and `isVisibleWD`, and theorems like `setCriticliThm` are also introduced to encode domain-specific rules, as well as to ensure the correct use of the defined operators. More details about this theory can be found in [19].

Finally, this theory is employed for developing the TCAS case study. First, the `DisplayabilityTheory` is instantiated to define the specific I/O concepts and properties required for modelling the TCAS. The instantiated context is further used for describing the dynamic behaviour of TCAS. The Event-B model of TCAS `TheoryOperatorsBasedModel`

MACHINE TheoryOperatorsBasedModel
SEES InstantiationContext
VARIABLES $system$
INVARIANTS
inv1 : $isVariableOfOntology(aircraftOntology, system)$
INITIALISATION
THEN
act1 : $system : |system' \subseteq instanceAssociation$
EVENT $CorrectAircraftStatusUpdate$
ANY i
WHERE
grd1 : $i \in dom(dom(system))$
grd1 : $ontologyContainsInstances(aircraftOntology, \{i\})$
grd2 : $isVisibleWDi(aircraftOntology, system, i)$
grd3 : $isVisiblei(aircraftOntology, system, i)$
grd4 : $isSafeWD(aircraftOntology, system, i)$
grd5 : $isSafe(aircraftOntology, system, i)$
grd6 : $isWDSetCriticali(aircraftOntology, system, i)$
THEN
act1 : $system := setCriticali(aircraftOntology, system, i)$
...

Code Snippet 5.4: Ontology theory based annotated model

(see Listing 5.4) specifies the safety requirements stated as *a critical aircraft must be visible* formalised using $inv1$. A set of events is introduced to correct update the aircraft status, make the aircraft visible, make the aircraft hidden, and ensure the aircraft safe. The first event `makeAircraftCritical` encodes the updating of system variable where some aircraft becomes critical, making it visible to preserve the invariant. The required guards are provided based on defined theory operators. Other events, `makeAircraftVisible`, `makeAircraftHidden`, and `makeAircraftSafe`, are defined in similar manner using the defined operators provided by `DisplayabilityTheory`.

Table 5.6 shows proof statistics of the TCAS model. The complete development of the TACS resulted in 73 (100%) proof obligations, of which 13 (17%) are automatically proven. The Rodin prover is used to prove the remaining 60 (83%) proof obligations interactively. There are too many interactive proofs, but they may decrease in the long run because theories are developed once and for all, and theorems of the theory must be proved when they are used in model development. Note that the majority of the proofs follow the same pattern, which can be easily automated using proof rules.

Model	**Total number of POs**	**Automatic Proof**	**Interactive Proof**
OntologyTheory	21	0(0%)	21(100%)
DisplayabilityTheory	16	0(0%)	16(100%)
InstantiationContext	4	1(25%)	3(75%)
SetTheoriticOperationsModel	10	2(20%)	8(80%)
TheoryOperatorsModel	22	10(45%)	12(55%)
Total	**73**	**13(17%)**	**60(83%)**

Table 5.6: Proof Statistics of TCAS

Discussions. TCAS development exemplifies the use of ontology for describing domain knowledge, as well as the explicit use of well-founded operators to transfer domain properties when designing any complex system. However, the safety properties are no longer proven inductively for each event, but are encoded in theory as a combination of working hypothesis and theorems. The modelling and verification process is systematic, and the ontology modelling language provides a unified framework for describing domain knowledge. Classical modelling, on the other hand, results in disparate descriptions of domain knowledge. Furthermore, such a modelling approach allows for the separation of

concerns in order to preserve triptych structure ($K, S \vdash R$), as well as the reusability and sharability of domain and system models. Furthermore, the modular architecture of our approach allows for more flexible evolution of both domain knowledge components and system models. Due to the use of theory, there is a lack of proof automation. The majority of the proof are interactive, but they can be discharged by instantiating and simplifying the predicate. Furthermore, we have developed a set of proof rules that can be used for unfolding, rewriting, and simplifying complex predicates for discharging POs.

5.3 Medical Systems

5.3.1 Context

Patient safety is a global challenge that requires practical knowledge and technical skills in clinical assessments, embedded systems, and software engineering including human factors and systems engineering (HFE). Many incidents related to patient safety are due to lack of attention to HFE in the design and implementation of technologies, processes, and usability. The main objective of HFE is to improve system performance including patient safety and technology acceptance [Carayon 2009]. A failure in these systems could result in loss of life, including reputation and economical damage. In fact, any failure in the medical domain is a serious public health problem and poses a threat to patient safety. For example, the USA Food and Drug Administration (FDA) has reported several recalls in which pacemaker, implantable cardioverter-defibrillator (ICD), and Insulin Infusion Pump (IIP) failures are responsible for a large number of serious illnesses and deaths. According to the FDA, 17,323 devices (8834 pacemakers and 8489 ICDs) were ex-planted during 1990-2002, and 61 deaths (30 pacemaker patients, 31 ICD patients) were found to be due to device malfunction, and 17000 adverse-events were reported during 2006-2009, where 41 deaths were found to be due to malfunctioning IIPs [31, 33][77][Chen 2014]. The FDA found that these deaths and other adverse-events were caused by product design and engineering flaws including firmware problems [Maisel 2006]. Critical systems, such as the pacemaker and IIP, need to be better designed to provide the required level of safety and dependability. Moreover, there is an increasing demand for developing embedded safety-critical medical systems to improve reliability, safety, performance and autonomy. Traditional techniques, like testing and simulation, become more crucial, time-consuming and expensive when used to deploy safety-critical medical systems.

Since software plays an important role in the medical domain, regulatory agencies, like the FDA, need effective means to evaluate the software embedded in the devices in order to certify the developed systems, and to assure the safe behaviour of each system [Chen 2014, Keatley 1999, NITRD 2009, Lee 2006]. Moreover, over the past few years, formal methods have also been used in the medical domain to check the correctness of operating modes, functions and desired behaviour of medical devices [Bowen 1993, Jetley 2004][1][98]. Regulatory agencies are striving for rigorous techniques and methods to provide safety assurance. Many people believe that formal methods have the potential to develop dependable, safe and secure systems that are also more amenable to certification with required features that can be used to certify dependable medical systems [Lee 2006, Bowen 1993][1][14][94].

There is a crucial need for new methods and better tool support for the development of safety-critical medical systems. In addition, there is also a need for a new framework to address safety and security issues related to the design and engineering of complex safety-critical medical systems. None of the existing life-cycles models uses formal methods at every phase of the system development, as well as there is no environment model for medical systems, which can be used for simulating and testing the system requirements at an early stage of the system development during design and development. The availability of new methods and tools could significantly save time and cost for developing and certifying safety critical medical systems.

In this context, our proposed framework that can automate the process of developing safety-critical medical systems from requirement analysis to code generation [25] as well as tackling the core challenges of medical domains. In addition, our other contribution is the development of a virtual environment model [52, 69, 77] that can be used for supporting closed-loop modelling [14]. Moreover, we propose a virtual environment model for verification, simulation and clinical trials [65][8]. Several case studies related to medical systems [70, 72] are developed using our proposed formal framework, virtual environment model, and closed-loop model. Some of them are summarised below.

	Result
Condition	POST
{ POST completed without problem }	Pass
{ POST completed and problems are detected }	Fail

Table 5.7: Tabular Expression for POST

5.3.2 Our contributions

This section presents a list of applications developed in the context of medical systems using the proposed framework, virtual environment model, and closed-loop modelling approach based on Event-B.

Note that we chose two complex medical system applications to demonstrate our approach. The first case study is Insulin Infusion Pump (IIP) [25][70, 72], which demonstrates how we used our approaches related to automatic refinement (see Section 3.1), formal framework to model, design and implement the functional behaviour (see Section 3.2), environment modelling (see Section 3.4), automatic code generation (see Section 3.5), and assurance case template (see Section 4.2). The second case study is Cardiac Resynchronization Therapy (CRT) [14][70] to demonstrate simple system modelling and closed-loop modelling, as well as their verification in Event-B using classical modelling and the correct by construction approach. Furthermore, we summarise the implementation of a cardiac pacemaker using concurrent programming languages. This case study will also demonstrate the use of various approaches related to implicit and explicit modelling (see Section 2.1), formal framework to model, design and implement the functional behaviour (see Section 3.2), environment modelling (see Section 3.4), automatic code generation (see Section 3.5), and assurance case template (see Section 4.2).

Insulin Infusion Pump (IIP) [25][9]

Requirements. An insulin pump is a small, software-intensive medical device that allows controllable, continuous subcutaneous infusion of insulin to patients. It delivers physiological amounts[7] of insulin between meals and at meal times. An insulin pump is composed of the physical pump mechanism, a disposable reservoir, and a disposable infusion set. The pump system includes a controller and a battery. The disposable infusion set includes a cannula for subcutaneous insertion and a tubing system to interface the insulin reservoir to the cannula. An IIP device can manage system functions such as stopping insulin delivery, managing reminders, and creating, validating, editing, and setting parameters for basal, bolus, and temporary basal profiles. If there is a problem, the device must stop all active basal or bolus delivery. Furthermore, only one profile can be active at a time, and it must keep a log of all operations. A detailed description about this informal requirements is provided in [25][72]

Formal development. We developed the IIP using the proposed framework described in Chapter 3. Based on this framework, we first elaborate and precisely document the system requirements of an Insulin Infusion Pump (IIP) in tabular expressions, and further, we use our developed tool TX2EB [Jiang 2015] to automate the task of formal model generation from tabular expressions. There are 49 tabular expressions in total to describe IIP requirements. In particular, we use refinement strategies in Event-B [61, 72] to generate formal models and formally verify the IIP requirements. We select a set of tabular expressions (see Table 5.7 and 5.8) related to *power status* from the developed tabular specification for developing an abstract model of IIP. This abstract model describes the power status functionalities by changing the system states *on/off*. In order to develop a formal model from these tabular expressions, we need to identify sets, constants, enumerated types and axioms from the given type definitions in tabular specification to define statistics properties of IIP in a context model that is done automatically using TX2EB. Three enumerated sets *e_pwrStatus*, *e_basicResp*, and *e_postResult* are defined in axioms ($axm1$ - $axm3$).

For the dynamic behaviour of IIP model, a list of variables is extracted from the given tabular specification, which are defined by invariants (*inv1* - *inv5*). A variable *POST* is used to state the result of power-on-self-test, where the

[7] Dosage is prescribed by the doctors

Condition		*Result* c_pwrStatus
c_pwrStatus$_{-1}$ = Standby	EXIST[M_pwrReq]	POST
	! EXIST[M_pwrReq]	NC
c_pwrStatus$_{-1}$ = POST	[POST] = Pass	Ready
	POST = Fail	Standby
c_pwrStatus$_{-1}$ = Ready	EXIST[M_pwrReq]	OffReq
	! EXIST[M_pwrReq]	NC
c_pwrStatus$_{-1}$ = OffReq	M_pwrResp = Accept	Standby
	M_pwrResp = Cancel	Ready

Table 5.8: Tabular Expression for Power Status

$axm1 : partition(e\_pwrStatus, \{Standby\_pwrStatus\}, \{POST\_pwrStatus\}, \{Ready\_pwrStatus\}, \{OffReq\_pwrStatus\})$
$axm2 : partition(e\_basicResp, \{Accept\_basicResp\}, \{Cancel\_basicResp\})$
$axm3 : partition(e\_POST, \{Pass\_POST\}, \{Fail\_POST\})$

$inv1 : POST \in e\_POST$
$inv2 : post\_completed \in BOOL$
$inv3 : c\_pwrStatus \in e\_pwrStatus$
$inv4 : M\_pwrReq \in BOOL$
$inv5 : M\_pwrResp \in e\_basicResp$

Code Snippet 5.5: Excerpt of IIP Model context model and invariants

result *Pass* means the system is safe to turn on, and the result *Fail* means it is unsafe for the system to start. The next variable *post_completed* holds the POST state of the system. The variable *c_pwrStatus* shows the current power status of the system. The variable *M_pwrReq* is used to model the request for power on or power off from the user, and the last variable *M_pwrResp* is used for modelling the response to the system prompt from the user.

To define functional behaviour abstractly, 10 events are derived for controlling the power status of IIP system, in which 2 events are generated from Table 5.7 and 8 events are generated from Table 5.8. These generated events include guards for enabling the given actions, and the actions define the changes to the states of power status (*c_pwrStatus*) and power-on-self-test (*POST*). Here, we provide only two events related to the power status and power-on-self-test in order to demonstrate the basic formalization process. An event *POST_Completed* assigns *Pass_POST* to *POST*, when *post_completed* is *TRUE*. This event is generated from Table 5.7, where the condition and result columns of Table 5.7 show the conditions and actions that are translated equivalently to event *POST_Completed*.

EVENT POST_Completed
WHEN
grd1 : $post\_completed = TRUE$
THEN
act1 : $POST := Pass\_POST$
END

EVENT PowerStatus1
WHEN
grd1 : $c\_pwrStatus = Standby\_pwrStatus$
grd2 : $\exists x \cdot x \in BOOL \land x = TRUE \land x = M\_pwrReq$
THEN
act1 : $c\_pwrStatus := POST\_pwrStatus$
END

Code Snippet 5.6: Excerpt of IIP Model events

Similarly, another event *PowerStatus1* is used to set $POST\_pwrStatus$ to *c_pwrStatus*, when power status is *standby*, and there exists a power request from the user. The condition and result columns of Table 5.8 present the conditions and actions that are translated equivalently to event *PowerStatus1*. The remaining events are formalized in a similar way that are translated from the remaining rows of Table 5.7 and Table 5.8.

Here, we summarise each refinement step of IIP development and omit detailed formalization and proof details. The first refinement introduces *user operations* that may be performed by the user to operate the system for delivering insulin, including other system activities related to creating, removing, activating, and managing the basal profile, bolus profile and reminders. The second refinement is related to *basal profile management* that controls storage and maintenance of basal profiles defined by the user. The third refinement introduces *temporary basal profile management* that covers activating, deactivating and checking the validity of a temporary basal profile. The fourth refinement is

related to *bolus management* that allows for creating a new bolus preset, removing an existing bolus preset, checking the validity of a created bolus preset, and activating a selected bolus preset. The fifth refinement is related to *bolus delivery* enables us to start bolus delivery, to calculate the required dose for insulin delivery, and to check the validity of a calculated bolus and a manually entered bolus. The sixth refinement is related to *reminder management* that stores and maintains reminders. The last refinement is related to *insulin output calculator* that calculates the insulin required throughout the course of the day, by tracking the insulin delivery of the selected basal profile, temporary basal profile and preset bolus. In addition, we introduce several safety properties to ensure the correctness of IIP. We use the Rodin [Abrial 2010b] tool for syntactical and consistency checking in our generated formal specification by defining a list of safety properties. The complete formal specification is available in the appendix of technical report [102], which is more than 1500 pages long.

The IIP formal model includes 16 complex data types, 15 enumerated types, 25 constants, and 263 events that are used to describe static and dynamic properties in the form of system requirements. The formal development of IIP is presented through one abstract model and seven refinement models based on the given system functionalities. Note that each refinement is a collection of tabular expressions, which are introduced in several layers. In this development, we have total 43 refinements for describing functional behaviour of IIP. The generated proof-obligations are related to well-definedness, feasibility, invariants, simulation and guard strengthening. This development results in 444 (100%) proof obligations, of which 342 (77%) are proved automatically, and the remaining 102 (23%) are proved interactively using the Rodin prover and other associated tools, such as SMT solvers (CVC4 [Barrett 2011], Z3 [de Moura 2008], and veriT [Bouton 2009]). Most of the interactive proof obligations are generated from the complex expressions, predicates and preservation of refinement relation between two successive models. To discharge these types of proof obligations, we use Rodin tool interactively by simplifying complex expressions, rewriting expressions, applying proper argument instantiation, and selecting the required proof strategies like PP, ML and SMT solvers. We have successfully discharged all the generated proof obligations that guarantee the consistency checking and refinement checking by achieving the required functional behaviour and safety properties for IIP models. In this work, we also use the ProB tool for validating the IIP requirements by executing abstract and successive refinement models.

Model	**Total number of POs**	**Automatic Proof**	**Interactive Proof**
Abstract Model	3	3(100%)	0(0%)
First Refinement	22	22(100%)	0(0%)
Second Refinement	98	82(83%)	16(17%)
Third Refinement	26	25(100%)	1(0%)
Fourth Refinement	52	45(87%)	7(13%)
Fifth Refinement	54	54(100%)	0(0%)
Sixth Refinement	66	60(91%)	6(9%)
Seventh Refinement	123	51(42%)	72(58%)
Total	**444**	**342(77%)**	**102(23%)**

Table 5.9: Proof Statistics of TCAS

Interactive simulation. For developing an interactive simulation for the IIP models, we use our proposed architecture that supports simulating the developed Event-B models in the form of visual animation by hiding mathematical complexity of the formalised specification. To connect our formalised IIP models with visual animation, we use the existing tool Brama [Servat 2006] that is a plug-in for Rodin. This Brama tool provides an interface protocol to communicate between flash-based visual animation and Event-B model in Rodin. We connect the IIP models with the developed visual animation according to the architecture to facilitate simulation. This visual animation shows insulin delivery activities of IIP according to the physical requirements of a patient by monitoring the desired level of blood glucose level. This graphical animation is very useful for domain experts, such as doctors and physician, to validate the formalised models. If the visual animation does not satisfy the expected behaviour according to domain experts

then we can rework on our previously developed models to correct them.

Implementation. We use our developed tool EB2ALL[8] [88][1] for generating source code from the formal model of IIP. In our case study, we use the EB2C[9] code generation tool to generate source code into C language from the formalised and verified IIP model. Before applying this tool, we refine our concrete model by introducing a new context that contains type definition according to the C language, and by removing the abstract operations and non-supported symbols through applying the data refinement. During the code generation process, the EB2C tool generates source code files corresponding to the concrete machines. The generated file contains constants, type definitions, variables and functions. The generated constants, type definitions and variables are extracted from the contexts and machines of IIP model. A set of functions is extracted from the formalised IIP model. These functions are equivalently translated from the model events.

Discussions. The IIP development exemplifies the use of formal framework that can help the community to develop safety-critical medical systems and to analyse system requirements precisely, including to aid in the process of system certification. The prime benefits of this approach are to improve the process for documenting system requirements, minimising error introduction through automation of refinement, formal development and code generation, improving error detection through rigorous analysis and interactive simulation, and minimising development cost. Applying an iterative process for developing a system from requirement analysis to code generation allows us to carry out rigorous analyses that can verify the important properties of completeness, disjointness, consistency, deadlock-freedom, a satisfaction of high-level requirements, functional correctness, and desired system behaviour according to domain experts through the graphical animation of the formal models.

However, it is desired to have some prior experience in documenting system requirements using tabular expressions. To apply this approach to automatic refinement, the system requirements must be developed in tabular expressions that requires less effort than the stepwise formal development. For example, most of the time we spent documenting the IIP system requirements in tabular form as discussed previously. Moreover, tabular expressions are not easy to maintain for handling large systems. Moreover, in our work, we have used existing software tools to manage different phases of development, which need careful analysis for each phase to ensure the consistency of the system for each phase of development, as we manage several parts of the process manually. The low-level system design depends on the generated code, which is a collection of functions. These functions can be organised according to the architecture prescribed for the implementation of the system. The current code generation tool does not take into account the requirements of safety critical safety system standards. We need to improve our automatic code generator so that it can meet such standards. Otherwise, the source code in the target language can be produced manually from the concrete model following the preferred standards.

Cardiac Resynchronization Therapy (CRT) [14][70]

Requirements. The Cardiac Resynchronization Therapy (CRT) or multi-site pacing device is one of the advanced pacemakers that is designed to maintain heart rate by treating a specific form of heart failure – poor synchronization of the two lower heart chambers. We describe the system requirements of biventricular sensing with biventricular pacing (BiSP), which allows pacing and sensing in the right atrium, left ventricle and right ventricle. Biventricular pacing coordinates the left ventricle (LV) and right ventricle (RV), and intra-ventricular regional wall contractions, by synchronizing with the sinus rhythm. There are various intrinsic activities related to pacing and sensing events that can reset escape intervals, such as atrioventricular interval (AVI) and ventriculoatrial interval (VAI). Biventricular pacing controls the heart rate using various combinations of the timing form events in either LV or RV. For example, the first ventricular sense either from the left or right ventricular chamber can reset the ventriculoatrial interval (VAI) and the heart rate depends on intervals between the first ventricular events in each cycle.

[8] http://singh.perso.enseeiht.fr/eb2all
[9] http://singh.perso.enseeiht.fr/eb2all/eb2c.html

Delays between RV and LV pacing introduce complications in biventricular timings. These timings allow multiple definitions of atrioventricular (AV) and ventriculoatrial (VA) escape intervals. The pacing rate is the sum of the VA and AV escape intervals for dual chamber timing. This definition can be preserved for biventricular timing if the VAI and AVI refer to pacing either the RV for RV-based timing or the LV for LV-based timing. Then the pacing delay can be represented by the RV-LV interval. This interval can be negative, positive or zero as per the occurrence order of the stimulations in both ventricles.

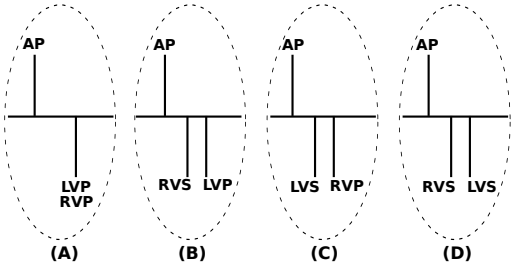

Figure 5.11: Possible scenarios of the biventricular sensing and pacing. AS = atrial sensed; AP = atrial paced; LVS = left ventricular sensed; LVP = left ventricular paced; RVS = right ventricular sensed; RVP = right ventricular paced.

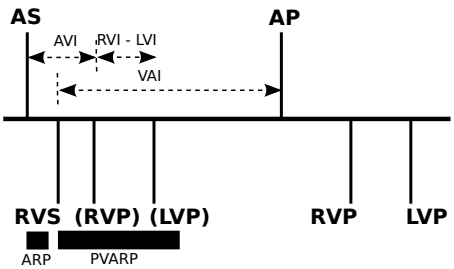

Figure 5.12: Biventricular sensing and pacing (BiSP) with a RVS event

Fig. 5.11 depicts the possible scenarios in a sequential order for biventricular sensing and pacing [Wang 2002]. The possible scenarios are described as follows, assuming normal pacing and sensing activities in the right atrium chamber:

- **Scenario A** shows a situation in which the pacemaker paces in both ventricles after an AV interval in which no intrinsic heart activity is detected.
- **Scenario B** shows a situation in which the pacemaker paces in LV only after an AV interval, while RV pacing is inhibited due to sensing of an intrinsic activity in RV.
- **Scenario C** shows a situation in which the pacemaker paces in RV only after an AV interval, while LV pacing is inhibited due to sensing of an intrinsic activity in LV.
- **Scenario D** shows the case where pacing activities are inhibited in the ventricles due to sensing of intrinsic activities in both LV and RV.

There are various possible scenarios to show biventricular sensing and pacing in order to capture possible behavioural requirements. For example, Fig. 5.12 presents a scenario for biventricular sensing and pacing, in which an event sense related to the right ventricle resets all the pacing intervals for both the right and left ventricles, so pacing is not allowed in the right ventricle or in the left ventricle following a RVS, and a RVS event resets the timing cycle and starts a new VAI. The complete system requirements of CRT is described in [14][70].

Formal development. An abstract model of the CRT pacemaker specifies only pacing and sensing behaviour of three electrodes for each chamber (RA, RV, LV). The CRT pacemaker delivers a pacing stimulus in the RA, RV, and LV as per the patient needs through sensing the intrinsic activities of the heart. The first refinement introduces the timing requirements by defining a logical clock. The pacemaker sensor starts sensing intrinsic activities during certain intervals to avoid sensing errors. A pacemaker actuator delivers a small intense electric pulse whenever the natural pace is absent and intrinsic activity is not detected by the sensor. A sensor can detect an intrinsic activity when the threshold value of the detected signal is greater than or equal to the standard threshold constant or pre-specified threshold by a physiologist[10]. In the second refinement, we introduce the threshold for right atrium, left ventricle and

[10]Standard threshold constant values of atria and ventricular chambers are different.

Model	Total number of POs	Automatic Proof	Interactive Proof
Abstract Model	0	0(0%)	0(0%)
First Refinement	102	101(99%)	1(1%)
Second Refinement	23	23(100%)	0(0%)
Third Refinement	59	57(98%)	2(2%)
Total	**184**	**181(98.37%)**	**3(1.63%)**

Table 5.10: Proof Statistics

right ventricle. In the last refinement, we introduce refractory and blanking periods for atrial and ventricular chambers. These blanking and refractory periods are used to suppress device-generated artefacts and unwanted signal artefacts. These periods are designed to promote appropriate sensing of intrinsic activities, and to prevent over sensing activities in another chamber. This incremental development preserves the required behaviour of the system in the abstract model as well as in the correctly refined models. In order to guarantee the correctness of the system behaviour, we have established various invariants in the incremental refinements.

The Event-B language is used to develop the CRT formal model, and the Rodin platform [Abrial 2010b] is used for verification and validation. This development results in 184(100%) proof obligations, in which 181(98.37%) are proved automatically, and the remaining 3(1.63%) are proved interactively using the Rodin prover (see Table 5.10). These proofs are quite simple, and can be achieved with the help of simplifying predicates. We also used the ProB model checker tool [Leuschel 2003] to analyse and validate the developed models of the CRT pacemaker. ProB animation helps to identify the desired behaviour of the CRT pacemaker in different scenarios and validates the developed formal models. This tool assists us in finding potential problems, and to improve the guard predicates of events. The ProB model checker is able to animate all the possible machines from abstract to concrete level, and to prove the absence of errors (no counter example exist).

Closed-loop model. The CRT model is further extended for developing a closed-loop formal model of the CRT and heart [14], in which the formal model of the heart is used as a virtual environment, and the formal model of the CRT is used to response according to intrinsic activities of the heart (see Fig. 5.13). The main objective of this closed-loop model is to verify and validate the complex properties of CRT pacemaker under the virtual environment, identifying new emergent behaviours and strengthening the given system requirements. As far as we know, this is the first closed-loop formal model of the CRT pacemaker and heart to analyse the functional behaviour of the CRT pacemaker under the virtual environment by satisfying the required safety properties. For developing the closed-loop model, we use the previously developed and verified formal models of the CRT [71] and heart [89][1][98]. In fact, we use our previous works as the basis for developing a closed-loop model of the CRT pacemaker and heart using stepwise refinement from scratch. To check the correctness of the closed-loop system, we introduce several safety properties and discharge all the generated proof obligations at each refinement level.

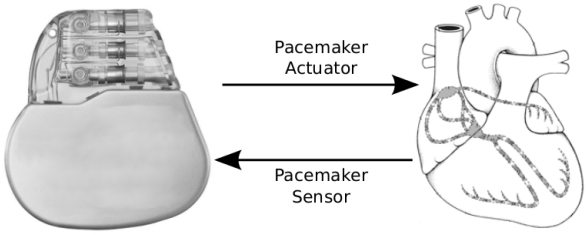

Figure 5.13: The Closed-loop Model[11]

Model	Total number of POs	Automatic Proof	Interactive Proof
Abstract Model	29	25(86%)	4(14%)
First Refinement	138	126(91%)	12(9%)
Second Refinement	36	27(75%)	9(25%)
Third Refinement	72	68(94%)	4(6%)
Total	**275**	**(246(90%)**	**29(10%)**

Table 5.11: Proof Statistics

Table 5.11 shows that the closed-loop model generates 275(100%) proof obligations (POs), in which 246(90%) POs are proved automatically with the help of inbuilt RODIN provers, and the remaining 29(10%) POs are proved interactively by simplifying the predicates using the Rodin provers. It should be noted that the simplifying predicates are quite simple. An integration of the heart model and CRT model generates some extra POs related to the joint behaviour of the closed-loop system and by sharing some common variables by both the heart and CRT models. For example, the current clock counter variable (*now*) is shared, which has been used in the events of the CRT and heart models. The CRT pacemaker shows functional properties of pacing and sensing modes under the virtual biological environment of the heart. The heart model represents normal and abnormal states of the heart, which is estimated by the physiological analysis. A list of safety properties is introduced in the incremental refinements to guarantee the correctness of the functional requirements of the closed-loop model of the heart and CRT pacemaker.

Implementation. In [84][30], we present the development of cardiac pacemaker to evaluate the concurrency model of two programming language subsets that target safety-critical systems development: Safety-Critical Java, SCJ (as subset of the Real-Time Specification for Java), and Ravenscar Ada (a subset of the real-time support provided by Ada 2005). In this experiment, we consider complex operating modes, including automatic switching in different modes. The implementation of operating modes are provided in RTSJ, SCJ and Ravenscar Ada languages. These implementations are based on formalised and demonstrated Event-B models. Note that the simple codes of cardiac pacemaker and CRT are automatically generated using our tool EB2ALL [88], while the concurrent codes of them are derived manually. The complete implementation in different programming languages is provided in [84][30].

Discussions. Refinement is always important for managing system complexity by enabling progressive system development. Thus, in the design of complex systems in the medical domain, as well as other domains, classical modelling using correct by construction is always important. On the other hand, using the closed-loop modelling approach for developing complex systems has shown many benefits: exposing errors that would not have been detected without the environment model; validating the given assumptions; increasing confidence and decreasing failure risks; and promoting the use of the closed-loop modelling approach for identifying emergent behaviour and improving system requirements for developing quality safety-critical systems. Furthermore, by simulating the desired behaviour, this approach allows for the consideration of feedback from domain experts. The closed-loop modelling approach has scientific and legal applications for better understanding, identifying desired functional behaviour, improving system requirements, and meeting certification requirements for developing safety-critical systems.

We conclude from the experiments of concurrent cardiac pacemaker and CRT implementation that for SCJ, the lack of explicit support for watch-dog timers results in a software architecture in which the time at which significant events occur must be saved, and polling must be used to detect their absence. In contrast, Ravenscar Ada supports for primitive timing events allow the construction of a highly optimised reactive solution. In addition, we identify an ease-of-use issue with Ada for developing small reactive systems [30]. The issue is that Ada defines program termination solely in terms of whether all tasks have terminated. To avoid this unexpected premature program termination, our work proposes simple changes to the program termination conditions in the language so that the environment task of

[11] The image of CRT pacemaker is adapted from: http://www.amayeza.co.za/files/content/images/img331.jpg

an active partition terminates when all its dependent tasks have terminated and the partition has no active timing events and no handlers are attached to interrupts that are to be serviced by the partition.

Summary of our contribution for developing case studies:
This chapters summaries the development of safety-critical cyber-physical systems to address the core challenges of hybrid systems, interactive systems, and medical systems. These systems are built using our proposed generic, reusable, and extensible formal modelling framework, design patterns, and correct-by-construction approach. The methodologies associated with domain knowledge engineering, system modelling, and system certification have been successfully used in the development of various types of systems. We have used the Event-B modelling language as well as its plugins extensions related to theory development and code generation. Most of the case studies are taken from the real-world industrial examples. The development of different classes of systems from various domains ensures that the approaches, including techniques and tools, proposed in the various chapters are usable, reliable, maintainable, portable, efficient, correct, and scalable.
Project: ANR-DISCONT, ANR-FORMEDICIS, IRT-INGEQUIP, ORF-RE-Certification of Safety-Critical Software-Intensive Systems, NSERC-NECSIS, EPSRC-HiJaC
Student supervision: Ismail Mendil (PhD, 2019 – 2023), Guillaume Dupont (PhD, 2017 – 2021), Romain Geniet (MS, 2017), Sasan Vakili (M.A.Sc, 2015), Yanjun Jiang (M.A.Sc, 2015), Mischa Geven (M.A.Sc, 2014), Nicholas Proscia (M.A.Sc, 2014)
Publications: [20, 22–25, 30][14, 15][41, 42, 44–48, 51, 53, 55, 56, 58, 61, 67, 69–72, 74, 75, 77, 80, 84][9][103]
Software: EB2ALL, TX2EB, Fluid2EB, CZT, theories, domain-specific theories, approximation, modelling and designing patterns, simulation framework, and models.
Models: https://www.irit.fr/~Guillaume.Dupont/models.php

CHAPTER 6

Conclusion and Perspectives

Today, we are surrounded by digital technologies and highly complex systems, where cyber-physical systems have taken central place in our lives and in various industrial sectors to improve our lives and boost economies. However, while they brought significant improvements to our lives and economies, they also brought a number of important issues in their development, particularly modelling of cyber and physical systems, composition, safety and security, and certification, which could jeopardise our well-being as well as the development and reliability of cyber physical systems. According to [NIST 2002], as software errors have a significant economic impact, formal verification has gradually become part of the development process. However, formal methods have mostly been used for niche purposes, such as behaviour analysis of a complex system, exposing flaws in system requirements, finding ambiguities in system specifications, and articulating implicit assumptions, and so on. In fact, the current software development process in industries, particularly for developing safety-critical cyber-physical systems, limits the scope of formal methods. This manuscript summarised our contributions to the development and investigation of methods for dealing with formal engineering processes such as modelling, refinement and simulation, domain knowledge engineering, design automation, heterogeneity, composition, safety, and certification issues for safety-critical cyber physical systems.

In this manuscript, we presented how formal methods, including techniques and tools, can be used to address various challenges in safety-critical cyber-physical systems. Our contributions are focused on the development of new analysis methods, with three main areas: domain knowledge engineering (reported in Chapter 2), system modelling (reported in Chapter 3), and certification and assurance case templates (reported in Chapter 4). These theoretical and applied contributions are required to address key issues from various domains of safety-critical cyber-physical systems. Note that all of the methods we proposed have not only been proven correct, but have also been implemented and validated through the development of complex examples from various domains (reported in Chapter 5).

The first part of our research is motivated by Question 1: *What is domain knowledge, and what is the relation between domain model and system model?*. This was the case for our work to understand domain concepts, and implicit and explicit modelling concepts in order to identify a set of modelling patterns applicable for refinement-based formal development as well as for model refactoring (Section 2.1) in order to support modularity, domain knowledge integration, re-usability, and maintainability. Moreover, this was also the case when Event-B theories were applied to handle domain knowledge as ontologies (Section 2.2) to represent explicit domain knowledge for complex systems. This approach enables model re-usability, reduces modelling effort, improves safety concepts, and aids in asynchronous evolution in both domain and systems models.

The second part of our research is motivated by Question 2: *How should a complex system and its environment be designed using the correct by construction state-based method?* This was the case when we proposed generic formal frameworks, modelling patterns, architecture patterns, approximation patterns, refinement automation, simulation, environment modelling, and code generation, for modelling, designing and verifying safety-critical cyber-physical systems. The proposed generic framework consists of a large set of theories that extend Event-B with mathematical features required to model continuous behaviours (e.g., differential equations). In particular, we defined three formal patterns, inspired by general practice in hybrid system designs: approximation, centralised control with multiple plants, and distributed hybrid systems (Section 3.1). The development life-cycle is proposed for rigorous development of critical systems, from requirement analysis to code generation. In a similar vein, to address the key challenges and modelling issues of safety-critical interactive systems, we contributed a formal framework, F3FLUID (Formal Framework For FLUID), as well as the structuring of Event-B models using *model-view-controller* (MVC) (Section 3.2) pattern. Furthermore, we also presented the reflexive EB4EB framework, which allows users to explicitly manipulate Event-B features through the use of reflection and meta-modelling concepts, as well as the extension of this framework to improve Event-B reasoning mechanisms for expressing deadlock freeness, invariant weakness analysis, reachability, and generation of new proof obligations associated with liveness properties expressed in LTL (Section 3.3). To design an environment model, we proposed the development of a virtual environment model, as well as its application

for verification, simulation, and clinical trials, including techniques and tools (Section 3.4). Finally, we contributed to the development of the code generation tool, EB2ALL, by developing a new plug-in, EB2Sol, for generating Solidity code from Event-B models for the Ethereum platform, as well as extending existing plugins (EB2C, EB2C++, EB2J, and EB2C#) to incorporate new modelling constructs like quantifiers (Section 3.5).

The third part of our research is motivated by Question 3: *What can be done to assist with software and system certification?*. This was the case when adapting formal methods for developing integrated verification framework that can be used to aid in the certification process. It was accomplished as part of the RATP (Parisian Public Transport Operator and Maintainer) B-PERFect project, which aimed to apply formal verification using the PERF approach to integrated safety-critical models of embedded software related to the railway domain expressed in a single unifying modelling language, HLL. The proposed approach employed Isabelle/HOL as a unified logical framework for describing formal semantics and formalising the transformation relationship between both modelling languages. The developed Isabelle/HOL models were validated to ensure the accuracy of our translation process (Section 4.1). Another contribution to address certification challenges, we proposed the assurance case template. The assurance case template is a key tool for assisting in the certification process of safety-critical cyber-physical systems. It provides explicit guidance to regulators and manufacturers on how to write an effective assurance case for a specific product, as well as developing and evaluating assurance cases for certifying safety-critical cyber physical systems (Section 4.2).

Finally, we demonstrated the usability, reliability, maintainability, portability, efficiency, correctness, and scalability of our proposed methods, techniques, and tools for domain knowledge engineering, system modelling, and certification and assurance cases on a variety of complex safety-critical cyber physical systems. We focused on three types of systems, including several examples: *Hybrid Systems*, *Interactive Systems*, and *Medical Systems*.

All of these are much broader research challenges, for which Chapters 2, 3, 4 and 5 only provide preliminary and some ground-breaking results to answer the raised questions; however, there are significant prospects for our research in the coming years. We conclude this manuscript with some sparkling ideas for future research, both short and long term.

6.1 Perspectives about theories, models, patterns and tools

The increasing complexity and use of software in safety-critical cyber-physical systems motivates the development of new methodologies and software engineering principles, as well as scaling them for sophisticated and advanced systems in the future by incorporating other engineering methods such as machine learning, swarm intelligence, blockchain, and so on. Our current research works address some key issues related to domain knowledge engineering (reported in Chapter 2), system modelling (reported in Chapter 3), and certification and assurance case templates (reported in Chapter 4). In this direction, we present some perspectives on modelling theories, design patterns, simulation, certification and tools.

Extending systems and domains theories

In our work for system and domain modelling, we defined some theories to characterise various aspects of cyber physical systems. For example, ordinary differential equation theories for dealing with continuous behaviour, ontology-based theories for dealing with interactive systems, standard conformance theories, and so on. It should be noted that once defined and proven, these theories can be reused in any other system for specifying and verifying them. These theories are currently being developed on an ad hoc basis to meet our objectives; however, they are not yet complete. For example, partial order differential equations, which are highly required in complex cyber physical systems, are not defined in current ODE theories. As a result, we would like to extend these theories to cover a broader range of systems and domains. Furthermore, a set of libraries can be developed as a package of domain specific theories and system theories to reduce modelling and proofs effort, and advanced users do not need to rework on them. These libraries, which contain generic properties, are required to handle a wide range of complex applications such as autonomous driving, trains, robotics, drones, and airplanes. This is an important step toward modelling safety-critical

cyber-physical systems, which may aid in the certification process as well as the incorporation of standard guidelines into product development.

Reflexive EB4EB framework

As Event-B lacks support for checking liveness and temporal properties, there is a strong desire among research communities to include such property verification in the core of the modelling language. Our work, presented in Chapter 3, provides initial concepts and vision by introducing the reflexive EB4EB framework that allows explicitly manipulation of Event-B features using meta modelling concepts. This framework is easily extended for advanced level reasoning such as checking liveness and temporal properties, reachability, deadlock free, non-intrusive analysis, and so on [34, 35]. In this direction, we intend to analyse and identify Event-B refinement operations and their semantics for inclusion in the EB4EB framework. Furthermore, this framework can be extended to include a reasoning mechanism for dealing with domain-specific analyses of Event-B models, such as continuous behaviours, human machine interaction, and so on. From an application standpoint, this framework can be used to analyse and certify existing plug-ins, such as code generation and composition/decomposition, among others. There is also the possibility of importing and exporting the meta theories and models into other proof assistants for certification purposes, such as Coq [Bertot 2010], PVS [Owre 1992], and Isabelle/HOL [Nipkow 2002].

Simulation and animation

Simulation and animation are traditional and widely used techniques for designing complex systems to validate the correctness of the behaviour. Regardless of whether we use formal techniques or not, we must use such techniques to simulate the desired behaviours of both cyber and physical systems. The life-cycle process is presented in Chapter 3, where simulation and animation is considered as a potential step for detecting unwanted functional behaviour. The use of simulation and animation techniques remains a challenge due to the continuous nature of safety-critical cyber-physical systems. It is important in current system development to simulate and animate discrete and continuous controllers for hybrid systems resulting from various refinement levels, as well as to design new tool support for animating and simulating hybrid systems. In this context, we have proposed a hybrid Event-B model transformation into Simulink/Stateflow models [MathWorks 2021], with the discrete part developed in Stateflow and the continuous part modelled in Matlab [MATLAB] user defined functions. Our framework has been applied to a variety of control theory problems, including computer-assisted cars, the European Train Control System (ETCS), water tanks, robots, and inverted pendulums. This work can be extended to analyse reachability, as well as adapted to work on continuous Event-B models to generate Simulink models and to work on discretisation to convert a continuous model to a discrete one.

The ProB [Leuschel 2003] model checker is important for model checking and animating discrete Event-B models. There is no support for dealing with continuous nature. In this direction, the ProB [Leuschel 2003] model checker can be extended to support continuous features for model checking, as well as the simulation and animation of cyber physical system models. Another potential work is the development of a co-simulation approach to simulate CPS models using Functional Mock-up Interface (FMI) [Blochwitz 2012]. The proposed approach can be implemented as an extension of the Rodin platform allowing for the heterogeneous composition and simulation of continuous Event-B and FMI models.

In Chapter 3, we have presented a promising simulated biological environment model that can be used during product development as well as a diagnostic tool to diagnose or understand patient needs. The basic concepts have been described, and we intend to develop the simulation using linear, nonlinear, and ODE equations before implementing it on a hardware platform to use as a test bench for developing medical devices and clinical trials.

Modelling and design patterns

The Chapters 3 has presented several frameworks, including modelling and design patterns, to deal with modelling problems of various components of safety-critical cyber-physical systems such as controllers, physical systems, human

machine interfaces, and so on. These defined frameworks can be extended, and new patterns can be introduced to address specific problems. We thus wish to explore these framework further in order to extend different classes of modelling and design patterns. We now describe potential extensions that appear promising to us.

The hybrid modelling framework presented in Chapter 3 only addresses abstract modelling of continuous and discrete behaviours, with no special treatment for transforming continuous behaviour into discrete behaviour. Thus, we wish to define discretisation operations that can be used to transform a continuous model into a discrete model while preserving all of the required properties defined at the abstract level. The defined discretisation operations must provide a generic implementation that can be used with real numbers.

Refinement is important because it enables us to build a complex system by introducing system requirements incrementally. Applying refinement to the design of a complex system is difficult, and the practicalities of performing automatic refinements are largely unknown. In Chapter 3, we have presented refinement strategies, including tool support TX2EB, for automating the process of generating Event-B formal models from tabular expressions. However, the presented approach only supports horizontal condition tables (HCT) and limited expression types such as FOL. There is no support for natural language, which is also widely used in tabular expressions. Thus, we would like to extend our refinement strategies to include support for natural language expressions as well as other tabular layouts. Furthermore, we would like to include these extensions in our TX2EB developed tool.

Refactoring operations allow us to restructure the formal model, specifically domain knowledge and system behaviour, without changing the functional behaviour of a system. In Chapter 2, we have presented refactoring approach that refactors complex models to extract domain specific knowledge and systems specific dynamic properties. We want to provide a semantic description and formalisation of the proposed refactoring operations, as well as investigate the refactoring laws in order to modify the refinement strategy for restructuring the formal development and optimising the refinement levels. Furthermore, there is an obvious need for tool development to support proposed refactoring operations in order to automate the process of refactoring complex formal models.

Certification

One of the primary goals in the field of safety-critical cyber-physical systems is certification, which ensures the safe use of the developed product. There are several methods for assisting with the certification process. Chapter 4 has introduced an integrated formal framework (PERF methodology) for certifying critical systems, as well as assurance case templates that can help regulators and manufacturers develop safe systems and certify them. There are numerous new research avenues in both approaches, some of which are described here.

An integrated formal framework is used on concrete B models, but this verification approach, which includes transformation, can be used at higher abstraction levels of B models, allowing us to optimise our verification approach by reducing state variables and focusing on precise abstract properties rather than addressing code level properties. In addition, we want to extend our B2HLL tool to support different B language modelling constructs, as well as extend the defined semantics of B and HLL in Isabelle/HOL to characterise the B and HLL semantics for validating the transformation from B to HLL at any level. In this work, the encoding of B and HLL models into Isabelle/ HOL is done by hand. We intend to develop a tool that transforms B and HLL models into Isabelle/HOL so that the models can be easily checked and a certificate confirming the correctness of the B to HLL transformation for a given model can be generated directly. We have used S3 solvers [Breton 2016] to verify the HLL code, which has a state explosion problem, so there is another interesting extension to restructuring models using composition/decomposition techniques and reducing state space problem. This can be accomplished by introducing operations (functions and procedures) as abstracted black boxes defined by their before-after predicates.

The development of assurance cases is one of the practices recommended by regulators and manufacturers in the development of safety critical systems. In Chapter 4, we have presented the development of an assurance case template as a standard to meet the needs of regulators and stakeholders by focusing attention on critical parts and assisting in the certification process. One of the driving forces behind this work is the desire to be able to deal with the complex, multi-disciplinary, interconnected nature of today's and tomorrow's CPS. Developing and certifying such systems to ensure their safety, security, and dependability appears to necessitate a significant shift in how we guide

people through the development and certification of these systems. We believe that a collaborative effort to understand how to create effective assurance case templates for this purpose will be one effective step toward meeting future challenges. Another important challenge is the development of incremental safety cases. If any changes are made to the developed safety cases, it is economically and logistically impossible to rebuild the safety cases from the ground up. Our proposal is to create explicit safety cases. These will serve as the foundation for analysing how incremental design changes affect existing safety artefacts. Then, flaws in the safety case can be identified and strategies to address the deficiencies, establishing the groundwork for eliciting the desired engineering principles. This hypothesis must be carefully developed and systematised in future work so that it can be evaluated in carefully conducted experiments.

Extending EB2ALL for code generation

EB2ALL[1] [EB2ALL 2022] is a collection of code generation plugins that allow us to generate source code from Event-B formal specifications in a variety of target programming languages such as C, C++, Java, and C#. The first prototype was developed in 2010 as part of my thesis work. Since then, it has been improved by adding new features and fixing bugs reported by users. We recently added new plugins, EB2Sol [EB2Sol 2021], EB2Ada and EB2Py, to generate Solidity code contracts, Ada code and Python code from an Event-B specification. The EB2ALL is still in development, and we have identified several new challenges. In the near future, we intend to expand the development of the code generation tool, EB2ALL, to support a wider range of programming languages. The current version of EB2ALL does not take the requirements of safety-critical safety system standards into account. As a result, we would like to improve the automatic code generator so that it can meet such standards, as well as provide an industrial version for generating certified code. Furthermore, the EB2ALL can be extended to generate code for complex interactive systems from an Event-B specification, particularly for designing graphical interfaces. Similarly, it can also be extended for controller implementation generated from an Event-B model with continuous and discrete modelling features.

Finally, it is important to ensure that the translation process, including the generated code, is correct. Hence, we intend to use the integrated formal framework developed in Chapter 4 to certify the code generation process of EB2ALL plugins, as well as to guarantee the correctness of the generated code in accordance with the formalised and proven Event-B models. Such an approach will be useful in meeting regulatory requirements and obtaining certification when designing safety-critical cyber-physical systems.

Theories consistency and proof processes

All developed theories for basic and advanced mathematical properties, domain knowledge, discrete and continuous systems, and standards are axiomatically defined and supported by numerous axioms. These axioms are defined for describing complex and sound properties. There is no standardised mechanism in Event-B theory plugin for validating all defined axioms. As a result, there is potential work that needs to be done in order to validate all of these defined axioms as well as identified imprecise and inconsistent axioms. This can be accomplished by identifying a minimal set of axioms and embedding them in other interactive theorem provers such as Coq, HOL/Isabelle, PVS, and so on.

The Event-B supports both proof automation and manual interaction proof processes. Proof automation allows us to discharge a large number of POs for discrete systems, but proof automation is lacking for theories development and continuous models, and most POs can be discharged manually. In fact, the manual interaction is inevitable in the proof process due to the current state of theorem prover technology. Thus, we want to improve the proving process by adding new theorem provers and SMTs, as well as new rules. We can introduce solvers for analysing continuous behaviour represented in differential equations in the case of continuous modelling. These solvers are already included in Mathematica, Matlab, Maple, SymPy, and a variety of other advanced engineering software. To improve proof automation, we intend to integrate these solvers into the core of the proof process. This is an important step toward improving the proving mechanism for cyber-physical systems, particularly for verifying continuous behaviour.

[1] http://singh.perso.enseeiht.fr/eb2all

6.2 Perspectives about safety-critical cyber physical systems

Our main research topic has been, and continue to be, software engineering principles and techniques, including formal methods to design and implement safe secure cyber physical systems. The Chapters 2, 3, and 4 describe our main contributions to address different kinds of key issues for developing safety-critical cyber-physical systems to cover various domains. We have only addressed a few aspects of CPS, but there are many more to consider. In the future, we would like to extend our approaches to cover other range of issues as well as chosen application domains.

AI/ML integration in safety-critical cyber-physical systems

Recent advancements in Artificial Intelligence/Machine Learning (AI/ML) enable the solution of highly complex classification and recognition problems (e.g., image classification, speech recognition, object detection etc.) and incorporating intelligence and autonomy [4]. There are several applications of safety-critical cyber-physical systems that require a high level of intelligence as well as autonomy in an uncertain environment to perform complex tasks such as perception, planning, and control. On the other hand, there are numerous applications, such as autonomous driving, robot surgery, unmanned underwater vehicles, aircraft collision avoidance systems, and so on, that cannot be imagined without incorporating AI/ML into the main development life cycle of safety-critical cyber-physical systems [Urban 2021].

The integration of AI/ML in safety-critical cyber-physical systems is a significant challenge due to a lack of trust in AI/ML learning algorithms [Spiegelhalter 2020] and a lack of predictive accuracy [Szegedy 2014], and a very high level of insurance is required to ensure the reliability and proper functionality of safety-critical cyber-physical systems. However, the primary barriers to AI/ML adoption are that real-world deployment requires trust in both software and hardware. Without trust, AI will not reach its full potential, particularly in safety-critical domains such as avionics, robotics and automation, transportation, healthcare, banking, and so on. Formal methods have been successfully used in the design of critical software systems [Woodcock 2009, Berry 2007, Liu 1995], such as avionics [Julian 2016, Britt 1994], microprocessor [Miller 1995], medical systems [Harrison 2017][1]. In contrast, the use of formal methods for developing machine learning algorithms as well as integration of machine learning components in cyber physical systems is extremely limited [Urban 2021]. However, some solutions have been proposed, such as the use of SMT solvers for neural network verification [Pulina 2010, Pulina 2012] and run-time verification for learning-enabled components [Wu 2021], and so on. The authors of [Urban 2021] describe a comprehensive survey of the use of formal methods in machine learning.

Our research vision is to make AI trustworthy through tight integration of formal methods (FM) and programming languages (PL) with AI/ML. Thus, we want to explore state based formal methods for verifying machine learning software, as well as the integration of machine learning components into the design of safety-critical cyber-physical systems. In this direction, we have identified some of the core AI/ML areas that can be investigated further in order to meet our long-term goal of integrating AI/ML for safety-critical cyber physical systems.

Knowledge reasoning. Chapter 2 discussed our work on domain knowledge engineering, specifically describing domain concepts in the form of theories composed of operators, well-defined conditions, theorems, and proof rules. This work can be easily extended to develop theories for logical formalism used in designing the strategic behaviour of intelligent agents, such as fuzzy logic, dynamic logic, and description logic. Developing a set of theories allows the formal community to directly use them in the formal development process to specify AI/ML components, including required behaviour related to classification and decision making as well as reasoning about core properties. Moreover, this theory extension will be generic and reusable, allowing it to be used in the design of strategic behaviour in a complex system.

Training data analysis and validation. The quality of machine learning approaches is solely dependent on data, thus training data validation is very important. For example, verification methods detect vulnerabilities in data such as duplicate, missing or incorrectly tagged data, and tempered data. As a result, developing formal models based on

refinement for analysing data properties and qualities is an important future direction. In this context, we can extend our simulation framework [25][1] to develop data preparation software based on formal techniques. Furthermore, there is another possible direction for deriving the inference of assumptions from training data in order to determine best abstractions that can be used for testing data validation and improving data preparation software.

Robust, trustworthy, scalable and verified AI/ML. Traditionally, formal verification approaches have been used to build trust in AI systems. The wide adoption of AI/ML systems introduces new challenges such as lack of generalisation, non-interpretability, and non-explainability, where formal methods could make foundational contributions. Our goal to address these issues by developing a formal framework by extending the framework proposed in Chapter 3 by integrating the machine learning software and components in the standard development process. In this line of work, we need to develop new theories for discovering and encapsulating domain-specific precise abstractions. Furthermore, the defined operators of these theories ensure the correct definition based on well-defined conditions as well as semantic-aware solutions that will allow the design of learning software and components by correct by construction synthesis. In addition, these theories and their operators enable the design of robust training processes by focusing on the relevant input space and imposing constraints on the training process to ensure that the required properties are met. Note that the verification methods should be able to verify the core behavior of trained models for any input space, which can be accomplished by exploring approximation theory to obtain another closed model from the original reference model. If the verification fails, the source of the fault must be identified in order to localise and guide any repairs. Such mechanisms enable the interpretability of models such as neural networks.

Certifying AI and ML components. Certifying AI/ML components in the context of critical systems is a challenge. Assurance cases [Kelly 1998] have been used successfully for certification purposes to elaborate safety cases concepts in structured form. The use of AI/ML components in critical systems introduces new challenges for introducing and integrating new safety concepts in safety cases. In Chapter 4, we presented Assurance Case Templates that can be used to guide the development of a software-intensive critical system. In this line of work, we will extend the Assurance Case Templates to support safety cases for AI/ML components in order to provide a structured argumentation to demonstrate safety in the future. Furthermore, the extended Assurance Case Templates can be used as a standard for designing safety cases for any system with AI/ML components by providing core evidences and other artefacts related to dataset, training specification, training and testing models, environment, coverage, stability and robustness, testing completeness, and so on.

Swarms of safety-critical cyber-physical systems

Swarm intelligence is a multi-agent framework inspired by swarm behaviour. Each agent in a swarm intelligence system acts autonomously, reacts to dynamic input, and collaborates with other swarm members without centralised control. The entire system, as a collection of multiple agents, is expected to exhibit global patterns and behaviours. Swarm intelligence is set to dramatically transform our world by bringing numerous benefits to many sectors such as industrial production, autonomous driving, smart traffic, healthcare, military activities, exploration and emergency response, space missions, and so on [Schranz 2021].

Natural disasters such as floods, tsunamis, earthquakes, fires, and major accidents in multi-story buildings and transportation may be life-threatening and have serious consequences such as death due to a lack of resources or inaccessibility of the affected areas. Furthermore, there is a growing demand for such systems in other areas, such as battlefield surveillance and reconnaissance, where drone swarms are required [Lukina 2018]. The deployment of swarms of cyber physical systems to these areas always ensures quick management and response to deal with the situation. In this direction, we have identified some of the core areas that can be investigated further in order to meet our long-term goal of designing swarms of safety-critical cyber physical systems.

Swarm specific domain knowledge. This is an active research area aimed at identifying generic domain knowledge in order to design a class of swarm systems. The SWARMs ontology [Li 2017] is defined to interconnect a number

of domain-specific ontologies, such as mission and planning, networking and communication, robotic vehicle, and environment and sensing ontologies. This ontology play a crucial role for reasoning about core functionalities as well as uncertainty. The work of Chapter 2 can be used to formally describe the swarms ontology using Event-B theory, which can be used to specify swarm behaviour, interaction mechanism, centralised/decentralise controls, communication, mission planning and domain specific knowledge.

Formal framework for swarm development. Swarms are distributed multi agent autonomous systems, where several challenges exist related to mission planning, behaviour and coordination and architecture. Our mission is to advance and design a unifying framework for designing swarms of safety-critical cyber-physical systems by extending the development life-cycle, including techniques and tools, presented in Chapter 3. The unifying framework will cover fundamental areas of swarm and autonomous systems through integrating domain knowledge, modelling and verifying continuous and discrete behaviour, coordination and communication protocol, composition, simulation, and code generation for a target platform.

Swarm certification. Swarms are now used in both critical and non-critical domains. They must be certified if they are used in public to ensure the correct behavior. In this line of work, we can use our certification framework presented in Chapter 4 to analyse swarm functionalities, path planning, and networking protocols. Furthermore, we can use Assurance Case Templates for any class of swarm, such as unmanned aerial/ground vehicle robots, medical surgery robots, and so on, from different domains to develop assurance safety cases for specific products within the identified product scope.

Blockchain technology integration in safety-critical cyber-physical systems

Today, we are surrounded by various types of critical and non-critical cyber-physical systems, and these systems are changing the way we interact with physical systems. The centralised approaches for safety-critical CPS are incapable of handling the unique challenges of CPS due to the complexity, heterogeneity, constraints and dynamic interactions. In fact, decentralised approaches are required to solve the particular problems [Dedeoglu 2020]. There are several complex safety-critical cyber physical-systems, such as smart grids, healthcare, industrial production processes, autonomous vehicles and drones, home automation, and so on, that require decentralised methods to address complex issues, such as safe interactions among distributed CPS components. Blockchains, with their inherent combination of consensus algorithms, secure protocols, and distributed data storage, can be used to support decentralised framework for building safe, robust and reliable safety-critical cyber-physical systems [Rathore 2020]. The incorporation of blockchain technologies into CPS reduces the risks associated with centralised architecture. We have identified new promising research avenues that can be pursued in order to achieve our long-term goal of integrating blockchain technologies for the decentralisation of safety-critical cyber physical systems.

Integrating blockchain concepts. Integrating blockchain technologies and their inherent characteristics to support a decentralised framework for designing complex safety-critical cyber-physical systems is an intriguing problem, as is using them in the development of safety-critical CPS such as smart grids, healthcare, industrial production processes, autonomous vehicles and drones, home automation, and so on. Such integration will have a significant impact in the future, as well as dealing with security and scalability issues caused by CPS's centralised architecture. This approach allows for greater transparency, trust, security, and immutability, as well as complete control for the safety critical cyber physical systems. These objectives can be met by developing new theories related to the blockchain technologies proposed in Chapter 2. Furthermore, these theories and their operators enable the design of decentralised systems with the necessary properties. In addition, we can scale this approach for security modelling as well as formal analysis.

Validating smart contracts. Smart contracts are self-executing code that follows some predefined rules, and discovering any errors late can be very costly. To improve the validation process, we may use the meta modelling EB4EB

framework proposed in Chapter 3 for validating contracts through formal reasoning of process mining in the future. This can be accomplished by transforming event logs into formal models and then performing formal verification to check (non)conformity in smart contracts as well as other required properties. In the same vein, we can transform smart contracts into formal models for checking deadlock freeness, bad-events, invariant preservation, reachability, temporal properties, and so on, using meta modelling framework EB4EB.

Designing smart contracts. Ethereum [Wood 2014] is an open-source blockchain computing platform with contract functionality. To manage digital assets, it executes byte code on a simple Solidity[2] [Solidity Documentation 2023] stack machine, which is difficult due to Ethereum's openness that allows both programs and anonymous users to call into the public methods of other programs. In such cases, combining trusted and untrusted code for large applications can be risky and lead to catastrophic failure. For example, TheDAO [Zhao 2017] is hacked by an attacker by examining EVM semantics to transfer 50 million USD in Ether. There is also a strong need for the use of formal methods for developing smart contracts that can be used by decentralised safety-critical cyber-physical systems. Thus, our another research avenue is for analysing, verifying, and implementing smart contracts using the correct by construction approach. The use of refinement enables the progressive development of smart contracts as well as the verification of the required properties. We have done some preliminary work in [10, 11]. This is the fundamental building block that must be formally designed and proven in order to incorporate blockchains into safety-critical cyber-physical systems.

[2]https://github.com/ethereum/solidity

APPENDIX A

Publications

Book

[1] Neeraj Kumar Singh. *Using Event-B for Critical Device Software Systems*. Springer-Verlag GmbH, 2013.

Book Editor

[2] Dominique Méry and Neeraj kumar Singh. *Modelling Software-based Systems*. Wiley-ISTE, 2024 (in process).

[3] Rajiv Pandey, Nidhi Srivastava, Neeraj Kumar Singh, and Kanishka Tyagi. *Quantum Computing: A Shift from Bits to Qubits*. Springer, 2023.

[4] Rajiv Pandey, Sunil Kumar Khatri, Neeraj kumar Singh, and Parul Verma. *Artificial Intelligence and Machine Learning for EDGE Computing*. Academic Press, 2022.

Conference Proceedings Editor

[5] Klaus-Dieter Schewe and Neeraj Kumar Singh, editors. *Model and Data Engineering - 9th International Conference, MEDI 2019, Toulouse, France, October 28-31, 2019, Proceedings*, volume 11815 of *Lecture Notes in Computer Science*. Springer, 2019.

Preprints

[6] Flavio Ferrarotti, Peter Rivière, Klaus-Dieter Schewe, Neeraj Kumar Singh, and Yamine Ait Ameur. A Complete Fragment of LTL(EB). 2023 (Submitted).

[7] Peter Rivière, Tsutomu Kobayashi, Neeraj Kumar Singh, Fuyuki Ishikawa, Yamine Ait Ameur, and Guillaume Dupont. On-the-Fly Proof-Based Verification of Reachability in Autonomous Vehicle Controllers Relying on Goal-Aware RSS. 2023 (Submitted).

[8] Neeraj Kumar Singh. A virtual environment model for verification, simulation and clinical trials: Glucose homeostasis. 2022 (Preprint Under Review).

[9] Neeraj Kumar Singh, Mark Lawford, Thomas Stephen Edward Maibaum, and Alan Wassyng. Refinement automation and proof-based development using tabular expression. 2022 (Preprint Under Review).

Book Chapters

[10] Raju Halder, Md. Imran Alam, Akshay M. Fajge, and Neeraj Kumar Singh. Analyzing information flow in solidity smart contract. In R. Pandey, S Goundar, and S. Fatima, editors, *Distributed Computing to Blockchain: Architecture, Technology, and Applications*. Elsevier, 2022.

[11] Neeraj Kumar Singh, Akshay M. Fajge, Raju Halder, and Md. Imran Alam. Formal verification and code generation for solidity smart contracts. In R. Pandey, S Goundar, and S. Fatima, editors, *Distributed Computing to Blockchain: Architecture, Technology, and Applications*. Elsevier, 2022.

[12] Neeraj Kumar Singh, Yamine Aït-Ameur, and Dominique Méry. Formal ontological analysis for medical protocols. In Yamine Aït-Ameur, Shin Nakajima, and Dominique Méry, editors, *Implicit and Explicit Semantics Integration in Proof-Based Developments of Discrete Systems: Communications of NII Shonan Meetings*, pages 83–107. Springer Singapore, Singapore, 2021.

[13] Manamiary Bruno Andriamiarina, Dominique Méry, and Neeraj Kumar Singh. Chapter 5 incremental proof-based development for resilient distributed systems. In Alexander Romanovsky and Fuyuki Ishikawa, editors, *Trustworthy Cyber-Physical Systems Engineering*, page 26. Taylor & Francis Group, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742 CRC Press, 2016.

[14] Neeraj Kumar Singh, Mark Lawford, Thomas S. E. Maibaum, and Alan Wassyng. Chapter 9 verifying trustworthy cyber-physical systems using closed-loop modeling. In Alexander Romanovsky and Fuyuki Ishikawa, editors, *Trustworthy Cyber-Physical Systems Engineering*, pages 199–236. Taylor & Francis Group, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742 CRC Press, 2016.

[15] Sasan Vakili, Neeraj Kumar Singh, Mark Lawford, Alan Wassyng, and Ben Breime. Stop and go adaptive cruise control: A case study of automotive cyber physical systems. In Alexander Romanovsky and Fuyuki Ishikawa, editors, *Trustworthy Cyber-Physical Systems Engineering*, pages 237–270. Taylor & Francis Group, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742 CRC Press, 2016.

[16] Alan Wassyng, Paul Joannou, Mark Lawford, Thomas S. E. Maibaum, and Neeraj Kumar Singh. New standards for trustworthy cyber-physical systems. In Alexander Romanovsky and Fuyuki Ishikawa, editors, *Trustworthy Cyber-Physical Systems Engineering*, pages 337–368. Taylor & Francis Group, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742 CRC Press, 2016.

[17] Dominique Méry and Neeraj Kumar Singh. Event b. In Jean-Louis Boulanger, editor, *Formal Methods Applied to Complex Systems*, chapter 10, pages 253–298. John Wiley & Sons, Ltd, 2014.

[18] Dominique Méry and Neeraj Kumar Singh. Event b. In Jean-Louis Boulanger, editor, *Mise en oeuvre de la metode b (informatique et systemes d'information, rta)*. Hermes Science Publications, 2013.

Refereed Journal Publications

[19] Ismail Mendil, Yamine Aït-Ameur, Neeraj Kumar Singh, Guillaume Dupont, Dominique Méry, and Philippe Palanque. Formal domain-driven system development in Event-B: Application to interactive critical systems. *Journal of Systems Architecture*, 135:102798, 2023.

[20] Guillaume Dupont, Yamine Aït Ameur, Neeraj Kumar Singh, and Marc Pantel. Formally verified architectural patterns of hybrid systems using proof and refinement with Event-B. *Sci. Comput. Program.*, 216:102765, 2022.

[21] Peter Reviere, Neeraj Kumar Singh, and Yamine Aït-Ameur. Reflexive Event-B: Semantics and Correctness the EB4EB Framework. *IEEE Transactions on Reliability*, pages 1–16, 2022.

[22] Neeraj Kumar Singh, Yamine Aït Ameur, Ismail Mendil, Dominique Méry, David Navarre, Philippe A. Palanque, and Marc Pantel. F3FLUID: A Formal Framework for Developing Safety-Critical Interactive Systems in FLUID. *J. Softw. Evol. Process.*, 2022.

[23] Guillaume Dupont, Yamine Aït Ameur, Neeraj Kumar Singh, and Marc Pantel. Event-B Hybridation: A Proof and Refinement-based Framework for Modelling Hybrid Systems. *ACM Trans. Embed. Comput. Syst.*, 20(4):35:1–35:37, 2021.

[24] Neeraj Kumar Singh, Yamine Aït Ameur, Romain Geniet, Dominique Méry, and Philippe A. Palanque. On the Benefits of Using MVC Pattern for Structuring Event-B Models of WIMP Interactive Applications. *Interact. Comput.*, 33(1):92–114, 2021.

[25] Neeraj Kumar Singh, Mark Lawford, Thomas Stephen Edward Maibaum, and Alan Wassyng. A formal approach to rigorous development of critical systems. *J. Softw. Evol. Process.*, 33(4), 2021.

[26] Alexandra Halchin, Yamine Aït Ameur, Neeraj Kumar Singh, Julien Ordioni, and Abderrahmane Feliachi. Handling B models in the PERF integrated verification framework: Formalised and certified embedding. *Sci. Comput. Program.*, 196:102477, 2020.

[27] Neeraj Kumar Singh. Detection of postural balance degradation using fuzzy neural network. *Int. J. Bioinform. Res. Appl.*, 15(4):371–394, 2019.

[28] Alan Wassyng, Neeraj Kumar Singh, Mischa Geven, Nicholas Proscia, Hao Wang, Mark Lawford, and Tom Maibaum. Can product-specific assurance case templates be used as medical device standards? *IEEE Des. Test*, 32(5):45–55, 2015.

[29] Manamiary Bruno Andriamiarina, Dominique Méry, and Neeraj Kumar Singh. Revisiting snapshot algorithms by refinement-based techniques. *Comput. Sci. Inf. Syst.*, 11(1):251–270, 2014.

[30] A. J. Wellings, A. Burns, A. L.C. Cavalcanti, and N. K. Singh. Programming simple reactive systems in ada: Premature program termination. *Ada Lett.*, 33(2):75–86, nov 2013.

[31] Dominique Méry and Neeraj Kumar Singh. Formal specification of medical systems by proof-based refinement. *ACM Trans. Embed. Comput. Syst.*, 12(1):15:1–15:25, 2013.

[32] Dominique Méry and Neeraj Kumar Singh. A generic framework: from modeling to code. *Innov. Syst. Softw. Eng.*, 7(4):227–235, 2011.

[33] Dominique Méry and Neeraj Kumar Singh. Functional Behavior of a Cardiac Pacing System. *International Journal of Discrete Event Control Systems (IJDECS)*, December 2010.

Refereed Conference Publications

[34] Peter Riviere, Neeraj Kumar Singh, Yamine Aït-Ameur, and Guillaume Dupont. Formalising Liveness Properties in Event-B with the Reflexive EB4EB Framework, 2023.

[35] Peter Riviere, Neeraj Kumar Singh, Yamine Aït-Ameur, and Guillaume Dupont. Standalone Event-B models analysis using the reflexive EB4EB framework, 2023.

[36] Yamine Aït Ameur, Sergiy Bogomolov, Guillaume Dupont, Neeraj Kumar Singh, and Paulius Stankaitis. Reachability analysis and simulation for hybridised Event-B models. In *Integrated Formal Methods - 17th International Conference, IFM 2022, Lugano, Switzerland, June 7-10, 2022, Proceedings*, pages 109–128, 2022.

[37] Yamine Aït Ameur, Guillaume Dupont, Ismaïl Mendil, Dominique Méry, Marc Pantel, Peter Riviere, and Neeraj Kumar Singh. Empowering the Event-B Method Using External Theories. In *Integrated Formal Methods - 17th International Conference, IFM 2022, Lugano, Switzerland, June 7-10, 2022, Proceedings*, pages 18–35, 2022.

[38] Ismaïl Mendil, Peter Reviére Yamine Aït Ameur, Neeraj Kumar Singh, Dominique Méry, and Philippe A. Palanque. Non-Intrusive Annotation-Based Domain-Specific Analysis to Certify Event-B Models Behaviours. In *29th Asia-Pacific Software Engineering Conference, APSEC 2022, Japan, December 6-9, 2022*. IEEE, 2022.

[39] Peter Riviere, Neeraj Kumar Singh, and Yamine Aït Ameur. EB4EB: A framework for reflexive event-b. In *26th International Conference on Engineering of Complex Computer Systems, ICECCS 2022, Hiroshima, Japan, March 26-30, 2022*, pages 71–80, 2022.

[40] Yamine Aït Ameur, Régine Laleau, Dominique Méry, and Neeraj Kumar Singh. Towards leveraging domain knowledge in state-based formal methods. In *Logic, Computation and Rigorous Methods - Essays Dedicated to Egon Börger on the Occasion of His 75th Birthday*, pages 1–13, 2021.

[41] Guillaume Dupont, Yamine Aït Ameur, Marc Pantel, and Neeraj Kumar Singh. Event-B Refinement for Continuous Behaviours Approximation. In Zhe Hou and Vijay Ganesh, editors, *Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18-22, 2021, Proceedings*, volume 12971 of *Lecture Notes in Computer Science*, pages 320–336. Springer, 2021.

[42] Ismaïl Mendil, Yamine Aït Ameur, Neeraj Kumar Singh, Dominique Méry, and Philippe A. Palanque. Leveraging event-b theories for handling domain knowledge in design models. In Shengchao Qin, Jim Woodcock, and Wenhui Zhang, editors, *Dependable Software Engineering. Theories, Tools, and Applications - 7th International Symposium, SETTA 2021, Beijing, China, November 25-27, 2021, Proceedings*, volume 13071 of *Lecture Notes in Computer Science*, pages 40–58. Springer, 2021.

[43] Ismaïl Mendil, Yamine Aït Ameur, Neeraj Kumar Singh, Dominique Méry, and Philippe A. Palanque. Standard Conformance-by-Construction with Event-B. In Alberto Lluch-Lafuente and Anastasia Mavridou, editors, *Formal Methods for Industrial Critical Systems - 26th International Conference, FMICS 2021, Paris, France, August 24-26, 2021, Proceedings*, volume 12863 of *Lecture Notes in Computer Science*, pages 126–146. Springer, 2021.

[44] Guillaume Dupont, Yamine Aït Ameur, Marc Pantel, and Neeraj Kumar Singh. An Event-B Based Generic Framework for Hybrid Systems Formal Modelling. In Brijesh Dongol and Elena Troubitsyna, editors, *Integrated Formal Methods - 16th International Conference, IFM 2020, Lugano, Switzerland, November 16-20, 2020, Proceedings*, volume 12546 of *Lecture Notes in Computer Science*, pages 82–102. Springer, 2020.

[45] Guillaume Dupont, Yamine Aït Ameur, Neeraj Kumar Singh, Fuyuki Ishikawa, Tsutomu Kobayashi, and Marc Pantel. Embedding Approximation in Event-B: Safe Hybrid System Design Using Proof and Refinement. In Shang-Wei Lin, Zhe Hou, and Brendan P. Mahony, editors, *Formal Methods and Software Engineering - 22nd International Conference on Formal Engineering Methods, ICFEM 2020, Singapore, Singapore, March 1-3, 2021, Proceedings*, volume 12531 of *Lecture Notes in Computer Science*, pages 251–267. Springer, 2020.

[46] Guillaume Dupont, Yamine Aït Ameur, Marc Pantel, and Neeraj Kumar Singh. Formally Verified Architecture Patterns of Hybrid Systems Using Proof and Refinement with Event-B. In *Rigorous State-Based Methods - 7th International Conference, ABZ*, volume 12071 of *LNCS*, pages 169–185. Springer, 2020.

[47] Ismaïl Mendil, Neeraj Kumar Singh, Yamine Aït Ameur, Dominique Méry, and Philippe A. Palanque. An integrated framework for the formal analysis of critical interactive systems. In *27th Asia-Pacific Software Engineering Conference, APSEC 2020, Singapore, December 1-4, 2020*, pages 139–148. IEEE, 2020.

[48] Guillaume Dupont, Yamine Aït Ameur, Marc Pantel, and Neeraj Kumar Singh. Handling Refinement of Continuous Behaviors: A Proof Based Approach with Event-B. In Dominique Méry and Shengchao Qin, editors, *2019 International Symposium on Theoretical Aspects of Software Engineering, TASE 2019, Guilin, China, July 29-31, 2019*, pages 9–16. IEEE, 2019.

[49] Alexandra Halchin, Yamine Aït Ameur, Neeraj Kumar Singh, Abderrahmane Feliachi, and Julien Ordioni. Certified embedding of B models in an integrated verification framework. In *2019 International Symposium on Theoretical Aspects of Software Engineering, TASE 2019, Guilin, China, July 29-31, 2019*, pages 168–175, 2019.

[50] Alexandra Halchin, Neeraj Kumar Singh, Yamine Aït Ameur, Julien Ordioni, and Abderrahmane Feliachi. Validation of Formal Models Transformation through Animation. In *First International Workshop on Knowledge and MOdel-driven engineering in formal development of Trustworthy Systems*, 2019.

[51] Neeraj Kumar Singh, Yamine Aït Ameur, Dominique Méry, David Navarre, Philippe A. Palanque, and Marc Pantel. Formal development of multi-purpose interactive application (MPIA) for ARINC 661. In Osman Hasan and Frédéric Mallet, editors, *Formal Techniques for Safety-Critical Systems - 7th International Workshop, FTSCS 2019, Shenzhen, China, November 9, 2019, Revised Selected Papers*, volume 1165 of *Communications in Computer and Information Science*, pages 21–39. Springer, 2019.

[52] Neeraj Kumar Singh and Hao Wang. Virtual environment model of glucose homeostasis for diabetes patients. In *IEEE International Conference on Industrial Cyber Physical Systems, ICPS 2019, Taipei, Taiwan, May 6-9, 2019*, pages 417–422, 2019.

[53] Paulius Stankaitis, Guillaume Dupont, Neeraj Kumar Singh, Yamine Aït Ameur, Alexei Iliasov, and Alexander B. Romanovsky. Modelling hybrid train speed controller using proof and refinement. In Jun Pang and Jing Sun, editors, *24th International Conference on Engineering of Complex Computer Systems, ICECCS 2019, Guangzhou, China, November 10-13, 2019*, pages 107–113. IEEE, 2019.

[54] Yamine Aït Ameur, Idir Aït-Sadoune, P. Casteran, J. Paul Gibson, Kahina Hacid, Souad Kherroubi, Dominique Méry, Linda Mohand-Oussaïd, Neeraj Kumar Singh, and Laurent Voisin. On the importance of explicit domain modelling in refinement-based modelling design. experiments with Event-B. In Michael J. Butler, Alexander Raschke, Thai Son Hoang, and Klaus Reichl, editors, *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 6th International Conference, ABZ 2018, Southampton, UK, June 5-8, 2018, Proceedings*, volume 10817 of *Lecture Notes in Computer Science*, pages 425–430. Springer, 2018.

[55] Guillaume Dupont, Yamine Aït Ameur, Marc Pantel, and Neeraj Kumar Singh. Hybrid systems and Event-B: A formal approach to signalised left-turn assist. In *New Trends in Model and Data Engineering - MEDI 2018 International Workshops, DETECT, MEDI4SG, IWCFS, REMEDY, Marrakesh, Morocco, October 24-26, 2018, Proceedings*, pages 153–158, 2018.

[56] Guillaume Dupont, Yamine Aït Ameur, Marc Pantel, and Neeraj Kumar Singh. Proof-Based Approach to Hybrid Systems Development: Dynamic Logic and Event-B. In Michael J. Butler, Alexander Raschke, Thai Son Hoang, and Klaus Reichl, editors, *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 6th International Conference, ABZ 2018, Southampton, UK, June 5-8, 2018, Proceedings*, volume 10817 of *Lecture Notes in Computer Science*, pages 155–170. Springer, 2018.

[57] Guillaume Dupont, Yamine Ait Ameur, Neeraj Kumar Singh, and Marc Pantel. On the use of the theory plug-in to define theories for differential equations. In *7th Rodin Workshop, Co-located with ABZ 2018, , UK, June 5*, 2018.

[58] Romain Geniet and Neeraj Kumar Singh. Refinement based formal development of human-machine interface. In Manuel Mazzara, Iulian Ober, and Gwen Salaün, editors, *Software Technologies: Applications and Foundations - STAF 2018 Collocated Workshops, Toulouse, France, June 25-29, 2018, Revised Selected Papers*, volume 11176 of *Lecture Notes in Computer Science*, pages 240–256. Springer, 2018.

[59] Neeraj Kumar Singh, Yamine Aït Ameur, and Dominique Méry. Formal ontology driven model refactoring. In *23rd International Conference on Engineering of Complex Computer Systems, ICECCS 2018, Melbourne, Australia, December 12-14, 2018*, pages 136–145, 2018.

[60] Alexandra Halchin, Abderrahmane Feliachi, Neeraj Kumar Singh, Yamine Aït Ameur, and Julien Ordioni. B-perfect - applying the PERF approach to B based system developments. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification - Second International Conference, RSSRail 2017, Pistoia, Italy, November 14-16, 2017, Proceedings*, pages 160–172, 2017.

[61] Neeraj Kumar Singh, Mark Lawford, T. S. E. Maibaum, and Alan Wassyng. Use of tabular expressions for refinement automation. In Yassine Ouhammou, Mirjana Ivanovic, Alberto Abelló, and Ladjel Bellatreche, editors,

Model and Data Engineering - 7th International Conference, MEDI 2017, Barcelona, Spain, October 4-6, 2017, Proceedings, volume 10563 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2017.

[62] Guillaume Babin, Yamine Aït Ameur, Neeraj Kumar Singh, and Marc Pantel. A System Substitution Mechanism for Hybrid Systems in Event-B. In Kazuhiro Ogata, Mark Lawford, and Shaoying Liu, editors, *Formal Methods and Software Engineering - 18th International Conference on Formal Engineering Methods, ICFEM 2016, Tokyo, Japan, November 14-18, 2016, Proceedings*, volume 10009 of *Lecture Notes in Computer Science*, pages 106–121, 2016.

[63] Guillaume Babin, Yamine Aït Ameur, Neeraj Kumar Singh, and Marc Pantel. Handling continuous functions in hybrid systems reconfigurations: A formal Event-B development. In Michael J. Butler, Klaus-Dieter Schewe, Atif Mashkoor, and Miklós Biró, editors, *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 5th International Conference, ABZ 2016, Linz, Austria, May 23-27, 2016, Proceedings*, volume 9675 of *Lecture Notes in Computer Science*, pages 290–296. Springer, 2016.

[64] Sarah Benyagoub, Meriem Ouederni, Neeraj Kumar Singh, and Yamine Aït Ameur. Correct-by-construction evolution of realisable conversation protocols. In Ladjel Bellatreche, Oscar Pastor, Jesús Manuel Almendros-Jiménez, and Yamine Aït Ameur, editors, *Model and Data Engineering - 6th International Conference, MEDI 2016, Almería, Spain, September 21-23, 2016, Proceedings*, volume 9893 of *Lecture Notes in Computer Science*, pages 260–273. Springer, 2016.

[65] Neeraj Kumar Singh. A Virtual Glucose Homeostasis Model for Verification, Simulation and Clinical Trials. In *Workshop 2016: Cyber Security and Functional Safety in Cyber-Physical Systems*. EuroAsiaSPI'2016, 2016.

[66] Neeraj Kumar Singh. Detection of hesitant dynamic postural control. In *2016 IEEE 12th International Colloquium on Signal Processing & Its Applications (CSPA)*, pages 36–40, 2016.

[67] Neeraj Kumar Singh, Yamine Aït Ameur, Marc Pantel, Arnaud Dieumegard, and Eric Jenn. Stepwise formal modeling and verification of self-adaptive systems with event-b. the automatic rover protection case study. In Hai Wang and Mounir Mokhtari, editors, *21st International Conference on Engineering of Complex Computer Systems, ICECCS 2016, Dubai, United Arab Emirates, November 6-8, 2016*, pages 43–52. IEEE Computer Society, 2016.

[68] Valentin Cassano, Silviya Grigorova, Neeraj Kumar Singh, Morayo Adedjouma, Mark Lawford, T. S. E. Maibaum, and Alan Wassyng. Is current incremental safety assurance sound? In *Computer Safety, Reliability, and Security - SAFECOMP 2015 Workshops, ASSURE, DECSoS, ISSE, ReSA4CI, and SASSUR, Delft, The Netherlands, September 22, 2015, Proceedings*, pages 397–408, 2015.

[69] Dominique Méry and Neeraj Kumar Singh. Analyzing requirements using environment modelling. In *17th International Conference on Human-Computer Interaction (HCI 2015)*, Lecture Notes in Computer Science. Springer International Publishing, 2015.

[70] Neeraj Kumar Singh, Mark Lawford, T. S. E. Maibaum, and Alan Wassyng. Formalizing the cardiac pacemaker resynchronization therapy. In Vincent G. Duffy, editor, *Digital Human Modeling - Applications in Health, Safety, Ergonomics and Risk Management: Ergonomics and Health - 6th International Conference, DHM 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015, Proceedings, Part II*, volume 9185 of *Lecture Notes in Computer Science*, pages 374–386. Springer, 2015.

[71] Neeraj Kumar Singh, Mark Lawford, Thomas Stephen Edward Maibaum, and Alan Wassyng. Stateflow to tabular expressions. In Huynh Quyet Thang, Le Anh Phuong, Luc De Raedt, Yves Deville, Marc Bui, Truong Thi Dieu Linh, Nguyen Thi-Oanh, Dinh Viet Sang, and Nguyen Ba Ngoc, editors, *Proceedings of the Sixth International Symposium on Information and Communication Technology, Hue City, Vietnam, December 3-4, 2015*, pages 312–319. ACM, 2015.

[72] Neeraj Kumar Singh, Hao Wang, Mark Lawford, Thomas Stephen Edward Maibaum, and Alan Wassyng. Stepwise formal modelling and reasoning of insulin infusion pump requirements. In Vincent G. Duffy, editor, *Digital Human Modeling - Applications in Health, Safety, Ergonomics and Risk Management: Ergonomics and Health - 6th International Conference, DHM 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015, Proceedings, Part II*, volume 9185 of *Lecture Notes in Computer Science*, pages 387–398. Springer, 2015.

[73] Manamiary Bruno Andriamiarina, Dominique Méry, and Neeraj Kumar Singh. Analysis of self-$\star$ and P2P systems using refinement. In Yamine Aït Ameur and Klaus-Dieter Schewe, editors, *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 4th International Conference, ABZ 2014, Toulouse, France, June 2-6, 2014. Proceedings*, volume 8477 of *Lecture Notes in Computer Science*, pages 117–123. Springer, 2014.

[74] Dominique Méry and Neeraj Kumar Singh. Formal evaluation of landing gear system. In *Proceedings of the Fifth Symposium on Information and Communication Technology, SoICT '14, Hanoi, Vietnam, December 4-5, 2014*, pages 75–84, 2014.

[75] Dominique Méry and Neeraj Kumar Singh. Modeling an aircraft landing system in Event-B. In *ABZ 2014: The Landing Gear Case Study - Case Study Track, Held at the 4th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z, Toulouse, France, June 2-6, 2014. Proceedings*, pages 154–159, 2014.

[76] Dominique Méry and Neeraj Kumar Singh. The semantics of refinement chart. In Vincent G. Duffy, editor, *Digital Human Modeling. Applications in Health, Safety, Ergonomics and Risk Management - 5th International Conference, DHM 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014. Proceedings*, volume 8529 of *Lecture Notes in Computer Science*, pages 415–426. Springer, 2014.

[77] Neeraj Kumar Singh, Hao Wang, Mark Lawford, T. S. E. Maibaum, and Alan Wassyng. Formalizing the glucose homeostasis mechanism. In *Digital Human Modeling. Applications in Health, Safety, Ergonomics and Risk Management - 5th International Conference, DHM 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014. Proceedings*, pages 460–471, 2014.

[78] Manamiary Bruno Andriamiarina, Dominique Méry, and Neeraj Kumar Singh. Integrating proved state-based models for constructing correct distributed algorithms. In Einar Broch Johnsen and Luigia Petre, editors, *Integrated Formal Methods, 10th International Conference, IFM 2013, Turku, Finland, June 10-14, 2013. Proceedings*, volume 7940 of *Lecture Notes in Computer Science*, pages 268–284. Springer, 2013.

[79] Dominique Méry and Neeraj Kumar Singh. Ideal mode selection of a cardiac pacing system. In Vincent G. Duffy, editor, *Digital Human Modeling and Applications in Health, Safety, Ergonomics, and Risk Management. Healthcare and Safety of the Environment and Transport - 4th International Conference, DHM 2013, Held as Part of HCI International 2013, Las Vegas, NV, USA, July 21-26, 2013, Proceedings, Part I*, volume 8025 of *Lecture Notes in Computer Science*, pages 258–267. Springer, 2013.

[80] Manoranjan Satpathy, S. Ramesh, Colin F. Snook, Neeraj Kumar Singh, and Michael J. Butler. A mixed approach to rigorous development of control designs. In *2013 IEEE International Symposium on Computer-Aided Control System Design, CACSD 2013, Hyderabad, India, August 28-30, 2013*, pages 7–12, 2013.

[81] Manamiary Bruno Andriamiarina, Dominique Méry, and Neeraj Kumar Singh. Revisiting snapshot algorithms by refinement-based techniques. In Hong Shen, Yingpeng Sang, Yidong Li, Depei Qian, and Albert Y. Zomaya, editors, *13th International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2012, Beijing, China, December 14-16, 2012*, pages 343–349. IEEE, 2012.

[82] Dominique Méry and Neeraj Kumar Singh. Closed-loop modeling of cardiac pacemaker and heart. In Jens H. Weber and Isabelle Perseil, editors, *Foundations of Health Information Engineering and Systems - Second International Symposium, FHIES 2012, Paris, France, August 27-28, 2012. Revised Selected Papers*, volume 7789 of *Lecture Notes in Computer Science*, pages 151–166. Springer, 2012.

[83] Dominique Méry and Neeraj Kumar Singh. Critical systems development methodology using formal techniques. In *Symposium on Information and Communication Technology 2012, SoICT '12, Halong City, Quang Ninh , Viet Nam, August 23-24, 2012*, pages 3–12, 2012.

[84] Neeraj Kumar Singh, Andy J. Wellings, and Ana Cavalcanti. The cardiac pacemaker case study and its implementation in safety-critical java and ravenscar ada. In *The 10th International Workshop on Java Technologies for Real-time and Embedded Systems, JTRES '12, Copenhagen, Denmark, October 24-26, 2012*, pages 62–71, 2012.

[85] 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, 2011)*, Beijing, China, November 2011.

[86] Dominique Méry and Neeraj Kumar Singh. EB2J : Code Generation from Event-B to Java. In *SBMF - Brazilian Symposium on Formal Methods*, São Paulo, Brazil, September 2011. CBSoft - Brazilian Conference on Software: Theory and Practice.

[87] Dominique Méry and Neeraj Kumar Singh. Analysis of DSR protocol in Event-B. In Xavier Défago, Franck Petit, and Vincent Villain, editors, *Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings*, volume 6976 of *Lecture Notes in Computer Science*, pages 401–415. Springer, 2011.

[88] 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.

[89] Dominique Méry and Neeraj Kumar Singh. Formalization of heart models based on the conduction of electrical impulses and cellular automata. In *Foundations of Health Informatics Engineering and Systems - First International Symposium, FHIES 2011, Johannesburg, South Africa, August 29-30, 2011. Revised Selected Papers*, pages 140–159, 2011.

[90] Dominique Méry and Neeraj Kumar Singh. Medical protocol diagnosis using formal methods. In *Foundations of Health Informatics Engineering and Systems - First International Symposium, FHIES 2011, Johannesburg, South Africa, August 29-30, 2011. Revised Selected Papers*, pages 1–20, 2011.

[91] Dominique Méry and Neeraj Kumar Singh. EB2C : A Tool for Event-B to C Conversion Support. 8th IEEE International Conference on Software Engineering and Formal Methods (SEFM), September 2010. Poster and Tool Demo submission, and published in a CNR Technical Report.

[92] David Hewson, Neeraj Kumar Singh, Hichem Snoussi, and Jacques Duchene. Classification of elderly as fallers and non-fallers using Centre of Pressure velocity. In *2010 32nd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC 2010)*, pages 3678–3681, Buenos Aires, Argentina, August 2010. IEEE.

[93] 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.

[94] 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.

[95] Neeraj Kumar Singh, Hichem Snoussi, David J. Hewson, and Jacques Duchêne. Detection of the critical point interval of postural control strategy using wavelet transform analysis. In Pedro Encarnação and António P. Veloso, editors, *BIOSIGNALS 2009 - Proceedings of the International Conference on Bio-inspired Systems and Signal Processing, Porto, Portugal, January 14-17, 2009*, pages 101–106. INSTICC Press, 2009.

[96] Neeraj Kumar Singh, Hichem Snoussi, David J. Hewson, and Jacques Duchêne. Wavelet transform analysis of the power spectrum of centre of pressure signals to detect the critical point interval of postural control. In Ana L. N. Fred, Joaquim Filipe, and Hugo Gamboa, editors, *Biomedical Engineering Systems and Technologies - International Joint Conference, BIOSTEC 2009 Porto, Portugal, January 14-17, 2009, Revised Selected Papers*, volume 52 of *Communications in Computer and Information Science*, pages 235–244, 2009.

[97] V. Singh, S. K. Yadav, N. K. Singh, and P. K. Kalra. Color image compression using block-based independent component analysis. In *2007 5th International Symposium on Image and Signal Processing and Analysis*, pages 288–292, 2007.

Thesis

[98] Neeraj Kumar Singh. *Reliability and Safety of Critical Device Software Systems*. Theses, Université Henri Poincaré - Nancy 1, November 2011.

[99] Neeraj Kumar Singh. Use of statistical mechanics methods to assess the effects of sensory perturbation and aging on stability during upright stance & classification and detection of an increased risk of falling in elderly a fuzzy neural network (fnn). Master's thesis, University of Technology of Troyes, Troyes, France, 2008.

[100] Neeraj Kumar Singh. Modeling, classification and fault detection of sensors using artificial intelligence & blind source separation using ica algorithms. Master's thesis, Uttar Pradesh Technical University, Lucknow, India, 2006.

Technical Reports

[101] Neeraj Kumar Singh, Mark Lawford, Thomas S. E. Maibaum, and Alan Wassyng. Report 19: Formal evaluation of the cardiac pacemaker resynchronization therapy. https://www.mcscert.ca/index.php/documents/mcscert-reports Technical Report 19, McSCert, McMaster University, October 2014.

[102] Neeraj Kumar Singh, Hao Wang, Mark Lawford, Thomas S. E. Maibaum, and Alan Wassyng. Report 18: Formalizing Insulin Pump using Event-B. https://www.mcscert.ca/index.php/documents/mcscert-reports Technical Report 18, McSCert, McMaster University, October 2014.

[103] Dominique Méry and Neeraj Kumar Singh. Modelling an aircraft landing system in Event-B (full report). *CoRR*, abs/1407.0927, 2014.

[104] 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, May 2011.

[105] 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.

[106] 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.

[107] 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.

APPENDIX B

CV and summary of professional and scientific activities

Education and experience

I earned my MCA (Master in Computer Application) degree and then worked as research associate at Indian Institute of Technology Kanpur in India, where I worked on a range of diverse projects funded by the MCIT (Ministry of Communications and Information Technology) and ISRO(Indian Space Research Organisation). In 2007, I received a scholarship to study in France for a Master's degree at University of Technology of Troyes. Later, in 2008, I began my PhD. In 2011, I obtained my Ph.D. in Computer Science from INRIA Nancy Grand Est, Henri Poincaré University, Nancy 1 (now the Université de Lorraine), France, under supervision of Prof. Dominique Mery. From January 2012 to July 2013, I worked as a postdoctoral researcher at the University of York, Computer Science Department, under the supervision of Prof. Ana Cavalcanti and Prof. Andy Wellings. I also worked as a postdoctoral researcher at the McMaster Centre for Software Certification (McSCert), the department Computing and Software, McMaster University, Canada from August 2013 to August 2015, under the supervision of Prof. Tom Maibaum, Prof. Mark Lawford and Prof. Alan Wassyng.

Current position

Since 2015, I have been a Maître de Conférence (equivalent to Associate Professor) in the Science and Numeric department at Toulouse INPT-ENSEEIHT, and I have a joint appointment with the Toulouse Institute for Computer Science Research (IRIT) to conduct research in the ACADIE team. My primary professional activities have been as follows:

International and national collaborations

My research aims to design and implement safe and secure safety-critical cyber-physical systems using software engineering principles and techniques as well as formal methods. In this direction, I have three broad research areas: (1) modelling, refinement, and proofs for state-based methods, (2) software and system certification, and (3) domain knowledge engineering and formal meta-modelling. These research topics are well-suited to the context of the ACADIE team as well as the rest of the department's activities (my work has connections with security, data science, embedded systems, networking, privacy, and cloud computing). Furthermore, my work is well connected on a national and international scale. Currently, I am collaborating with Prof. Yamine Ait-Ameur, Dr. Marc Pantel and Dr. Guillaume Dupont (INPT-ENSEEIHT, France) on proof and refinement for state based methods, Prof. Dominique Mery (University of Lorraine, France) on distributed system verification, Prof. Alan Wassyng, Prof. Mark Lawford and Prof. Tom Maibaum (McMaster University, Canada) on software certification and development of safety cases, Dr. Sergiy Bogomolov and Dr. Paulius Stankaitis (Newcastle University, UK) on cyber-physical systems, Dr. Raju Halder (Indian Institute of Technology Patna, India) on block chain technology, Dr. Rajiv Pandey (Amity University, India) and Dr. Kanishka Tayagi (Aptiv Technology, USA) on the edge of computing and autonomous driving, Prof. Philippe Palanque (Université Paul Sabatier - Toulouse III, France) on the development of safe interactive systems based on correct by construction approach, Dr. S Ramesh (General Motors, USA) and Dr. Manoranjan Satpathy (Indian Institute of Technology Bhubaneswar, India) on designing, modelling and proof of Simulink models using correct by construction, and Dr. Hao Wang (NTNU, Norway) on environment modelling and its implementation and simulation.

As part of my efforts to establish major national and international collaborators, as well as as an independent researcher, I have hired and advised 10 PhD students and 10 master and under graduate students (see the page 142-144 of my detailed CV for a list and the dedicated section below for details).

Publications

My research findings have been disseminated through publications and invited talks (see pages 144-155 of my detailed CV). The scientific quality of my research has been recognized by various invitations of scientific nature (invited tutorials, invited talks, participation to program committees, book authored, book editors, etc.). Currently, I have published 1 book (authored), 2 books (co-edited), 9 book chapters, 15 journals (peer-reviewed), and 67 conferences (peer-reviewed). The book is the result of thesis work published by Springer; one book has been co-edited for Springer; one proceeding has been co-edited; and book chapters have been contributed as part of a handbook and collection of articles. I have received the *best paper awards* for one journal paper (TECS 2023 - ACM Transaction on Embedded Computing System Award 2023), and two conference papers (ICEFEM 2021 and BIOSIGNAL 2009). The majority of the 15 journals are published in prestigious formal method venues, such as ACM Transactions on Embedded Computing Systems (TECS), Science of Computer Programming (SCP), IEEE Design & Test, NASA Journal, Interacting with Computers (IwC) and others. Conference papers are published in top conferences, such as ATVA, ICFEM, IFM, ICECCS, APSEC, ABZ, SAFECOMP, as well as in other conferences dedicated to formal methods community. See the full list of my professional service and scientific responsibilities on pages 156-160 of my detailed CV.

Impact

My work has had a significant impact on the industrial sector, for example: (1) An industrialisation of B2HLL prototype tool by RATP (Régie autonome des transports parisiens); (2) In 2013, I was hired by the Cambridge consultant, Oxford, UK, to assist a company with automatic code generation using my developed tool EB2ALL; and (3) In 2007, I worked on a data-mining tool used by ISRO (Indian Space Research Organisation). Page 16 of my detailed CV contains a complete list of my tooling development and responsibilities.

In addition, I am in charge of improving the developed tool EB2ALL in order to maintain and release the next version of it.

Seeking funding

Since 2015, I have been actively seeking funding at both the national and international levels. So far, I have co-written four ANR projects (ICSPA (2022-2026), EBRP (2019-2023), DISCONT (2018-2023), FORMEDICIS (2017-2022)) in collaboration of other faculty and university partners, as well as two funded COMET projects (IntegR (2018-2020), Str@se(2020-2022)). It should be noted that I am a Co-Principal Investigator or Co-Investigator on all funded projects. Furthermore, I have received two CIFRE Ph.D grant funded by RATP and one university PhD grant funded by ED MITT, as detailed on pages 138-141 of my detailed CV.

Teaching and advising

I had the privilege of teaching both undergraduate and graduate students over the ten years. I have taught courses ranging from the introductory to specialised courses in computer science and engineering. In each of these courses, I had a great deal of autonomy to give lectures, preparing tutorials and practical exercises, including projects, exams, and grading papers.

I have eight years of teaching experience assisting in undergraduate and graduate courses at the INPT-ENSEEIHT in Toulouse, France. For first-year students (195 students majoring in computer science and Telecommunication & network), I teach N5EN03B - Imperative Programming in Ada, N5EN05B - Automatic (Cyber Physical System), N6EN05A - Object-Oriented Technology in Java, N5AN02A - Programming Methodology, and N5EN03C - C Language. I teach N8EN10A - Compilers and Formal Semantics to second-year students (60 students majoring in computer science and software engineering), and N9EN12B - Formal System Development in Event-B to third-year students (60 students majoring in computer science and software engineering). In addition, I teach master students (60 students) a course on N9EN25C - Safety-Critical Embedded Systems. In each of these courses, I had a great deal of autonomy to give lectures, preparing tutorials (TD - travaux dirigś (TD)) and practical exercises (TP - travaux pratiques), including projects, exams, and grading papers.

Teaching and internal responsibilities

- Advisory committee member of the SN department council (since 2023)
- In charge of third-year software engineering students (since 2023)
- Advisory committee member of the apprenti (work-study) students (since 2019)
- Teaching tutor for three apprenti (work-study) students for every year (since 2016)
- Course coordinator of the following courses: N9EN12B - Formal System Development in Event-B; N5EN05B - *Automatic (Cyber Physical System)*; and N5AN02A - *Programming Methodology* (since 2016)
- Participated for developing the following undergraduate courses: *"Programation Imperative"* and *"Automatique"*
- Participated for developing the course of Master degree "Performance in Software, Media and Scientific Computing (PSMSC)"
- Involve as a jury member for undergraduate students (since 2016)
- Involve in defense committee for more than 50 master undergraduate students (since 2016)

Since my start at ENSEEIHT, I have advised students at all levels (see also list in pages 6-7 of my detailed CV):

Ph.D Student Advising
In Progress

- Yanins Benabbi(Jan. 2023 - Present), INPT-ENSEEIHT/IRIT, Toulouse, France: *"Formal verification and validation of railway systems using PERF"* - Funding CIFRE Grant with RATP - Co-supervision with Yamine Aït Ameur.
- Abhishek Kumar Saxena (Jul. 2021 - Present), Amity University, Lucknow, India: *"Latency Optimization of edge learning through dimension reduction"* - Funding PhD Scholarships - Co-supervision with Rajiv Pandey.

Completed

- Guillaume Dupont (Oct. 2017 - Apr. 2021), INPT-ENSEEIHT/IRIT, Toulouse, France: *"Correct-by-Construction Design of Hybrid Systems Based on Refinement and Proof"* - Funding ANR-DISCONT Project - Co-supervision with Yamine Aït Ameur and Marc Pantel - Working as Maître de Conférences at INP-ENSEEIHT/IRIT, Toulouse, France.
- Alexandra Halchin (Oct. 2016 - Dec. 2021), INPT-ENSEEIHT/IRIT, Toulouse, France: *"Development of a Formal Verification Methodology for B specifications using PERF formal toolkit"* - Funding CIFRE Grant with RATP - Co-supervision with Yamine Aït Ameur, Abderrahmane Feliachi, and Julien Ordioni - Working as Research Engineer at RATP (Régie autonome des transports parisiens).
- Ismail Mendil (Dec. 2019 - Oct. 2023), INPT-ENSEEIHT/IRIT, Toulouse, France: *"A Framework for Explicit Modelling of Domain Knowledge in State-Based Formal Methods: the Case of Interactive Critical Systems"* - Funding ANR-FORMEDICIS Project - Co-supervision with Yamine Aït Ameur, Philippe Palanque, and Dominique Mery - Working as Research Engineer at Huawei.
- Peter Riviere (Oct. 2020 - Jun. 2024), INPT-ENSEEIHT/IRIT, Toulouse, France: *"Automatic generation of proof obligations parameterised by domain theories implementation in Event-B: The EB4EB Framework"* - Funding EDMITT Grant (French doctoral scholarship) - Co-supervision with Yamine Aït Ameur.

Masters & Undergraduate Advising

- Clement Torti, undergraduate final project, “A Framework for Automating Testing Process", 2023, INP-ENSEEIHT/IRIT
- Mathieu Teissedre, undergraduate final project, “Input-Output Simulators for Aeronautical Bus", 2022, INP-ENSEEIHT/IRIT
- Mickael Dalbin, undergraduate final project, “WebApp: SSO Authentication and Permissions Management", 2021, INP-ENSEEIHT/IRIT
- Thomas Salinas Broutee, undergraduate final project, “Development of VoIP analysis tools", 2020, INP-ENSEEIHT/IRIT
- Hasnae Dada, undergraduate project, “Cloud Services Integration Optimisation", 2017, INP-ENSEEIHT/IRIT
- Romain Geniet, M.Sc, “Verification of critical interactive systems through refinement and proof using Event-B", 2016, INP-ENSEEIHT/IRIT, co-advised with Yamine Ait Ameur
- Mohamed Anas Charafi, undergraduate project, “Transformation of B model to HLL", 2016, INP-ENSEEIHT/IRIT
- Yanjun Jiang, M.A.Sc,“A Tabular Expression to Event-B Language Transformation Tool", 2015, McMaster University, co-advised with Tom Maibaum
- Sasan Vakili, M.A.Sc., “Design and Formal Verification of an Adaptive Cruise Control Plus (ACC+) System", 2015, McMaster University, co-advised with Mark Lawford
- Mischa Geven, M.A.Sc., 2014, McMaster University, co-advised with Alan Wassyng and Mark Lawford
- Nicholas Proscia, M.A.Sc., 2014, McMaster University, co-advised Alan Wassyng and Mark Lawford

Neeraj Kumar Singh

CONTACT INFORMATION

F-301, INP-ENSEEIHT/IRIT
2 Rue Charles Camichel, 31000
Toulouse, France

Mobile: +33(0)768777292
Off. Phone: +33(0)534322182
E-mail: neeraj.singh@toulouse-inp.fr
iit.neeraj@gmail.com

Web Page: `https://sites.google.com/site/singhnne`

RESEARCH INTERESTS

- Logic in computer science, refinement and proofs, programming languages & formal methods
- Formal verification of safety-critical cyber-physical systems, human machine interfaces (HMI), and medical protocols
- Development life-cycle and software certification
- Domain engineering and environment modelling
- Simulation and automatic code generation

EDUCATION

- **Ph.D in Computer Science, Université Henri Poincaré Nancy 1, France (2008-2011)**
  Title: *"Reliability and Safety of Critical Device Software Systems"*
  Advisor: Prof. Dominique Méry, LORIA, University of Lorraine, Nancy, France
  Referee: Prof. Yamine Ait-Ameur, INP-ENSEEIHT/IRIT, Toulouse, France
  Referee: Prof. John Fitzgerald, Newcastle University, UK

  Thesis is published by Springer. Available at : `http://www.springer.com/computer/theoretical+computer+science/book/978-1-4471-5259-0`.
- **M.S in Optimization of System and Security (OSS), Université de technologie de Troyes, France (2007-2008)**
  Title: *"Use of statistical mechanics methods to assess the effects of sensory perturbation and aging on stability during upright stance & Classification and detection of an increased risk of falling in elderly a Fuzzy Neural Network (FNN)"*
  Adviser: Prof. David Hewson
- **MCA (Master of Computer Applications) in Computer Science with Distinction, Uttar Pradesh Technical University, India (2003-2006)**
  Title: *"Modeling, Classification and Fault Detection of Sensors using Artificial Intelligence & Blind Source Separation using ICA Algorithms"*.
  Advisor: Prof. P. K. Kalra, IIT Kanpur, India.
- **B.Sc in Computer Science, Lucknow University, India (2000-2003)**

RESEARCH EXPERIENCE

- **Associate Professor** **September, 2015 - Present**
  Department of Science and Numeric, INP-ENSEEIHT/IRIT, Toulouse, France
- **Postdoctoral Researcher** **August, 2013 - August, 2015**
  Department of Computing and Software, McMaster University, Canada
- **Research Associate** **January, 2012 - July, 2013**
  Department of Computer Science, University of York, UK
- **Graduate Researcher** **October, 2008 - December, 2011**
  INRIA Nancy - Grand Est Research Centre, LORIA, France

- **Research Assistant** **August, 2007 - September, 2008**
  System Modelling and Dependability Laboratory, University of Technology of Troyes, France
- **Project Associate** **January, 2006 - July, 2007**
  Electrical Engineering Department, Indian Institute of Technology, Kanpur, India.
- **Senior Instructor** **August, 2000 - June, 2003**
  Center of Development of Advanced Computing(C-DAC), Lucknow, India

---

Professional Experience

- **Industrial Intern** **July, 2010 - September, 2010**
  General Motor, India Science Lab, Bangalore, India
- **Senior Programmer** **August, 2000 - June, 2003**
  Center of Development of Advanced Computing(C-DAC), Lucknow, India

---

Awards and Honours

1. Received **PEDR(Prime d'Encadrement Doctoral et de Recherche)** in 2019 (4+ years grant).
2. **Best Paper Awards** at,
   - **TECS 2023** - ACM Transaction on Embedded Computing System Award 2023
   - **ICFEM 2021** - International Conference on Formal Engineering Methods 2021
   - **BIOSIGNALS 2009** - International Conference on Bio-inspired Systems and Signal Processing 2009
3. Received **IDEX**[1] University of Toulouse Programme "Nouveaux Entrants" grant, 2015.
4. Received an **Erasmus teaching grant** to teach a two-week course on Dependable Software System (DESEM) for M.Sc students in Nancy, France, 2015.
5. Postdoctoral Research Fellowship, McMaster University, Canada, August, 2013 for three years.
6. Postdoctoral Research Associate Fellowship, University of York, UK, from January, 2012 for three years.
7. Received Qualification from the French Ministry of Research and Education (Conseil National des Universités).
8. Awarded French Ministry Scholarship, LORIA, University of Henri Poincare Nancy 1, Nancy, France for PhD study, 2008.
9. Awarded Scholarship from the PARAChute research project, University of Technology of Troyes, France for Master study, 2007.
10. Appreciation Certificate for excellent services in short term course on Application of Matlab in Engineering, Indian Institute of Technology Kanpur, India, 2006

Projects and Fundings

Current Research Grants

- **Interoperable and Confident Set-based Proof Assistants (ANR-ICSPA)**

[1] IDEX - Initiative d'excellence (Excellence Initiative)

Sponsor : French National Research Agency (ANR)
Programe : PRCE
Role : Co-Investigator
Duration : 2022-2026
Total amount : EUR 665,600
Share : EUR 89,152
Students : 1 PhD (hiring)
Publications : Not yet
Software and Models : Models
Website : `http://icspa.inria.fr/`

- **Enhancing EventB and RODIN: EventB-RODIN-Plus (ANR-EBRP)**
  Sponsor : French National Research Agency (ANR)
  Programe : PRC
  Role : Co-Investigator
  Duration : 2019-2023
  Total amount : EUR 651,400
  Share : EUR 329,400
  Students : 1 PhD (Peter Riviere)
  Publications : [J2, C1, C2, C7, C8]
  Software and Models : Context instantiation plugins and Theory plugins for Rodin
  Website : `https://www.irit.fr/EBRP`

- **Correct Integration of Discrete and Continuous model (ANR-DISCONT)**
  Sponsor : French National Research Agency (ANR)
  Programe : PRC
  Role : Co-Principal Investigator
  Duration : 2018-2023
  Total amount : EUR 814,677
  Share : EUR 163,080
  Students : 1 PhD (Guillaume Dupont)
  Publications : [J4, J7, C6, C11, C14, C15, C16, C18, C20, C24, C26, C27]
  Software and Models : Development patterns, theories, models and case studies
  Website : `https://discont.loria.fr`

- **Formal verification and validation of railway systems using PERF**
  Title : Formal verification and validation of railway systems using PERF
  Sponsor : RATP (Régie autonome des transports parisiens)
  Programe : CIFRE
  Role : Co-Principal Investigator
  Duration : 2022-2025
  Total amount : 60K EUR + Ph.D scholarship
  Students : 1 PhD (Yannis Benabbi)
  Publications : Not yet
  Software and Models : Not yet
  Website : `https://www.ratp.fr`

- **Reflexive Event-B**

Sponsor : EDMITT (Ecole Doctorale Mathématiques, Informatique, Télécommunications de Toulouse)
Programe : French doctoral scholarships
Role : Principal Investigator
Duration : 2020-2023
Total amount : EUR 108,000 (3 x 12 x 3000)
Share : EUR 108,000
Students : 1 PhD (Peter Riviere)
Publications : [J2, C1, C2, C7, C8]
Software and Models : Reflective Event-B, Meta theories for Event-B
Website : `https://ed-mitt.univ-toulouse.fr/as/ed/edmitt/page.pl`

Past Research Grants

- **Formal Methods for the Development and the Engineering of Critical Interactive Systems (FORMEDICIS)**
  Sponsor : French National Research
  Programe : PRC
  Role : Co-Investigator
  Duration : 2017-2022
  Total amount : EUR 943,337
  Share : EUR 155,520
  Students : 1 PhD (Ismail Mendil) + 1 Master (Romain Geniet)
  Publications : [J1, J3, J5, C5, C9, C10, C12, C13, C17, C21, C25]
  Software and Models : Methodologies, domain theories, models and industrial case studies
  Website : `https://forge.onera.fr/projects/formedicis`

- **Strategic Software Engineering & Tools (Str@se)**
  Sponsor : SCCH (Software Competence Center Hagenberg)
  Programe : COMET project (following the IntegR COMET project)
  Role : Co-Investigator
  Duration : 2020-2022
  Total amount : EUR 50,000
  Share : EUR 50,000
  Students : 1 PhD (Nassima Djema)
  Software and Models : Models and case studies
  Website : `https://www.scch.at/scch/comet/comet-program`

- **Elaboration d'une méthodologie de vérification formelle de spécifications B dans l'atelier de preuve PERF**
  Sponsor : RATP (Régie autonome des transports parisiens)
  Programe : CIFRE
  Role : Co-Principal Investigator
  Duration : 2016-2020
  Total amount : 40K EUR + Ph.D scholarship
  Students : 1 PhD (Alexandra Halchin)
  Publications : [J8, C19, C22, C29]
  Software and Models : B2HLL tool, certification framework, models, and case studies
  Website : `https://www.ratp.fr`

- **Integration of Rigorous Methods & Tools (IntegR)**

Sponsor : SCCH (Software Competence Center Hagenberg)
Programe : COMET project
Role : Co-Investigator
Duration : 2018-2020
Total amount : EUR 48,000
Share : EUR 48,000
Students : 1 PhD (Sarah Benyagoub)
Publications : [C33]
Software and Models : Models and case studies
Website : `https://www.scch.at/scch/comet/comet-program`

- **Implicit and Explicit Semantics Integration in Proof-based Developments of Discrete Systems (ANR-IMPEX)**
  Sponsor : French National Research (ANR)
  Programe : PRC
  Role : Participant
  Duration : 2015-2017
  Publications : [B3, C12, C23, C34]
  Software and Models : Models and case studies
  Website : `http://impex.gforge.inria.fr`

- **The INGEQUIP Project: Modelling and Formal Verification in Action**
  Sponsor : Institut de Recherche Technologique Antoine de Saint Exupery (IRT)
  Role : Co-Investigator
  Duration : 2015-2016
  Publications : [C31]
  Software and Models : Methodology, modele, and case study
  Website : `http://www.irt-saintexupery.com`

Teaching Experience

INPT-ENSEEIHT, France (2015 – Present)

- N9EN12B Formal System Development in Event-B (Undergraduate course in Science and Numeric Department (60 Students), Fall 2016, Fall 2017, Fall 2018, Fall 2019, Fall 2020, Fall 2021, Fall 2022, co-developed with Yamine Ait-Ameur, Involvement and duties: 3 CM (60 Students), 4 TD (30 Students), 4 TP (20 Students), 5 Project (20 Students)[2])
- N8EN10A Compilers and Formal Semantics (Undergraduate course in Science and Numeric Department (60 Students), Spring 2020, Spring 2021, Spring 2022, Spring 2023, Involvement and duties: 10 TD (30 Students), 10 TP (30 Students), 4 Project (30 Students))
- N5EN03B Imperative Programming in Ada (Undergraduate course in Science and Numeric Department (195 Students), Fall 2015, Fall 2016, Fall 2017, Fall 2018, Fall 2019, Fall 2020, Fall 2021, Fall 2022, Involvement and duties: 10 TD (30 Students), 2 x 10 TP (15 Students), 4 Project (30 Students))
- N5AN02A Programming Methodology (Undergraduate course in Science and Numeric Department (28 Students), Fall 2015, Fall 2016, Fall 2017, Fall 2018, Fall 2019, Fall 2020, Fall 2021, Fall 2022, Involvement and duties: 18 TP (28 Students), 17 Project (28 Students))
- N6EN05A Object-Oriented Technology (Undergraduate course in Science and Numeric Department (195 Students), Spring 2017, Spring 2018, Spring 2019, Spring 2020, Spring 2021, SPring 2022, SPring 2023, Involvement and duties: 2 x TP (15 Students))
- NDI13C Cyber-Physical Systems (Undergraduate course in Science and Numeric Department (90

[2]Cours Magistraux (CM): Lectures, Travaux Dirigés (TD): Tutorials, Travaux Pratiques (TP): Practical Tutorials

Students), Fall 2016, Fall 2017, co-developed with Marc Pantel, Joseph Gergaud, Involvement and duties: 2 CM (105 Students), 2 x 5 TP (30 Students))

- N5EN05B Automatic (Undergraduate course in Science and Numeric Department (195 Students), Fall 2018, Fall 2019, Fall 2020, Fall 2021, Fall 2022, co-developed with Marc Pantel, Joseph Gergaud, Involvement and duties: 2 CM (195 Students), 5 TP (30 Students))
- N5EN03C C Language (Undergraduate course in Science and Numeric Department (195 Students), Fall 2018, Fall 2019, Fall 2020, Fall 2021, Fall 2022, Spring 2023 Involvement and duties: 6 TP (30 Students))
- N9EN25C Software development security (Master course in Science and Numeric Department (60 Students), Fall 2016, Fall 2018, Fall 2019, Fall 2020, Fall 2021, Fall 2022, Involvement and duties: 2 CM (60 Students), 2 TD (30 Students), 2 TP (20 Students))
- P22AT14 Computer science (Python introductory course at La Prépa des INP, Toulouse (90 Students), Fall 2016, Fall 2017, Fall 2018, Spring 2018, Fall 2019, Spring 2019, Fall 2020, Involvement and duties: 7 TP (30 Students))

McMaster University, Canada (2014 – 2015)

- *Software Design I*
  Guest lecturer, undergraduate level (instructor: Prof. Alan Wassyng)
- *Engineering Computation*
  Guest lecturer, graduate level (instructor: Prof. William Farmer)
- *Real-time system*
  Guest lecturer, graduate level (instructor: Prof. Prof. Douglas Down)

Université Henri Poincaré Nancy 1, France (2010)

Algorithm for Parallel and Distributed Systems
Teaching assistant("moniteur") in CS department

Indian Institute Of Technology Kanpur, India (2006)

Application of MATLAB in Engineering
Lectures and laboratory sessions, Quality Improvement Programme (QIP) Short term Course

Center of Development of Advanced Computing(C-DAC), India (2000 – 2003)

Programming languages, data structure database systems, computer architecture, etc.
Part-time instructor, diploma program

Invited Lectures

- *A Proof and Refinement Based Development for Cyber-Physical Systems*
  Invited guest lectures under the Global Initiative of Academic Networks (GIAN) program by the Indian Institute of Technology Patna (IITP)(Organiser: Prof. Raju Halder), India, (20 June - 01 July, 2022).
- *Development of Medical Devices: Techniques & Tools and Case Studies*
  Invited guest lectures under the Erasmus Mundus program for Dependable Software System (DESEM) by University of Lorraine (Organiser: Prof. Dominique Méry), France, (25 May - 29 May, 2015).

Ph.D Student Advising

In Progress

02/2023 – Present : **Yannis Benabbi**
Enrolment : INP-ENSEEIHT/IRIT, Toulouse
Funding : CIFRE (60K EUR + Ph.D scholarship)
Thesis : Formal verification and validation of railway systems using PERF
Supervision : Co-supervision with Yamine Aït Ameur
Publications : Not yet

07/2021 – Present : **Abhishek Kumar Saxena**
Enrolment : Amity University, Lucknow, India
Funding : PhD Scholarships
Thesis topic : Latency Optimization of edge learning through dimension reduction
Supervision : Co-supervision with Rajiv Pandey
Publications : [C4]

Completed

10/2017 – 04/2021 : **Guillaume Dupont**
Employment : Maîter de Conférences at INP-ENSEEIHT/IRIT, Toulouse
Funding : DISCONT project
Thesis : Correct-by-Construction Design of Hybrid Systems Based on Refinement and Proof **(Léopold ESCANDE thesis prize)**
Supervision : Co-supervision with Yamine Aït Ameur and Marc Pantel
Publications : [J1, J4, J7, C1, C2, C6, C7, C11, C14, C15, C16, C18, C20, C24, C26, C27]

10/2016 – 12/2021 : **Alexandra Halchin**
Employment : Research Engineer at RATP (Régie autonome des transports parisiens)
Funding : CIFRE (40K EUR + Ph.D scholarship)
Thesis : Development of a Formal Verification Methodology for B specifications using PERF formal toolkit
Supervision : Co-supervision with Yamine Aït Ameur, Abderrahmane Feliachi, and Julien Ordioni
Publications : [J8, C19, C22, C29]

12/2019 – 10/2023 : **Ismail Mendil**
Enrolment : INP-ENSEEIHT/IRIT, Toulouse
Funding : FORMEDICIS project
Thesis topic : A Framework for Explicit Modelling of Domain Knowledge in State-Based Formal Methods: the Case of Interactive Critical Systems
Supervision : Co-supervision with Yamine Aït Ameur, Philippe Palanque and Dominique Méry
Publications : [J1, J3, C5, C7, C9, C10, C13]

10/2020 – 06/2024 : **Peter Riviere**
Enrolment : INP-ENSEEIHT/IRIT, Toulouse
Funding : EDMITT Grant - French doctoral scholarships
Thesis topic : Automatic generation of proof obligations parameterised by domain theories implementation in Event-B: The EB4EB Framework
Supervision : Co-supervision with Yamine Aït Ameur
Publications : [J2, C1, C2, C5, C7, C8]

PhD Committee

- Brajesh Kumar Shukla, Ph.D, "*Development of an Instrument of Sit-to-stand for measurement of Sarcopenia in Older Indians*", Indian Institute of Technology Jodhpur (IITJ), Jodhpur, India, October 2020

Masters & Undergraduate Advising

- Clement Torti, undergraduate final project, "A Framework for Automating Testing Process", 2023, INP-ENSEEIHT/IRIT
- Mathieu Teissedre, undergraduate final project, "Input-Ooutput Simulators for Aeronautical Bus", 2022, INP-ENSEEIHT/IRIT
- Mickael Dalbin, undergraduate final project, "WebApp: SSO Authentication and Permissions Management", 2021, INP-ENSEEIHT/IRIT
- Thomas Salinas Broutee, undergraduate final project, "Development of VoIP analysis tools", 2020, INP-ENSEEIHT/IRIT
- Hasnae Dada, undergraduate project, "Cloud Services Integration Optimisation", 2017, INP-ENSEEIHT/IRIT
- Romain Geniet, M.Sc, "Verification of critical interactive systems through refinement and proof using Event-B", 2016, INP-ENSEEIHT/IRIT, co-advised with Yamine Ait-Ameur
- Mohamed Anas Charafi, undergraduate project, "Transformation of B model to HLL", 2016, INP-ENSEEIHT/IRIT
- Yanjun Jiang, M.A.Sc,"A Tabular Expression to Event-B Language Transformation Tool", 2015, McMaster University, co-advised with Tom Maibaum
- Sasan Vakili, M.A.Sc., "Design and Formal Verification of an Adaptive Cruise Control Plus (ACC+) System", 2015, McMaster University, co-advised with Mark Lawford
- Mischa Geven, M.A.Sc., 2014, McMaster University, co-advised with Alan Wassyng and Mark Lawford
- Nicholas Proscia, M.A.Sc., 2014, McMaster University, co-advised Alan Wassyng and Mark Lawford

Undergraduate Work-Study Student Supervision

- Yanis Kouidri, Thales Alenia Space France, 2022-2025
- Arnaud Fleury, Airbus Operation, 2021-2024
- Clement Torti, EDELIC, 2020-2023
- Matieu Teissedre, Rockwell Collins France, 2019-2022
- Mickael Dalbin, Dimension Data France, 2018-2021
- Thomas Salinas Broutee, Orange Direction Conception-Engineer, 2017-2020

Book Publications

[E1] **Neeraj Kumar Singh** *"Using Event-B for Critical Device Software Systems"*, Springer, ISBN: 978-1-4471-5259-0, I-XVIII, 1-326, 2013.

Book Editor

[BE1] Dominique Méry and **Neeraj Kumar Singh**, *"Vol-I: Modelling Software-based Systems"*, Wiley-ISTE, 2024 (in process).

[BE2] Dominique Méry and **Neeraj Kumar Singh**, *"Vol-II: Modelling Software-based Systems"*, Wiley-ISTE, 2024 (in process).

[BE3] Rajiv Pandey, Sunil Kumar Khatri, **Neeraj Kumar Singh**, Parul Verma *"Artificial Intelligence and Machine Learning for Edge Computing"*, Academic Press, ISBN: 978-0-12-824054-0, 2022.

[BE4] Rajiv Pandey, Nidhi Srivastava, **Neeraj Kumar Singh**, Kanishka Tyagi, *"Quantum Computing: A Shift from Bits to Qubits"*, to be published in book series "Studies in Computational Intelligence", Springer Singapore, ISBN: 978-981-19-9529-3, 05 May 2023.

Conference Proceedings Editor

[CPE1] Klaus-Dieter Schewe, **Neeraj Kumar Singh** *"Model and Data Engineering - 9th International Conference, MEDI 2019, Toulouse, France, October 28-31,2019, Proceedings"*, Lecture Notes in Computer Science, Springer, Vol-11815, ISBN: 978-3-030-32064-5.

Preprints

[PP1] Domnique Cansell and **Neeraj Kumar Singh**, "*Correct-by-Construction Synthesis of Sequential Algorithms*, 2024 (Submitted).

[PP2] Peter Rivière, **Neeraj Kumar Singh**, Yamine Ait-Ameur and Guillaume Dupont "*Extending the EB4EB framework with parameterised events*, 2024 (Submitted).

[PP3] Peter Rivière, Tsutomu Kobayashi, **Neeraj Kumar Singh**, Fuyuki Ishikawa, Yamine Ait-Ameur and Guillaume Dupont "*On-the-Fly Proof-Based Verification of Reachability in Autonomous Vehicle Controllers Relying on Goal-Aware RSS*, 2023 (Submitted).

[PP4] **Neeraj Kumar Singh**, Mark Lawford, Tom Maibaum and Alan Wassyng "*Refinement Automation and Proof-based Development using Tabular Expression*", 2024 (Submitted).

[PP5] **Neeraj Kumar Singh**,"*A virtual environment model for verification, simulation and clinical trials: Glucose homeostasis*", 2024 (Submitted).

Book Chapters

[B1] **Neeraj Kumar Singh**, Akshay M. Fajge, Raju Halder, and Md. Imran Alam"*Formal Verification and Code Generation for Solidity Smart Contracts*", In R. Pandey, S Goundar, and S. Fatima (Eds.), "Distributed Computing to Blockchain: Architecture, Technology, and Applications", Elsevier, 2022.

[B2] Raju Halder, Md. Imran Alam, Akshay M. Fajge, **Neeraj Kumar Singh** and Agostino Cortesi, "*Analyzing Information Flow in Solidity Smart Contract*", In R. Pandey, S Goundar, and S. Fatima (Eds.), "Distributed Computing to Blockchain: Architecture, Technology, and Applications", Elsevier, 2022.

[B3] **Neeraj Kumar Singh**, Yamine Ait-Ameur, and Dominique Méry "*Formal Ontological Analysis for Medical Protocols*", In Y. Ait-Ameur, S. Nakajima and D. Méry (Eds.), "Implicit and Explicit Semantics Integration in Proof Based Development of Discrete Systems", Lecture Notes in Computer Science, 2020.

[B4] **Neeraj Kumar Singh**, Mark Lawford, Thomas S. E. Maibaum, and Alan Wassyng "*Verifying Trustworthy Cyber-Physical Systems using Closed-loop Modelling*", In A. Romanovsky and F. Ishikawa (Eds.), "Trustworthy Cyber-Physical Systems Engineering", CRC Press Taylor & Francis Group, New York, pp. 462, ISBN 978149874245, 2016.

[B5] Sasan Vakili, **Neeraj Kumar Singh** and Mark Lawford "*Stop and Go Adaptive Cruise Control: A Case Study of Automotive Cyber-Physical Systems*", In A. Romanovsky and F. Ishikawa (Eds.), "Trustworthy Cyber-Physical Systems Engineering", CRC Press Taylor & Francis Group, New York, pp. 462, ISBN 9781498742450, 2016.

[B6] Alan Wassyng, **Neeraj Kumar Singh**, Mark Lawford, Thomas S. E. Maibaum"*New Standards for Trustworthy Cyber-Physical Systems*", In A. Romanovsky and F. Ishikawa (Eds.), "Trustworthy Cyber-Physical Systems Engineering", CRC Press Taylor & Francis Group, New York, pp. 462, ISBN 9781498742450, 2016.

[B7] Dominique Méry, Manamiary Bruno Andriamiarina and **Neeraj Kumar Singh** "*Incremental proof-based development for resilient distributed systems*", In A. Romanovsky and F. Ishikawa (Eds.), "Trustworthy Cyber-Physical Systems Engineering", CRC Press Taylor & Francis Group, New York, pp. 462, ISBN 9781498742450, 2016.

[B8] Dominique Méry and **Neeraj Kumar Singh** *"Event-B"*, in Formal Methods Applied to Industrial Complex System: Implementation of the B Method, John Wiley & Sons, 253-298, 2014.

[B9] Dominique Méry and **Neeraj Kumar Singh** *"Event-B"*, Jean-Louis Boulanger. Mise en oeuvre de la méthode B, HERMES, Apr. 2013, Informatique et SystÃ¨mes d'Informations, ISBN:978-2-7462-3810-7.

Refereed Journal Publications

[J1] Ismail Mendil, Yamine Ait-Ameur, **Neeraj Kumar Singh**, Guillaume Dupont, Dominique Méry, and Philippe Planque,". *Formal domain-driven system development in Event-B: Application to interactive critical systems*",Journal of Systems Architecture, Volume 135, ISSN 1383-7621, 2023.

[J2] Peter Reviere, **Neeraj Kumar Singh**, and Yamine Ait-Ameur,"*Reflexive Event-B: Semantics and Correctness The EB4EB framework*", IEEE Transactions on Reliability, p. 1-16, 2022.

[J3] **Neeraj Kumar Singh**, Yamine Ait-Ameur, Ismail Mendil, Dminique Méry, David Navarre, Philippe Palanque and Marc Pantel, "*Framework for Developing Critical Interactive Systems in FLUID*", Journal of Software: Evolution and Process,p. e2439,2022).

[J4] Guillaume Dupont, Yamine Ait-Ameur, **Neeraj Kumar Singh**, and Marc Pantel "*Formally Verified Architectural Patterns of Hybrid Systems Using Proof and Refinement with Event-B*", Science of Computer Programming, Vol. 216, p. 102765, 2022.

[J5] **Neeraj Kumar Singh**, Yamine Ait-Ameur, Romain Geniet, Dominique Méry and Philippe Palanque "*On the Benefits of Using MVC Pattern for Structuring Event-B Models of WIMP Interactive Applications*", Interacting with Computers, Volume 33, Issue 1, January 2021, Pages 9–114. `https://doi.org/10.1093/iwcomp/iwab016`

[J6] **Neeraj Kumar Singh**, Mark Lawford, Tom Maibaum and Alan Wassyng "*A Formal Approach to Rigorous Development of Critical Systems*", Journal of Software: Evolution and Process, 2021,33:e2334. `https://doi.org/10.1002/smr.2334`

[J7] Guillaume Dupont, Yamine Ait-Ameur, **Neeraj Kumar Singh**, and Marc Pantel "*Event-B Hybridation:A Proof and Refinement Based Framework for Modelling Hybrid Systems*", ACM Transactions on Embedded Computing Systems, 20, 4, Article 35 (June 2021), 37 pages. https://doi.org/10.1145/3448270.

[J8] Alexandra Halchin, Yamine Ait-Ameur, **Neeraj Kumar Singh**, Julien Ordioni, and Abderrahmane Feliachi "*Handling B Models in the PERF Integrated Verification Framework: Formalised and certified embedding*", Science of Computer Programming, Volume 196, p. 102477, 2020. https://doi.org/10.1016/j.scico.2020.102477

[J9] **Neeraj Kumar Singh** "*Detection of Postural Balance Degradation using Fuzzy Neural Network*", "International Journal of Bioinformatics Research and Applications (IJBRA)", Vol. 15, No. 4, 2019. https://dx.doi.org/10.1504/IJBRA.2019.103788

[J10] Alan Wassyng, **Neeraj Kumar Singh** , Mischa Geven, Nicholas Proscia, Hao Wang, Mark Lawford, Tom Maibaum "*Can Product-Specific Assurance Case Templates Be Used as Medical Device Standards?*", IEEE Design & Test 32(5): 45-55, 2015.

[J11] Manamiary Bruno Andriamiarina, Dominique Méry and **Neeraj Kumar Singh** "*Revisiting Snapshot Algorithms by Refinement-based Techniques*", Computer Science and Information Systems, 11(1):251–270, 2014.

[J12] Andy Wellings, Alan Burns, Ana Cavalcanti and **Neeraj Kumar Singh** "*Programming Simple Reactive Systems in Ada: Premature Program Termination*", ACM SIG Ada Letters, Vol-33(2), 75–86, 2013.

[J13] Dominique Méry and **Neeraj Kumar Singh** "*A Generic Framework: from Modeling to Code*", Journal of Innovations in Systems and Software Engineering, Springer London, 1–9, 2011.

[J14] Dominique Méry and **Neeraj Kumar Singh** "*Formal Specification of Medical Systems by Proof-Based Refinement*", ACM Transaction on Embedded Computing Systems, Vol-12(1), 15:1–15:25, January 2013.

[J15] Dominique Méry and **Neeraj Kumar Singh** "*Functional behavior of a cardiac pacing system*", International Journal of Discrete Event Control System, Vol-1, 129-149, January 2011.

Refereed Conference Publications

[C1] Flavio Ferrarotti Peter Reviere, Klaus-Dieter Schewe, **Neeraj Kumar Singh**, and Yamine Ait-Ameur,"*A Complete Fragment of LTL(EB)*", 13th International Symposium on Foundations of Information and Knowledge Systems, Lecture Notes in Computer Science, Springer, Sheffield, UK, 2024.

[C2] Peter Reviere, **Neeraj Kumar Singh**, Yamine Ait-Ameur, and Guillaume Dupont,"*Formalising Liveness Properties in Event-B with the Reflexive EB4EB Framework*", NASA Formal Methods, Lecture Notes in Computer Science, Springer International Publishing, USA, 2023.

[C3] Peter Reviere, **Neeraj Kumar Singh**, Yamine Ait-Ameur, and Guillaume Dupont,"*Standalone Event-B models analysis using the reflexive EB4EB framework*", 9th International Conference on Rigorous State Based Methods, Lecture Notes in Computer Science, Springer International Publishing, France 2023.

[C4] Abhishek Kumar Saxena, Rajiv Pandey, and **Neeraj Kumar Singh**. "*Latency Analysis and Reduction Methods for Edge Computing*", 2023 IEEE World Conference on Applied Intelligence and Computing (AIC), India 2023.

[C5] Ismail Mendil, Peter Riviere, Yamine Ait-Ameur, **Neeraj Kumar Singh**, Dominique Méry and Philippe Palanque, "*Non-Intrusive Annotation-Based Domain-Specific Analysis to Certify Event-B Models Behaviours*", The 29th Asia-Pacific Software Engineering Conference (APSEC '22), IEEE, pp. 129-138, Japan 2022.

[C6] Yamine Ait-Ameur, Sergiy Bogomolov, Guillaume Dupont, **Neeraj Kumar Singh**, Paulius Stankaitis "*Reachability Analysis and Simulation for Hybridised Event-B Models*", Integrated Formal Methods - 17th International Conference, IFM 2022, p. 109–128, Lugano, Switzerland, 2022.

[C7] Yamine Ait-Ameur, Guillaume Dupont, Ismail Mendil, Dominique Méry, Marc Pantel, Peter Riviere, and **Neeraj Kumar Singh**,"*Empowering the Event-B Method Using External Theories*", Integrated Formal Methods - 17th International Conference, IFM 2022, p. 18–35, Lugano, Switzerland, 2022.

[C8] Peter Riviere, **Neeraj Kumar Singh**, and Yamine Ait-Ameur, "*EB4EB: A Framework for Reflexive Event-B*", 26th International Conference on Engineering of Complex Computer Systems (ICECCS '22), IEEE, 2022, p. 71–80, Hiroshima City, Japan, 2022.

[C9] Ismail Mendil, Yamine Ait-Ameur, **Neeraj Kumar Singh**, Dominique Méry, and Philippe Palanque "*Leveraging Event-B Theories for handling domain knowledge in design models*", Symposium on Dependable Software Engineering - SETTA 2021, Lecture Notes in Computer Science, vol 6416, p. 312–326, Springer, Beijing, China, 2021.

[C10] Ismail Mendil, Yamine Ait-Ameur, **Neeraj Kumar Singh**, Dominique Méry, and Philippe Palanque "*Standard Conformance-by-Construction with Event-B*", Formal Methods for Industrial Critical Systems - 26th International Conference, FMICS 2021, Lecture Notes in Computer Science, vol 12863, p. 126–146, Springer, Paris, France, 2021.

[C11] Guillaume Dupont, Yamine Ait-Ameur, Marc Pantel and **Neeraj Kumar Singh**, "*Event-B Refinement for Continuous Behaviours Approximation*", The 19th International Symposium on Automated Technology for Verification and Analysis (ATVA'21), Lecture Notes in Computer Science, vol 12971, p. 320–336, Springer, Gold Coast, Australia, 2020,.

[C12] Yamine Ait-Ameur, Regine Laleau, Dominique Méry and **Neeraj Kumar Singh**, "*Towards Leveraging Domain Knowledge in State-Based Formal Methods*", In: Raschke A., Riccobene E., Schewe KD. (eds) Logic, Computation and Rigorous Methods. Lecture Notes in Computer Science, vol 12750. Springer, Cham, 2021.

[C13] Ismail Mendil, **Neeraj Kumar Singh**, Yamine Ait-Ameur, Dominique Méry and Philippe Palanque, "*An Integrated Framework for the Formal Analysis of Critical Interactive Systems*", The 27th Asia-Pacific Software Engineering Conference (APSEC '20), IEEE, 139–148, Singapore, 2020.

[C14] Guillaume Dupont, Yamine Ait-Ameur, Marc Pantel and **Neeraj Kumar Singh**, "*An Event-B Based Generic Framework for Hybrid Systems Formal Modelling*", 16th International Conference on integrated Formal Methods (iFM '20) , Lecture Notes in Computer Science, Springer International Publishing, 2020, Lugano, Switzerland.

[C15] Guillaume Dupont, Yamine Ait-Ameur, **Neeraj Kumar Singh**, F. Ishikawa, T. Kobayashi, and Marc Pantel, "*Embedding Approximation in Event-B: Safe Hybrid System Design using Proof and Refinement*", the 22nd International Conference on Formal Engineering Methods

(ICFEM '20), Lecture Notes in Computer Science, Springer, 2020, Singapore. [**Best Paper Award**]

[C16] Guillaume Dupont, Yamine Ait-Ameur, Marc Pantel and **Neeraj Kumar Singh**, "*Formally Verified Architecture Patterns of Hybrid Systems Using Proof and Refinement with Event-B*", 7th International ABZ Conference ASM, Alloy, B, TLA, VDM, Z (ABZ '20) , Lecture Notes in Computer Science, Springer International Publishing, p. 169-185, 2020, Ulm, Germany.

[C17] **Neeraj Kumar Singh**, Yamine Ait-Ameur, Dominique Méry, David Navarre, Philippe Palanque and Marc Pantel "*Formal Development of Multi-Purpose Interactive Application (MPIA) for ARINC 661*", Seventh International Workshop on Formal Techniques for Safety-Critical Systems, Co-located with ICFEM'2019, Shenzhen, China.

[C18] Paulius Stankaitis, Guillaume Dupont, Yamine Ait-Ameur, **Neeraj Kumar Singh**, Alexei Iliasov and Alexander Romanovsky, "*Modelling Hybrid Train Speed Controller using Proof and Refinement*", 24th International Conference on Engineering of Complex Computer Systems (ICECCS '19), IEEE, 2019, Nansha, Guangzhou, China.

[C19] Alexandra Halchin, Yamine Ait-Ameur, **Neeraj Kumar Singh**, Abderrahmane Feliachi and Julien Ordioni "*Certified Embedding of B Models in an IntegratedVerification Framework*", The 13th International Symposium on Theoretical Aspects of Software Engineering TASE'2019, 2019, Guilin, China.

[C20] Guillaume Dupont, Yamine Ait-Ameur, Marc Pantel and **Neeraj Kumar Singh**, "*Handling Refinement of Continuous Behaviors: A Proof Based Approach with Event-B*", The 13th International Symposium on Theoretical Aspects of Software Engineering TASE'2019, 2019, Guilin, China.

[C21] **Neeraj Kumar Singh**, Hao Wang "*Virtual Environment Model of Glucose Homeostasis for Diabetes Patients*", IEEE International Conference on Industrial Cyber-Physical Systems (ICPS '19), IEEE, Taipei, Taiwan, 2019.

[C22] Alexandra Halchin, **Neeraj Kumar Singh**, Yamine Ait-Ameur, Abderrahmane Feliachi and Julien Ordioni "*Validation of Formal Models Transformation through Animation*", First International Workshop on Knowledge and Model-driven engineering in formal development of Trustworthy Systems. Co-located with TASE'2019, 2019, Guilin, China.

[C23] **Neeraj Kumar Singh**, Yamine Ait-Ameur, and Dominique Méry "*Formal Ontology Driven Model Refactoring*", 23th International Conference on Engineering of Complex Computer Systems (ICECCS '18), IEEE, 2018, Melbourne, Australia.

[C24] Guillaume Dupont, Yamine Ait-Ameur, Marc Pantel and **Neeraj Kumar Singh**, "*Hybrid Systems and Event-B: A Formal Approach to Signalised Left-Turn Assist*", 2nd International Workshop on Cybersecurity and Functional Safety in Cyber-Physical Systems (IWCFS 2018), CCIS Springer, 2018, Marrakesh, Morocco.

[C25] Romain Geniet and **Neeraj Kumar Singh** "*Refinement Based Formal Development of Huma-Machine Interface*", 7th International Workshop on Formal Methods for Interactive Systems FMIS'18, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2018 Toulouse, France.

[C26] Guillaume Dupont, Yamine Ait-Ameur, Marc Pantel and **Neeraj Kumar Singh**, "*Proof-based approach to hybrid systems development: Dynamic Logic and Event-B*", 6th International ABZ Conference ASM, Alloy, B, TLA, VDM, Z (ABZ '18) , Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2018, Southampton, UK.

[C27] Guillaume Dupont, Yamine Ait-Ameur, **Neeraj Kumar Singh** and Marc Pantel, "*On the use of the Theory Plug-In to define theories for differential equations*", In 7th Rodin Workshop, ABZ 2018, Southampton, UK, June 5, 2018.

[C28] Yamine Aït Ameur, Dominique Méry, Idir Ait Sadoune, **Neeraj Kumar Singh**, Laurent Voisin and Paul Gibson "*On the importance of explicit domain modelling in refinement-based modelling design. Experiment with Event-B*", 6th International ABZ Conference ASM, Alloy, B, TLA, VDM, Z (ABZ '18), Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2018, Southampton, UK.

[C29] Alexandra Halchin, Abderrahmane Feliachi, **Neeraj Kumar Singh**, Yamine Ait-Ameur, and Julien Ordioni "*BPERFect: Applying the PERF approach to B based system development*", International Conference on Reliability, Safety and Security of Railway Systems: Modelling, Analysis, Verification and Certification RSSR '17, Lecture Notes in Computer Science, Springer, 2017, Pistoia, Italy.

[C30] **Neeraj Kumar Singh**, Mark Lawford, Thomas S. E. Maibaum, and Alan Wassyng "*Use of Tabular Expressions for Refinement Automation*", 7th International Conference on Model and Data Engineering MEDI '17, Lecture Notes in Computer Science, Springer, 2017, Barcelona, Spain.

[C31] **Neeraj Kumar Singh**, Yamine Ait-Ameur, Marc Pantel, Arnaud Dieumegard, and Eric Jenn, "*Stepwise Formal Modeling and Verification of Self-Adaptive systems with Event-B. The Automatic Rover Protection case study*", 21st International Conference on Engineering of Complex Computer Systems (ICECCS '16), IEEE Society, 2016, Dubai, UAE.

[C32] Guillaume Babin, Yamine Ait-Ameur, **Neeraj Kumar Singh**, and Marc Pantel, "*A Sytem Substitution Mechanism for Hybrid Systems in Event-B*", 18th International Conference on Formal Engineering Methods (ICFEM '16), Lecture Notes in Computer Science, Springer, 2016, Tokyo, Japan.

[C33] Sarah Benyagoub, Meriem Ouederni, **Neeraj Kumar Singh**, and Yamine Ait-Ameur,"*Correct-by-Construction Evolution of Realisable Conversation Protocols*", 6th International Conference on Model and Data Engineering MEDI '16 , Lecture Notes in Computer Science, Springer, 2016, Almeria, Spain.

[C34] **Neeraj Kumar Singh**, "*A Virtual Glucose Homeostasis Model for Verification, Simulation and Clinical Trials*", EuroSPI '16 , CCIS Springer, 2016, Graz, Austria.

[C35] Guillaume Babin, Yamine Ait-Ameur, **Neeraj Kumar Singh**, and Marc Pantel, "*Handling continuous functions in hybrid systems reconfigurations : a formal Event-B development*", ABZ '15 , Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2016, Linz, Austria.

[C36] **Neeraj Kumar Singh**,"*Detection of Hesitant Dynamic Postural Control*", 12th IEEE Colloquium on Signal Processing and its Applications (CSPA'2016), IEEE, pp. 36–40, Malacca Malaysia, 2016.

[C37] **Neeraj Kumar Singh**, Mark Lawford, T. S. E. Maibaum, Alan Wassyng "*Stateflow to Tabular Expressions*", The Sixth Symposium on Information and Communication Technology, SoICT'15, pp. 312–319, ACM New York, USA, 2015.

[C38] Valentin Cassano, Silviya Grigorova, **Neeraj Kumar Singh**, Morayo Adedjouma, Mark Lawford, T. S. E. Maibaum, Alan Wassyng "*Is Current Incremental Safety Assurance Sound?*", SAFECOMP Workshops 2015, Lecture Notes in Computer Science, Springer, pp. 397–408, Vol-9338, 2015.

[C39] Dominique Méry and **Neeraj Kumar Singh** "*Analyzing Requirements using Environment Modelling*", 17th International Conference on Human-Computer Interaction (HCI 2015),ecture Notes in Computer Science, Springer International Publishing, pp. 345–357, Vol-9185, 2015.

[C40] **Neeraj Kumar Singh**, Mark Lawford, Thomas S. E. Maibaum, and Alan Wassyng "*Formalizing The Cardiac Pacemaker Resynchronization Therapy*", 17th International Conference on Human-Computer Interaction (HCI 2015), Lecture Notes in Computer Science, Springer International Publishing, pp. 374–386, Vol-9185, 2015.

[C41] **Neeraj Kumar Singh**, Hao Wang, Mark Lawford, Thomas S. E. Maibaum, and Alan Wassyng "*Stepwise Formal Modelling and Reasoning of Insulin Infusion Pump Requirements*", 17th International Conference on Human-Computer Interaction (HCI 2015), Lecture Notes in Computer Science, Springer International Publishing, pp. 387–398, Vol-9185, 2015.

[C42] Dominique Méry and **Neeraj Kumar Singh** "*Formal Evaluation of Landing Gear System*", The Fifth Symposium on Information and Communication Technology, SoICT '14, ACM, New York, USA, pp. 75–84, 2014.

[C43] Dominique Méry and **Neeraj Kumar Singh** "*Modeling an Aircraft Landing System in Event-B*", ABZ 2014: The Landing Gear Case Study, Communications in Computer and Information Science, Springer International Publishing, pp. 154–159, Vol-433, 2014.

[C44] Manamiary Bruno Andriamiarina, Dominique Méry and **Neeraj Kumar Singh** "*Analysis of Self-* and P2P Systems Using Refinement*", ABZ '14 , Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 117-123, Vol-8477, 2014.

[C45] **Neeraj Kumar Singh**, Hao Wang, Mark Lawford, Thomas S. E. Maibaum, and Alan Wassyng "*Formalizing the Glucose Homeostasis Mechanism*", 16th International Conference on Human-Computer Interaction (HCI 2014), Lecture Notes in Computer Science, Springer International Publishing, pp. 460–4271, Vol-8529, 2014.

[C46] Dominique Méry and **Neeraj Kumar Singh** "*The Semantics of Refinement Chart*", 16th International Conference on Human-Computer Interaction (HCI 2014), Lecture Notes in Computer Science, Springer International Publishing, pp. 415–426, Vol-8529, 2014.

[C47] Manoranjan Satpathy, S. Ramesh, Colin F. Snook, **Neeraj Kumar Singh**, Michael J. Butler "*A Mixed Approach to Rigorous Development of Control Designs*", 2011 IEEE International Symposium on Computer-Aided Control System Design (CACSD '13), pp. 7–12, 2013.

[C48] Manamiary Bruno Andriamiarina, Dominique Méry and **Neeraj Kumar Singh** "*Integrating Proved State-Based Models for Constructing Correct Distributed Algorithm*", Integrated Formal Methods, Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 268-284, Vol-7940, 2013.

[C49] Dominique Méry and **Neeraj Kumar Singh** "*Ideal Mode Selection of a Cardiac Pacing System*", 15th International Conference on Human-Computer Interaction (HCII 2013), Lecture Notes in Computer Science, Springer Berlin Heidelbergevada, pp. 258-267, Vol-8025, 2013.

[C50] Manamiary Bruno Andriamiarina, Dominique Méry and **Neeraj Kumar Singh** "*Revisiting Snapshot Algorithms by Refinement-based Techniques*", 13th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2012), pp. 343-349, IEEE, 2012.

[C51] **Neeraj Kumar Singh**, Andy Wellings and Ana Cavalcanti "*The Cardiac Pacemaker Case Study and its Implementation in Safety-Critical Java and Ravenscar Ada*", Proceedings of the 10th International Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES 2012), ACM, 62–71, 2012.

[C52] Dominique Méry and **Neeraj Kumar Singh** "*Closed-loop modeling of Cardiac Pacemaker and Heart*", International Symposium on Foundations of Health Information Engineering and Systems (FHIES 2012), Springer LNCS, Vol-7789, pp. 151-166, 2013.

[C53] Dominique Méry and **Neeraj Kumar Singh** "*Critical Systems Development Methodology using Formal Techniques*", Proceedings of the Third Symposium on Information and Communication Technology, SoICT '12, ACM,New York, USA , pp. 3–12, 2012.

[C54] Dominique Méry and **Neeraj Kumar Singh** "*Automatic Code Generation from Event-B Models*", Proceedings of the Second Symposium on Information and Communication Technology, SoICT '11, ACM, New York, USA, pp. 179–188,2011.

[C55] Dominique Méry and **Neeraj Kumar Singh** "*Analysis of DSR protocol in Event-B*", 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2011), Springer LNCS, Vol-6976, pp. 401–415, October 2011.

[C56] Dominique Méry and **Neeraj Kumar Singh** "*Formal Development and Automatic Code Generation : Cardiac Pacemaker*", International Conference on Computers and Advanced Technology in Education (ICCATE 2011), Beijing, China, 3-4 November 2011 (Appear in a book EICE 2012, ASME Press, New york).

[C57] Dominique Méry and **Neeraj Kumar Singh** "*EB2J : Code Generation from Event-B to Java*", 14th Brazilian Symposium on Formal Methods (SBMF 2011), ISBN: 978-85-87837-21-9, 2011.

[C58] Dominique Méry and **Neeraj Kumar 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 '11), Springer LNCS, Vol-7151, pp. 140-159, 2012.

[C59] Dominique Méry and **Neeraj Kumar Singh** "*Medical Protocol Diagnosis using Formal Methods*", International Symposium on Foundations of Health Information Engineering and Systems (FHIES 2011), Springer LNCS, Vol-7151, pp. 1-20, 2012.

[C60] Dominique Méry and **Neeraj Kumar Singh** "*Trustable Formal Specification for Software Certification*", in Proceeding ISoLA (2), Springer LNCS,Vol-6416, pp. 312-326, 2010.

[C61] Dominique Méry and **Neeraj Kumar Singh** "*Real-time animation for formal specification*", in Proceeding Complex Systems Design & Management, Springer Berlin Heidelberg, pp. 49-60, 2010.

[C62] 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.

[C63] David Hewson, **Neeraj Kumar Singh**, Hichem Snoussi, Jacques Duchêne, "*Classification of elderly as fallers and non-fallers using Centre of Pressure velocity*" in Proceeding IEEE Engineering in Medicine and Biology Society (EMBC), 2010, p.3678-81.

[C64] **Neeraj Kumar Singh**, Hichem Snoussi, David Hewson, and Jacques Duchêne, "*Wavelet transform analysis of the power spectrum of centre of pressure signals to detect the critical point interval of postural control*" Communications in Computer and Information Science: Biomedical Engineering Systems and Technologies, vol. 52 (3), p. 235-244, 2010.

[C65] **Neeraj Kumar Singh**, Hichem Snoussi, David Hewson, and Jacques Duchêne, "*Detection of the critical point interval of postural control strategy using wavelet transform analysis*" in

Proceeding BIOSIGNAL 2009- Bio-inspired Systems and Signal Processing , p. 96-101, 2009 **(Best Paper Award)**.

[C66] Vrijendra Singh, Sandeep Yadav, **Neeraj Kumar Singh** and Prem Kumar Kalra "*Color image compression using block based independent component analysis*" , 5th International Symposium on Image and Signal Processing and Analysis (ISPA' 07), pp. 288-292, 2007.

THESIS

[TH1] **Neeraj Kumar Singh** "*Reliability and Safety of Critical Device Software Systems* ", Ph.D. Thesis, Department of Computer Science, LORIA, INRIA, Université Henri Poincaré Nancy 1, France, 344 pages, 2011. Appeared with Springer as *Using Event-B for Critical Device Software Systems,* Springer, 2013.

[TH2] **Neeraj Kumar Singh** "*Use of statistical mechanics methods to assess the effects of sensory perturbation and aging on stability during upright stance & Classification and detection of an increased risk of falling in elderly a Fuzzy Neural Network (FNN)* ", Master's Thesis, University of Technology of Troyes, Charles Delaunay Institute, Troyes, France, 155 pages, 2008.

[TH3] **Neeraj Kumar Singh** "*Modeling, Classification and Fault Detection of Sensors using Artificial Intelligence & Blind Source Separation using ICA Algorithms* ", MCA (Master of Computer Applications) Thesis, Uttar Pradesh Technical University, Lucknow, India, 135 pages, 2006.

TECHNICAL REPORTS

[R1] **Neeraj Kumar Singh**, Mark Lawford, Thomas S. E. Maibaum, and Alan Wassyng "*Report19 : Formal Evaluation of The Cardiac Pacemaker Resynchronization Therapy*", Technical Report 19 (https://www.mcscert.ca/index.php/documents/mcscert-reports), McSCert, McMaster University, October 2014.

[R2] **Neeraj Kumar Singh**, Hao Wang, Mark Lawford, Thomas S. E. Maibaum, and Alan Wassyng "*Report18 : Formalizing Insulin Pump using Event-B*", Technical Report 18 (https://www.mcsce-rt.ca/index.php/documents/mcscert-reports), McSCert, McMaster University, October 2014.

[R3] Dominique Méry and **Neeraj Kumar Singh** "*Technical Report on Modelling an Aircraft Landing System in Event-B*", Technical Report (http://hal.inria.fr/hal-00971787), 2014.

[R4] 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/), 2011.

[R5] 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/), 2011.

[R6] Dominique Méry and **Neeraj Kumar Singh** "*Pacemaker's Functional Behavior in Event-B*", Technical Report (http://hal.inria.fr/inria-00419973/en/), 2009.

[R7] Dominique Méry and **Neeraj Kumar Singh** "*Formal Development of Two-Electrode Cardiac Pacing System*" Technical Report, (https://hal.archives-ouvertes.fr/inria-00465061/en/), 2010).

Academic Software Development

[S1] EB2ALL : Automatic code generation from Event-B models to multiple programming languages (C, C++, Java, C#, Solidity, Ada, and Python), Web Site : `http://singh.perso.enseeiht.fr/eb2all/`.

[S2] B2HLL : Automatic translation of B models into High Level Language (HLL).
(To be technology transfer by RATP)

[S3] TX2EB : Automatic translation of tabular expressions into Event-B models, Web Site : `https://groke.cas.mcmaster.ca/gitlab/tables/jtet`

[S4] CZT : Extension of Community Z Tools (CZT) for supporting circus-time by developing parser and type-checker, Web Site : `http://czt.sourceforge.net`.

International Invited Meetings

[I1] Invited participation in the NII Shonan Meeting entitle *Implicit and Explicit Semantics Integration in Proof-Based Developments of Discrete Systems*, 21-25 November 2016.

[I2] Invited participation in the NII Shonan Meeting entitle *Science and Practice of Engineering Trustworthy Cyber-Physical Systems (TCPS)*, 27-30 October 2014.

[I3] Invited participation in the Dagstuhl Seminar (14062) on *The Pacemaker Challenge: Developing Certifiable Medical Devices*, 02-07 February 2014.

Invited Talks in Conference/Workshop

[IT1] Keynote speech on "*A Formal Approach to Rigorous Development of Critical Systems*", 4th Doctoral Symposium on Computational Intelligence (DoSCI 2023), 3 March, 2023, Lucknow, India.

[IT2] Invited talk on "*Emerging New Research Direction in Technologies for COVID-19*", Technical session of the two days International E-Conference on"Technological Support to fight against COVID-19", Awadhesh Pratap Singh University, India, 21-22 June, 2020.

[IT3] Invited talk on "*Do or Not Do a PhD?*", IAEM, University of Lorraine, Nancy, 18 October, 2016.

Conference Talks/Seminars

[T1] "*A Formal Framework for Developing Safety-Critical Systems*", Invited talk at Indian Institute of Technology Patna, Patna, India, 3 March, 2020.

[T2] "*Refinement Based Formal Development of Human-Machine Interface*", Conference talk at FMIS'18, Toulouse, France, 25 June, 2018.

[T3] "*Use of Tabular Expressions for Refinement Automation*", Conference talk at MEDI'17, Barcelona, Spain, 4-6 October, 2017.

[T4] "*A Formal Ontological Analysis in Medical Domain*", Seminar talk "Implicit and Explicit semantics integration in proof-based developments of discrete systems", NII Shonan Meetings Seminars, Tokyo Japan, 21-25 November, 2016.

[T5] "*A Virtual Glucose Homeostasis Model for Verification, Simulation and Clinical Trials*", Conference talk at EuroAsiaSPI'16, Graz, Austria, 14-16 September, 2016.

[T6] "*Stepwise Formal Modelling and Verification of Self-Adaptive systems with Event-B. The Automatic Rover Protection vase study*", Conference talk at ICECCS'16, Dubai, UAE, 06-08 November, 2016.

[T7] "*A Perspective on Environment Modelling for Verifying Cyber-Physical Systems*", Seminar talk "Journées formalisation des activités concurrentes", LAAS Toulouse, France, 30-31 March, 2016.

[T8] "*Stateflow to Tabular Expressions*", Conference talk at SoICT'15, Hue, Vietnam, 03-04 December, 2015.

[T9] "*Is Current Incremental Safety Assurance Sound?*", Conference talk at SAFECOMP'15, TU Delft, Netherlands, 22-25 September, 2015.

[T10] "*Step- wise Formal Modelling and Reasoning of Insulin Infusion Pump Requirements*", Conference talk at HCII'15, Los Angeles, USA, 03-07 August, 2015.

[T11] "*Analyzing Requirements using Environment Modelling*", Conference talk at HCII'15, Los Angeles, USA, 03-07 August, 2015.

[T12] "*Formalizing The Cardiac Pacemaker Resynchronization Therapy* ", Conference talk at HCII'15, Los Angeles, USA, 03-07 August, 2015.

[T13] "*A Perspective on Environment Modelling for Verifying Cyber-Physical Systems*", Seminar talk "Science and Practice of Engineering Trustworthy Cyber-Physical Systems", NII Shonan Meetings Seminars, Tokyo Japan, 27-30 October, 2014.

[T14] "*Development of Critical Systems using Event-B*", Seminar talk at Event B Day, NII Seminars, Tokyo Japan, 31 October, 2014.

[T15] "*Formalizing the Glucose Homeostasis Mechanism* ", Conference talk at HCII'14, Crete, Greece, 22-27 June, 2014.

[T16] "*The Semantics of Refinement Chart* ", Conference talk at HCII'14, Crete, Greece, 22-27 June, 2014.

[T17] "*Stateflow to Tabular Expression*", NECSIS Workshop, McMaster University, Canada, 17 July, 2014.

[T18] "*Development of Medical Device Software System*", Dagstuhl School, Germany, 03-08 February, 2014.

[T19] "*The Cardiac Pacemaker Case Study and its implementation in Safety-Critical Java and Ravenscar Ada* ", Conference talk at JTRES'12, Denmark, 25-27 October, 2012.

[T20] "*Formal Development of a Cardiac Pacemaker using a Refinement Approach*", University of York, York, UK, 08 May, 2012.

[T21] "*Automatic Code Generation from Event-B Models*", Conference talk at SoICT'11, Hanoi, Vietnam, 14 October, 2011.

[T22] "*Formalisation of the Heart based on Conduction of Electrical Impulses and Cellular-Automata*", Conference talk at FHIES'11, Johannesburg, South Africa, 30 August, 2011.

[T23] "*Medical Protocol Diagnosis using Formal Methods*", Conference talk at FHIES'11, Johannesburg, South Africa, 29 August, 2011.

[T24] "*A Generic Framework: from Modeling to Code*", Workshop talk at UML & FM'11, Limerick, Ireland, 20 June, 2011.

[T25] "*Refinement Based Development of Medical Systems*", Journée du groupe MFDL, TELECOM ParisTech, Paris, 02 December 2010.

[T26] "*Real-Time Animation for Formal Specification*", Conference talk at CSDM'10, Paris, France, 29 October, 2010.

[T27] "*Trustable Formal Specification for Software Certification* ", Conference talk at ISoLA'10, Crete, Greece, 16-21 October, 2010.

[T28] "*Refinement Based Development of Control Designs*", LORIA, France, 08 October, 2010.

[T29] "*Refinement Based Development of Control Designs*", General Motors, India Science Lab, Bangalore, India, 15 September 2010.

[T30] "*Formal Development of Cardiac Pacemaker using Refinement Approach*", General Motors, India Science Lab, Bangalore, India, 29 July 2010.

[T31] "*Formal Development of Cardiac Pacemaker using Refinement Approach*", SORIN GROUP Pacemaker Industry, Paris, 30 March 2010.

[T32] "*Development of Pacemaker Operating Modes using Refinement Approach*", A Pre-FM2009 workshop on Pacemaker challenge, Eindhoven, 1 November 2009.

[T33] "*Formal Model of Pacemaker*", RIMEL, Paris, 29 June 2009.

Scientific Activities

Teaching and Internal Responsibilities

- Advisory committee member of the SN department council (since 2023)
- In charge of third-year software engineering students (since 2023)
- Advisory committee member of the apprenti (work-study) students (since 2019)
- Teaching tutor for three apprenti (work-study) students for every year (since 2016)
- Course coordinator of the following courses: N9EN12B - Formal System Development in Event-B; N5EN05B - *Automatic (Cyber Physical System)*; and N5AN02A - *Programming Methodology* (since 2016)
- Participated for developing the following undergraduate courses: *"Programation Imperative"* and *"Automatique"*
- Participated for developing the course of Master degree "Performance in Software, Media and Scientific Computing (PSMSC)"
- Involve as a jury member for undergraduate students (since 2016)
- Involve in defense committee for more than 50 master undergraduate students (since 2016)

Program Chair

- 9th International Conference on Model and Data Engineering (MEDI 2019), 2019, INP Toulouse, France
  `https://www.irit.fr/MEDI2019`

Publication Chair

- 17th International Symposium on Theoretical Aspects of Software Engineering (TASE 2023), 2023, Bristol, United Kingdom
  `https://bristolpl.github.io/tase2023/index.html`

Publicity Chair

- 24th International Conference on Formal Engineering Methods (ICFEM 2023), 2023, Brisbane, Australia
  `https://formal-analysis.com/icfem/2023/`

Guest Editor

- Science of Computer Programming: Special issue of the 17th international symposium on Theoretical Aspects of Software Engineering
- International Journal of Embedded Systems - Inderscience Publishers

Scientific Evaluation Committees

- Natural Sciences and Engineering Research Council of Canada (NSERC 2023)
- Agence National de la Recherche (ANR 2016)

Conference/Event Organiser

- The 11th Rodin User and Developer Workshop, June, 2024, Bergamo, Italy (*Role:* Workshop co-organiser)
- The 27th International Conference on Engineering of Complex Computer Systems (ICECCS 2023), INP Toulouse, France (*Role:* Local conference organiser, and web chair)
- The 10th Rodin User and Developer Workshop, 30th May, 2023, Nancy, France (*Role:* Workshop co-organiser)
- The 9th International Conference on Model and Data Engineering (MEDI 2019), INP Toulouse, France (*Role:* Local conference organiser, financial chair and web site administrator)
- The 7th International Workshop on Formal Methods for Interactive Systems (FMIS 2018), INP Toulouse, France (*Role:* Local workshop organiser)
- Ecóle jeunes chercheurs en programmation 2017 (EJCP 2017), INP Toulouse, France (*Role:* Local EJCP program co-organiser)

Program Committee Member

- The 28th International Conference on Engineering of Complex Computer Systems (ICECCS 2024)
- 10th International Conference on Rigorous State-Based Methods (ABZ 2024)
- 7th International Conference on Signal Processing and Machine Learning (SPML 2024)
- 8th International Workshop on Cybersecurity and Functional Safety in Cyber-Physical Systems (IWCFS 2024)
- The 5th International Conference on Information Systems and Software Technologies (ICI2ST 2024)
- International Conference on Innovative Intelligent Industrial Production and Logistics (IN4PL 2024)
- 17th Theoretical Aspects of Software Engineering Conference (TASE 2023)
- The 27th International Conference on Engineering of Complex Computer Systems (ICECCS 2023)

- 9th International Conference on Rigorous State-Based Methods (ABZ 2023)
- The 12th International Conference on Model and Data Engineering (MEDI 2023)
- The 4th International Conference on Information Systems and Software Technologies (ICI2ST 2023)
- International Conference on Innovative Intelligent Industrial Production and Logistics (IN4PL 2023)
- 7th International Workshop on Cybersecurity and Functional Safety in Cyber-Physical Systems (IWCFS 2023)
- 6th International Conference on Signal Processing and Machine Learning (SPML 2023)
- 2nd International Workshop on Formal Engineering of Cyber-Physical Systems (FE-CPS 2023)
- 16th Theoretical Aspects of Software Engineering Conference (TASE 2022)
- The 11th International Conference on Model and Data Engineering (MEDI 2022)
- The 6th Workshop on Formal Co-Simulation of Cyber-Physical Systems (CoSim-CPS 2022)
- The 3rd International Conference on Information Systems and Software Technologies (ICI2ST 2022)
- 6th International Workshop on Cybersecurity and Functional Safety in Cyber-Physical Systems (IWCFS 2022)
- 1st International Workshop on Formal Engineering of Cyber-Physical Systems (FE-CPS 2022)
- International Conference on Innovative Intelligent Industrial Production and Logistics (IN4PL 2022)
- 5th International Conference on Signal Processing and Machine Learning (SPML 2022)
- The 5th Workshop on Formal Co-Simulation of Cyber-Physical Systems (CoSim-CPS 2021)
- IEEE International Conference on Smart Data Services (IEEE-SMDS 2021)
- The 10th International Conference on Model and Data Engineering (MEDI 2021)
- International Conference on Innovative Intelligent Industrial Production and Logistics (IN4PL 2021)
- 2021 4th International Conference on Signal Processing and Machine Learning (SPML 2021)
- 3rd ICSE Workshop on Software Engineering for Healthcare (Collocated to ICSE21) (SEH 2021)
- 5th International Workshop on Cybersecurity and Functional Safety in Cyber-Physical Systems (IWCFS 2021)
- International Conference on Innovative Intelligent Industrial Production and Logistics (IN4PL 2020)
- The 4th Workshop on Formal Co-Simulation of Cyber-Physical Systems (CoSim-CPS 2020)
- 2020 3rd International Conference on Signal Processing and Machine Learning (SPML 2020)
- The IEEE International Conference on Smart Data Services (SMDS 2020) (formerly the IEEE Big Data Congress)
- 4th International Workshop on Cybersecurity and Functional Safety in Cyber-Physical Systems (IWCFS 2020)
- 2nd International Workshop on Software Engineering for Healthcare (co-located with ICSA 2020) (SEH 2020)
- The 10th International Conference on Model and Data Engineering (MEDI 2020)
- The 3rd Workshop on Formal Co-Simulation of Cyber-Physical Systems (CoSim-CPS 2019)

- The 6th International Symposium on Big Data Principles, Architectures & Applications (BDAA 2019)
- The 5th IEEE International Conference on Big Data Intelligence and Computing (IEEE DataCom 2019)
- The 11th International Symposium on UbiSafe Computing (UbiSafe 2019)
- 2019 2nd International Conference on Signal Processing and Machine Learning (SPML 2019)
- Workshop on Practical Formal Verification for Software Dependability (AFFORD 2019)
- 3rd International Workshop on Cybersecurity and Functional Safety in Cyber-Physical Systems (IWCFS 2019)
- 1st International Workshop on Software Engineering for Healthcare (co-located with ICSE 2019) (SEH 2019)
- IEEE International Congress on Big Data 2019 (BigData Congress 2019)
- The 2018 International Conference on Signal Processing and Machine Learning (SPML 2018)
- Workshop on Practical Formal Verification for Software Dependability (AFFORD 2018)
- The 18th IEEE International Conference on Computer and Information Technology (IEEE CIT-2018)
- 2nd International Workshop on Cybersecurity and Functional Safety in Cyber-Physical Systems (IWCFS 2018)
- The 2nd Workshop on Formal Approaches for Advanced Computing Systems (FAACS 2018)
- 21st Brazilian Symposium on Formal Methods (SBMF 2018)
- 9th EAI International Conference on Big Data Technologies and Applications (BDTA 2018)
- The 8th International Conference on Model and Data Engineering (MEDI 2018)
- The 2nd Workshop on Formal Co-Simulation of Cyber-Physical Systems (CoSim-CPS 2018)
- The 16th International Conference on Software Engineering and Formal Methods (SEFM 2018)
- The IEEE 2018 7th International Congress on Big Data (BigData Congress 2018)
- Workshop on Practical Formal Verification for Software Dependability (AFFORD 2017)
- The 5th International Conference on Enterprise System (ES 2017)
- The 7th International Conference on Model and Data Engineering (MEDI 2017)
- The 1st Workshop on Formal Co-Simulation of Cyber-Physical Systems (CoSim-CPS 2017)
- 20th Brazilian Symposium on Formal Methods (SBMF 2017)
- The 1st Workshop on Formal Approaches for Advanced Computing Systems (FAACS 2017)
- The 13th IEEE Colloquium on Signal Processing and its Applications (CSPA 2017)
- The 17th IEEE International Conference on Computer and Information Technology (IEEE CIT-2017)
- The 3rd IEEE International Conference on Big Data Intelligence and Computing (IEEE DataCom 2017)
- Formal Verification for Practicing Engineers (FVPE 2016)
- 19th Brazilian Symposium on Formal Methods (SBMF 2016)
- The 6th International Conference on Model and Data Engineering (MEDI 2016)
- The 2nd IEEE International Conference on Big Data Intelligence and Computing (IEEE DataCom 2016)

- International Conference on Cancer Diagnostic and Medical Treatment Approaches (ICCDMTA 2016)
- International Workshop on Software Engineering in Healthcare Systems (SEH 2016)
- A Multidisciplinary View on Digital Support for Healthy Living and Self-management for Health (DIGITAL HEALTHY LIVING 2016)
- The 1st IEEE International Conference on Big Data Intelligence and Computing (IEEE DataCom 2015)
- Foundations of Health Information Engineering and Systems (FHIES 2012)
- 4th International workshop UML and Formal Methods (UML&FM 2011)

Book referee

- Springer
- Academic Press
- CRC Press

Journal referee

- ACM Transactions on Embedded Computing Systems
- Science of Computer Programming (SCP) (**Certificate of Outstanding Contribution in Reviewing, 2017, 2021)**
- Future Generation Computer Systems (FGCS)
- Formal Methods in System Design
- International Journal on Software Tools for Technology Transfer (STTT)
- The Computer Journal Formal Aspects of Computing
- Frontiers in Computer Science
- Journal of Software: Evolution Process
- Transactions on Network Science and Engineering
- Journal of Information and Telecommunication
- Journal of Universal Computer Science
- Journal of Information Security and Applications
- Journal of Systems Architecture
- Oxford Journal
- Computer Standard & Interface
- Computer Languages, Systems and Structures
- IEEE Software
- IEEE Design & Test
- Simulation Modelling Practice and Theory
- International Journal of Intelligent Information and Database Systems

Conferences referee

FM (2009, 2012, 2014, 2015, 2016, 2018, 2019), ICFEM (2009, 2010, 2011, 2012, 2014, 2015, 2018) SEFM (2010, 2018), ABZ (2010, 2014, 2023, 2024), iFM (2009, 2010, 2012, 2013, 2016, 2018), AFDL (2010), FHIES (2011, 2012), PSI(2011), B (2011), TASE(2012, 2013, 2020, 2021, 2022, 2023) , UML & FM(2012), ISoLA (2012), ICTAC (2013, 2015, 2016, 2019, 2020), ICECCS (2013, 2018, 2019,

2020, 2022, 2023, 2024), IEEE-SMDS (2021), MedicalCPS (2014), Model (2014), ASSURE (2014), DataCom (2015, 2016, 2017), MEMOCODE (2015), DIGITAL HEALTHY LIVING (2016), SEH (2016, 2019, 2020, 2021), EuroSPI (2016), MEDI (2016, 2017, 2018, 2019, 2021, 2022, 2023), FVPE (2016), SBMF (2016, 2017, 2018), CSPA (2017), IEEE-CIT (2017, 2018) , SCSS (2017), AFFORD (2017, 2018, 2019), Co-Sim-CPS (2017, 2018, 2019, 2020, 2022), ES (2017), FAACS (2017, 2018), MISP (2017), SCSS (2017), FMIS (2018), BDTA (2018), IEEE BigData (2018, 2019, 2020, 2021), SPML (2018, 2019, 2020, 2021, 2022, 2023, 2024), IWCFS (2018, 2019, 2020, 2021, 2022, 2023, 2024), UbiSafe (2019), SMDS (2020), FOSSACS (2019, 2020), IN4PL(2020, 2021, 2022, 2023, 2024),FE-CPS (2022, 2023), ICI2ST (2022,2023,2024).

References

[Solidity Github 2023] Solidity Github. https://github.com/ethereum/solidity, 2023. (Cited on page 62.)

[Abowd 1995] Gregory D. Abowd, Hung-Ming Wang and Andrew F. Monk. *A Formal Technique for Automated Dialogue Development*. In Proceedings of the 1st Conference on Designing Interactive Systems: Processes, Practices, Methods, & Techniques, DIS '95, pages 219–226, New York, NY, USA, 1995. ACM. (Cited on page 98.)

[Abrial 1996] Jean-Raymond Abrial. The B-book - Assigning Programs to Meanings. Cambridge University Press, 1996. (Cited on pages 2, 60 and 75.)

[Abrial 2007] Jean-Raymond Abrial and Stefan Hallerstede. *Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B*. Fundam. Inform., vol. 77, no. 1-2, pages 1–28, 2007. (Cited on page 98.)

[Abrial 2009] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Michael Leuschel, Matthias Schmalz and Laurent Voisin. *Proposals for Mathematical Extensions for Event-B*. Rapport technique, Southampton University, 2009. (Cited on pages 18 and 49.)

[Abrial 2010a] J.-R. Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010. (Cited on pages 2, 11, 18, 42, 46, 49, 57, 60 and 66.)

[Abrial 2010b] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta and Laurent Voisin. *Rodin: an open toolset for modelling and reasoning in Event-B*. International Journal on Software Tools for Technology Transfer, vol. 12, no. 6, pages 447–466, 2010. (Cited on pages 18, 30, 40, 42, 62, 100, 108 and 111.)

[Aït-Ameur 1998a] Yamine Aït-Ameur, Patrick Girard and Francis Jambon. *A Uniform Approach for Specification and Design of Interactive Systems: the B Method*. In Design, Specification and Verification of Interactive Systems'98, Supplementary Proceedings of the Fifth International Eurographics Workshop, June 3-5, 1998, Abingdon, United Kingdom, pages 51–67, 1998. (Cited on page 98.)

[Aït-Ameur 1998b] Yamine Aït-Ameur, Patrick Girard and Francis Jambon. *Using the B Formal Approach for Incremental Specification Design of Interactiv Systems*. In Engineering for Human-Computer Interaction, IFIP TC2/TC13 WG2.7/WG13.4 Seventh Working Conference on Engineering for Human-Computer Interaction, September 14-18,, Heraklion, Crete, Greece, pages 91–109, 1998. (Cited on page 98.)

[Aït-Ameur 2000] Yamine Aït-Ameur. *Cooperation of Formal Methods in an Engineering Based Software Development Process*. In Integrated Formal Methods, Second International Conference, IFM 2000, Dagstuhl Castle, Germany, November 1-3, 2000, Proceedings, pages 136–155, 2000. (Cited on page 98.)

[Aït-Ameur 2006] Yamine Aït-Ameur, Idir Aït-Sadoune, Jean-Marc Mota and Mickaël Baron. *Validation et vérification formelles de systèmes interactifs multi-modaux fondées sur la preuve*. In Proceedings of the 18th International Conference of the Association Francophone d'Interaction Homme-Machine, Montreal, Quebec, Canada, 18-21 April 2006, pages 123–130, 2006. (Cited on page 98.)

[Ait-Ameur 2016] Yamine Ait-Ameur and Dominique Méry. *Making Explicit Domain Knowledge in Formal System Development*. Sci. Comput. Program., vol. 121, no. C, pages 100–127, June 2016. (Cited on pages 3, 9 and 10.)

[Aït Ameur 2021] Yamine Aït Ameur, Shin Nakajima and Dominique Méry. Implicit and explicit semantics integration in proof-based developments of discrete systems. Springer, 2021. (Cited on page 10.)

[Ajmera 2013] Ishan Ajmera, Maciej Swat, Camille Laibe, Nicolas Le Novère and Vijayalakshmi Chelliah. *The impact of mathematical modeling on the understanding of diabetes and related complications*. CPT: Pharmacometrics & Systems Pharmacology, vol. 2, page e54, 2013. (Cited on pages 53, 54, 55 and 56.)

[Alt 2018] Leonardo Alt and Christian Reitwiessner. *SMT-Based Verification of Solidity Smart Contracts*. In Tiziana Margaria and Bernhard Steffen, Editors, Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice, pages 376–388, Cham, 2018. Springer International Publishing. (Cited on page 62.)

[Alur 1995] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis and Sergio Yovine. *The algorithmic analysis of hybrid systems*. Theoretical Computer Science, vol. 138, no. 1, pages 3–34, 1995. Hybrid Systems. (Cited on page 88.)

[Anand 2018] A. Anand, S. Boulier, C. Cohen, M. Sozeau and N. Tabareau. *Towards Certified Meta-Programming with Typed Template-Coq*. In Jeremy Avigad and Assia Mahboubi, Editors, 9th International Conference, ITP. Part of FloC 2018, volume 10895 of *LNCS*, pages 20–39. Springer, 2018. (Cited on page 48.)

[Annenkov 2021] Danil Annenkov, Mikkel Milo, Jakob Botsch Nielsen and Bas Spitters. *Extracting Smart Contracts Tested and Verified in Coq*. In Proceedings of the 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2021, pages 105–121, New York, NY, USA, 2021. Association for Computing Machinery. (Cited on page 62.)

[Antoniou 2004] Grigoris Antoniou and Frank van Harmelen. *Web Ontology Language: OWL*. In Steffen Staab and Rudi Studer, Editors, Handbook on Ontologies, pages 67–92. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. (Cited on pages 18 and 19.)

[Arduino] Arduino. `https://www.arduino.cc`. (Cited on page 59.)

[ARINC 661 specification 2002] ARINC 661 specification. *Cockpit Display System Interfaces To User Systems, Prepared by Airlines Electronic Engineering Committee, Published by AERONAUTICAL RADIO, INC., April 22*, 2002. (Cited on pages 48 and 99.)

[ARINC 2019] ARINC. Arinc 661 specification: Cockpit display system interfaces to user systems. by aeec, published by sae, 16701 melford blvd., suite 120, bowie, maryland 20715 usa. ARINC Industry Activities, June 2019. (Cited on page 20.)

[Asarin 2002] Eugene Asarin, Thao Dang and Oded Maler. *The d/dt Tool for Verification of Hybrid Systems*. In Ed Brinksma and Kim Guldstrand Larsen, Editors, Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings, pages 365–370. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. (Cited on page 88.)

[Atzei 2017] Nicola Atzei, Massimo Bartoletti and Tiziana Cimoli. *A Survey of Attacks on Ethereum Smart Contracts SoK*. In Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204, pages 164–186, Berlin, Heidelberg, 2017. Springer-Verlag. (Cited on page 62.)

[Back 1996] R.J.R. Back and K. Sere. *Superposition refinement of reactive systems*. Formal Aspects of Computing, vol. 8, no. 3, pages 324–346, 1996. (Cited on page 24.)

[Back 1998] Ralph-Johan Back and Joakim von Wright. Refinement calculus - A systematic introduction. Graduate Texts in Computer Science. Springer, 1998. (Cited on page 46.)

[Back 2000] Ralph-Johan Back, Luigia Petre and Ivan Porres. *Generalizing Action Systems to Hybrid Systems*. In Proceedings of the 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT '00, pages 202–213, London, UK, UK, 2000. Springer-Verlag. (Cited on page 88.)

[Badeau 2004] Frédéric Badeau, Didier Bert, Sylvain Boulmé, Christophe Métayer, Marie-Laure Potet, Nicolas Stouls and Laurent Voisin. *Adaptabilité et validation de la traduction de B vers C. Points de vue et résultats du projet BOM*. Technique et Science Informatiques, vol. 23, no. 7, pages 879–903, 2004. (Cited on page 78.)

[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 2.)

[Banach 2013] Richard Banach. *Pliant Modalities in Hybrid Event-B*. In Zhiming Liu, Jim Woodcock and Huibiao Zhu, Editors, Theories of Programming and Formal Methods, volume 8051 of *LNCS*, pages 37–53. Springer Berlin Heidelberg, 2013. (Cited on page 88.)

[Banach 2015] Richard Banach, Michael Butler, Shengchao Qin, Nitika Verma and Huibiao Zhu. *Core Hybrid Event-B I: Single Hybrid Event-B machines*. Science of Computer Programming, 2015. (Cited on page 88.)

[Banach 2020] Richard Banach. *Verification-Led Smart Contracts*. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne and Massimiliano Sala, Editors, Financial Cryptography and Data Security, pages 106–121, Cham, 2020. Springer International Publishing. (Cited on page 62.)

[Barboni 2003] Eric Barboni, Rémi Bastide, Xavier Lacaze, David Navarre and Philippe Palanque. *Petri Net Centered versus User Centered Petri Nets Tools*. In 10th Workshop on Algorithms and Tools for. Petri Nets (AWPN 2003), Eichstätt, Germany, September 2003. (Cited on pages 43 and 46.)

[Barney 2001] David Barney, David Haley and George Nikandros. *Calculating Train Braking Distance*. In Proceedings of the Sixth Australian Workshop on Safety Critical Systems and Software - Volume 3, SCS '01, pages 23–29, AUS, 2001. Australian Computer Society, Inc. (Cited on page 92.)

[Barrett 2011] Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds and Cesare Tinelli. *CVC4*. In Ganesh Gopalakrishnan and Shaz Qadeer, Editors, Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011. (Cited on page 108.)

[Benaissa 2016] Nazim Benaissa, David Bonvoisin, Abderrahmane Feliachi and Julien Ordioni. *The PERF Approach for Formal Verification*. In RSSRail, pages 203–214, 2016. (Cited on page 74.)

[Bengtsson 1996] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson and Wang Yi. *UPPAAL — a tool suite for automatic verification of real-time systems*. In Rajeev Alur, Thomas A. Henzinger and Eduardo D. Sontag, Editors, Hybrid Systems III, pages 232–243, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. (Cited on pages 2 and 17.)

[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 pages 2 and 119.)

[Bert 2003] Didier Bert, Sylvain Boulmé, Marie-Laure Potet, Antoine Requet and Laurent Voisin. *Adaptable Translator of B Specifications to Embedded C Programs*. In Keijiro Araki, Stefania Gnesi and Dino Mandrioli, Editors, FME 2003: Formal Methods, pages 94–113, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. (Cited on pages 60 and 74.)

[Bertot 2010] Yves Bertot and Pierre Castran. Interactive theorem proving and program development: Coq'art the calculus of inductive constructions. Springer Publishing Company, Incorporated, 1st edition, 2010. (Cited on pages 2, 17, 32, 48, 74, 88 and 116.)

[Besnard 2009] Loïc Besnard, Thierry Gautier, Matthieu Moy, Jean-Pierre Talpin, Kenneth Johnson and Florence Maraninchi. *Automatic translation of C/C++ parallel code into synchronous formalism using an SSA intermediate form*. In Proceedings of the Ninth International Workshop on Automated Verification of Critical Systems (AVOCS 2009), volume 23, 2009. (Cited on page 74.)

[Bettini 2013] Lorenzo Bettini. Implementing domain-specific languages with xtext and xtend. Packt Publishing, 2013. (Cited on page 45.)

[Beyer 2007] Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala and Rupak Majumdar. *The software model checker Blast*. International Journal on Software Tools for Technology Transfer, vol. 9, no. 5, pages 505–525, Oct 2007. (Cited on pages 23 and 32.)

[Bhargavan 2016] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy and Santiago Zanella-Béguelin. *Formal Verification of Smart Contracts: Short Paper*. In Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, PLAS '16, pages 91–96, New York, NY, USA, 2016. Association for Computing Machinery. (Cited on page 62.)

[Bhattacharyya 2020] Shuvra S. Bhattacharyya and Marilyn C. Wolf. *Research Challenges for Heterogeneous CPS Design*, 2020. (Cited on page 5.)

[Bicarregui 1991] J. C. Bicarregui and B. Ritchie. *Reasoning about VDM developments using the VDM support tool in mural*. In S. Prehn and W. J. Toetenel, Editors, VDM'91 Formal Software Development Methods, pages 371–388, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg. (Cited on page 49.)

[Biernacki 2007] Dariusz Biernacki, Jean louis Colaco and Marc Pouzet. *Clock-directed Modular Code Generation from Synchronous Block Diagrams*. In APGES, 2007. (Cited on page 74.)

[Bittner 2007] Thomas Bittner and Maureen Donnelly. *Logical Properties of Foundational Relations in Bio-ontologies*. Artif. Intell. Med., vol. 39, no. 3, pages 197–216, March 2007. (Cited on page 14.)

[Bjørner 1978] Dines Bjørner and Cliff B. Jones, Editors. The Vienna Development Method: The Meta-Language, London, UK, 1978. Springer-Verlag. (Cited on pages 2 and 49.)

[Bjørner 2009] Dines Bjørner. Domain engineering - technology management, research and engineering, volume 4 of *COE Research Monograph Series*. JAIST, 2009. (Cited on page 9.)

[Bjørner 2017] Dines Bjørner. *Manifest domains: analysis and description*. Formal Asp. Comput., vol. 29, no. 2, pages 175–225, 2017. (Cited on pages 3, 9 and 18.)

[Bjørner 2019] Dines Bjørner. *Domain Analysis and Description Principles, Techniques, and Modelling Languages*. ACM Trans. Softw. Eng. Methodol., vol. 28, no. 2, pages 8:1–8:67, 2019. (Cited on pages 9 and 18.)

[Blech 2004] Jan Olaf Blech and Sabine Glesner. *A Formal Correctness Proof for Code Generation from SSA Form in Isabelle/HOL*. In GI Jahrestagung, 2004. (Cited on pages 74 and 75.)

[Blech 2007] Jan Olaf Blech and Arnd Poetzsch-Heffter. *A Certifying Code Generation Phase*. Electron. Notes Theor. Comput. Sci., vol. 190, no. 4, pages 65–82, November 2007. (Cited on pages 74 and 75.)

[Blochwitz 2011] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J. v. Peetz, S. Wolf, Atego Systems Gmbh, Qtronic Berlin, Fraunhofer Scai and St. Augustin. *The Functional Mockup Interface for Tool independent Exchange of Simulation Models*. In In Proceedings of the 8th International Modelica Conference, 2011. (Cited on page 41.)

[Blochwitz 2012] Torsten Blochwitz, Martin Otter, Johan Åkesson, Martin Arnold, Christoph Clauß, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauß, Dietmar Neumerkel, Hans Olsson and Antoine Viel. *Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models*. In Proceedings of the 9th International Modelica Conference, pages 173–184. The Modelica Association, 2012. (Cited on page 116.)

[Bodeveix 2021] Jean-Paul Bodeveix and Mamoun Filali. *Event-B Formalization of Event-B Contexts*. In Alexander Raschke and Dominique Méry, Editors, Rigorous State-Based Methods, pages 66–80, Cham, 2021. Springer International Publishing. (Cited on page 49.)

[Boehm 1976] B. W. Boehm. *Software Engineering*. IEEE Trans. Comput., vol. 25, pages 1226–1241, December 1976. (Cited on page 32.)

[Boehm 1988] B. W. Boehm. *A spiral model of software development and enhancement*. Computer, vol. 21, no. 5, pages 61–72, May 1988. (Cited on page 32.)

[Bogomolov 2019] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Kostiantyn Potomkin and Christian Schilling. *JuliaReach: A Toolbox for Set-Based Reachability*. In Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC '19, pages 39–44, New York, NY, USA, 2019. Association for Computing Machinery. (Cited on page 91.)

[Bolie 1961] Victor W. Bolie. *Coefficients of normal blood glucose regulation*. Journal of Applied Physiology, vol. 16, no. 5, pages 783–788, 1961. (Cited on pages 53 and 55.)

[Bolton 2009] Matthew L. Bolton and Ellen J. Bass. *Building a Formal Model of a Human-interactive System: Insights into the Integration of Formal Methods and Human Factors Engineering*. In First NASA Formal Methods Symposium - NFM, California, USA, April 6-8., pages 6–15, 2009. (Cited on page 98.)

[Bolton 2011] M. L. Bolton, R. I. Siminiceanu and E. J. Bass. *A Systematic Approach to Model Checking Human - Automation Interaction Using Task Analytic Models*. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 41, no. 5, pages 961–976, 2011. (Cited on page 98.)

[Bonichon 2015] Richard Bonichon, David Déharbe, Thierry Lecomte and Valério Medeiros. LLVM-Based Code Generation for B, pages 1–16. Springer International, 2015. (Cited on page 74.)

[Boniol 2014] Frédéric Boniol and Virginie Wiels. *The Landing Gear System Case Study*. In Frédéric Boniol, Virginie Wiels, Yamine Aït-Ameur and Klaus-Dieter Schewe, Editors, ABZ 2014: The Landing Gear Case Study, pages 1–18, Cham, 2014. Springer International Publishing. (Cited on pages 95, 96 and 97.)

[Börger 2003] Egon Börger and Robert F. Stärk. Abstract State Machines. A Method for High-Level System Design and Analysis. Lecture Notes in Computer Science. Springer, 2003. (Cited on pages 2 and 48.)

[Bourke 2017] Timothy Bourke, Lélio Brun, Pierre-Évariste Dagand, Xavier Leroy, Marc Pouzet and Lionel Rieg. *A Formally Verified Compiler for Lustre*. In Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, pages 586–601. ACM, 2017. (Cited on pages 74 and 78.)

[Bouton 2009] Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe and Pascal Fontaine. *veriT: An Open, Trustable and Efficient SMT-Solver*. In Renate A. Schmidt, Editor, Automated Deduction – CADE-22, pages 151–156, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. (Cited on page 108.)

[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 2, 23 and 105.)

[Breton 2016] Nicolas Breton and Yoann Fonteneau. *S3: Proving the Safety of Critical Systems*. In Thierry Lecomte, Ralf Pinger and Alexander Romanovsky, Editors, Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification, pages 231–242, Cham, 2016. Springer International Publishing. (Cited on pages 98 and 117.)

[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 119.)

[Brunel 2018] Julien Brunel, David Chemouil, Alcino Cunha and Nuno Macedo. *The Electrum Analyzer: Model Checking Relational First-Order Temporal Specifications*. In 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18), Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Montpellier, France, September 2018. ACM Press. (Cited on page 42.)

[Butler 1996] 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 page 2.)

[Butler 2013] Michael J. Butler and Issam Maamria. *Practical Theory Extension in Event-B*. In Theories of Prog. and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday, pages 67–81, 2013. (Cited on pages 18, 34, 35 and 49.)

[Butler 2016] Michael Butler, Jean-Raymond Abrial and Richard Banach. *Modelling and Refining Hybrid Systems in Event-B and Rodin*. In Luigia Petre and Emil Sekerinski, Editors, From Action Systems to Distributed Systems: The Refinement Approach, Computer and Information Science Series, pages 29–42. Chapman and Hall/CRC, April 2016. (Cited on pages 32 and 88.)

[Butler 2020] Michael J. Butler, Philipp Körner, Sebastian Krings, Thierry Lecomte, Michael Leuschel, Luis-Fernando Mejia and Laurent Voisin. *The First Twenty-Five Years of Industrial Use of the B-Method*. In Formal Methods for Industrial Critical Systems - 25th International Conference, FMICS 2020, Vienna, Austria, September 2-3, 2020, Proceedings, pages 189–209, 2020. (Cited on page 60.)

[Calegari 2016] Daniel Calegari, Till Mossakowski and Nora Szasz. *Heterogeneous verification in the context of model driven engineering*. Science of Computer Programming, Elsevier Journal., vol. 126, pages 3–30, 2016. (Cited on page 18.)

[Campos 2008] J. Creissac Campos and M. D. Harrison. *Systematic Analysis of Control Panel Interfaces Using Formal Tools*. In T. C. Nicholas Graham and Philippe Palanque, Editors, 15th International Workshop on Interactive Systems. Design, Specification, and Verification (DSV-IS), pages 72–85. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. (Cited on page 98.)

[Carayon 2009] Pascale Carayon and Kenneth E. Wood. *Patient safety*. Information, Knowledge, Systems Management, vol. 8, no. 1-4, pages 23–46, 2009. (Cited on pages 31, 53 and 105.)

[Cataño 2012] Néstor Cataño, Tim Wahls, Camilo Rueda, Víctor Rivera and Danni Yu. *Translating B Machines to JML Specifications*. In Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12, pages 1271–1277. ACM, 2012. (Cited on page 74.)

[CFD-ACE+] CFD-ACE+. https://www.esi-group.com/software-solutions/virtual-environment/cfd-multiphysics/ace-suite/cfd-ace. (Cited on page 58.)

[Chandy 1989] K. Mani Chandy and Jayadev Misra. Parallel Program Design - a Foundation. Addison-Wesley, 1989. (Cited on page 2.)

[Charafi 2017] Anas Charafi. *Development of a tool for the translation of B models to HLL*. Rapport technique, National Polytechnic Institute of Toulouse, 2017. Master project report. (Cited on pages 75, 77 and 80.)

[Chatty 2015] Stéphane Chatty, Mathieu Magnaudet and Daniel Prun. *Verification of Properties of Interactive Components from Their Executable Code*. In Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS '15, pages 276–285, New York, NY, USA, 2015. ACM. (Cited on page 98.)

[Chay 1985] T.R. Chay and J. Keizer. *Theory of the effect of extracellular potassium on oscillations in the pancreatic beta-cell*. Biophysical Journal, vol. 48, no. 5, pages 815–827, 1985. (Cited on page 53.)

[Chen 2013] Xin Chen, Erika Ábrahám and Sriram Sankaranarayanan. *Flow*: An Analyzer for Non-linear Hybrid Systems*. In Natasha Sharygina and Helmut Veith, Editors, Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings, volume 8044 of *Lecture Notes in Computer Science*, pages 258–263. Springer, 2013. (Cited on page 88.)

[Chen 2014] Yihai Chen, Mark Lawford, Hao Wang and Alan Wassyng. *Insulin Pump Software Certification*. In Jeremy Gibbons and Wendy MacCaull, Editors, Foundations of Health Information Engineering and Systems, volume 8315 of *LNCS*, pages 87–106. Springer Berlin Heidelberg, 2014. (Cited on pages 1, 32 and 105.)

[Cimatti 2002] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani and Armando Tacchella. *NuSMV 2: An OpenSource Tool for Symbolic Model Checking*. In Ed Brinksma and Kim Guldstrand Larsen, Editors, Computer Aided Verification (CAV'02), pages 359–364, Berlin, Heidelberg, 2002. Springer. (Cited on pages 2 and 17.)

[Cockburn 2006] Alistair Cockburn. Agile software development: The cooperative game (2nd edition) (agile software development series). Addison-Wesley Professional, 2006. (Cited on page 32.)

[COMSOL Multiphysics] COMSOL Multiphysics. https://www.comsol.com. (Cited on page 58.)

[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 page 2.)

[CPS Steering Group 2008] CPS Steering Group. *Cyber-Physical SystemsExecutive Summary*, 2008. (Cited on pages 4 and 5.)

[Cuoq 2012] Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles and Boris Yakobowski. *Frama-C: A Software Analysis Perspective*. In Proceedings of the 10th International Conference on Software Engineering and Formal Methods, SEFM'12, pages 233–247, Berlin, Heidelberg, 2012. Springer-Verlag. (Cited on pages 23, 32 and 71.)

[Curzon 2014] Paul Curzon, Paolo Masci, Patrick Oladimeji, Rimvydas Rukšėnas, Harold Thimbleby and Enrico D'Urso. *Human-Computer Interaction and the Formal Certification and Assurance of Medical Devices: The CHI+MED Project*. In 2nd Workshop on Verification and Assurance (Verisure2014), in association with Computer-Aided Verification (CAV), part of the Vienna Summer of Logic, 2014. (Cited on page 98.)

[Dalvandi 2019] Mohammadsadegh Dalvandi, Michael J. Butler and Asieh Salehi Fathabadi. *SEB-CG: Code Generation Tool with Algorithmic Refinement Support for Event-B*. In Emil Sekerinski, Nelma Moreira, José N. Oliveira, Daniel Ratiu, Riccardo Guidotti, Marie Farrell, Matt Luckcuck, Diego Marmsoler, José Creissac Campos, Troy Astarte, Laure Gonnord, Antonio Cerone, Luis Couto, Brijesh Dongol, Martin Kutrib, Pedro Monteiro and David Delmas, Editors, Formal Methods. FM 2019 International Workshops - Porto, Portugal, October 7-11, 2019, Revised Selected Papers, Part I, volume 12232 of *Lecture Notes in Computer Science*, pages 19–29. Springer, 2019. (Cited on page 60.)

[De Gaetano 2000] Andrea De Gaetano and Ovide Arino. *Mathematical modelling of the intravenous glucose tolerance test*. Journal of Mathematical Biology, vol. 40, no. 2, pages 136–168, 2000. (Cited on page 53.)

[de Moura 2008] Leonardo de Moura and Nikolaj Bjørner. *Z3: An Efficient SMT Solver*. In C. R. Ramakrishnan and Jakob Rehof, Editors, Tools and Algorithms for the Construction and Analysis of Systems, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. (Cited on page 108.)

[Dedeoglu 2020] Volkan Dedeoglu, Ali Dorri, Raja Jurdak, Regio A. Michelin, Roben C. Lunardi, Salil S. Kanhere and Avelino F. Zorzo. *A Journey in Applying Blockchain for Cyberphysical Systems*. In 2020 International Conference on COMmunication Systems & NETworkS (COMSNETS), pages 383–390, 2020. (Cited on page 121.)

[Déharbe 2014] David Déharbe, Pascal Fontaine, Yoann Guyot and Laurent Voisin. *Integrating SMT Solvers in Rodin*. Sci. Comput. Program., vol. 94, no. P2, pages 130–143, October 2014. (Cited on page 91.)

[Denney 2013] Ewen Denney and Ganesh Pai. *A Formal Basis for Safety Case Patterns*. In Proceedings of the 32nd International Conference on Computer Safety, Reliability, and Security - Volume 8153, SAFECOMP 2013, pages 21–32, Berlin, Heidelberg, 2013. Springer-Verlag. (Cited on page 83.)

[DO-278 011] RTCA DO-278. *Software Integrity Assurance Considerations for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems*, (December 13, 2011). (Cited on page 2.)

[DODD-5000.61 018] DODD-5000.61. *DoD Modeling and Simulation (M&S) Verification, Validation, and Accreditation (VV&A)*, (October 15, 2018). (Cited on page 2.)

[Dori 2016] Dov Dori. Model-based systems engineering with opm and sysml. Springer Publishing Company, Incorporated, 1st edition, 2016. (Cited on page 1.)

[Drozdov 1995] A. Drozdov and H. Khanina. *A model for ultradian oscillations of insulin and glucose*. Mathematical and Computer Modelling, vol. 22, no. 2, pages 23–38, 1995. (Cited on page 53.)

[Du Bois 2004] Bart Du Bois, Serge Demeyer and Jan Verelst. *Refactoring " Improving Coupling and Cohesion of Existing Code*. In Proceedings of the 11th Working Conference on Reverse Engineering, WCRE '04, pages 144–151. IEEE Computer Society, 2004. (Cited on pages 10 and 12.)

[Dupont 2021] Guillaume Dupont. *Correct-by-Construction Design of Hybrid Systems Based on Refinement and Proof*. PhD thesis, Institut National Polytechnique de Toulouse, Toulouse, France, 2021. (Cited on pages 34, 35, 49 and 89.)

[EB2ALL 2022] EB2ALL. *Automatic code generation from Event-B to many Programming Languages*. http://singh.perso.enseeiht.fr/eb2all/, 2011-2022. (Cited on pages 64 and 118.)

[EB2Sol 2021] EB2Sol. *Automatic code generation from Event-B to Solidity*, 2021. (Cited on page 118.)

[Ebner 2017] Gabriel Ebner, Sebastian Ullrich, Jared Roesch, Jeremy Avigad and Leonardo de Moura. *A Metaprogramming Framework for Formal Verification*. Proc. ACM Program. Lang., vol. 1, no. ICFP, August 2017. (Cited on page 49.)

[ECG Ontology] ECG Ontology. *https://bioportal.bioontology.org/ontologies/ECG*. (Cited on page 14.)

[Edmunds 2010] Andrew Edmunds and Michael Butler. *Tool Support for Event-B Code Generation*. In WS-TBFM2010, February 2010. (Cited on page 60.)

[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 60.)

[Eggers 2011] Andreas Eggers, Nacim Ramdani, Nedialko S. Nedialkov and Martin Fränzle. *Improving SAT Modulo ODE for Hybrid Systems Analysis by Combining Different Enclosure Methods*. In Gilles Barthe, Alberto Pardo and Gerardo Schneider, Editors, Software Engineering and Formal Methods - 9th International Conference, SEFM 2011, Montevideo, Uruguay, November 14-18, 2011. Proceedings, volume 7041 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2011. (Cited on page 88.)

[Eijk 1989] P. Van Eijk and Michel Diaz, Editors. Formal description technique lotos: Results of the esprit sedos project. Elsevier Science Inc., New York, USA, 1989. (Cited on page 98.)

[Eles 2011] Colin Eles and Mark Lawford. *A Tabular Expression Toolbox for Matlab/Simulink*. In Mihaela Bobaru, Klaus Havelund, Gerard J. Holzmann and Rajeev Joshi, Editors, NASA Formal Methods, pages 494–499, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. (Cited on pages 30, 40 and 42.)

[EUROCAE 2013] EUROCAE. *ED 143 - Minimum Operational Performance Standards for Traffic Alert and Collision Avoidance System II (TCAS II)*, 2013. (Cited on pages 20 and 102.)

[EUROCONTROL 2017] EUROCONTROL. *Airborne Collision Avoidance System (ACAS) guide*, December 2017. (Cited on pages 20 and 102.)

[FAA 2012] FAA. Aircraft Landing Gear System, Chapter 13 in Aviation Maintenance Technician Handbook - Airframe Vol-1. U.S. Department of Transportation, Washington, D.C., 2012. (Cited on page 95.)

[Fallenstein 2015] Benja Fallenstein and Ramana Kumar. *Proof-Producing Reflection for HOL - With an Application to Model Polymorphism*. In Christian Urban and Xingyuan Zhang, Editors, Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, Proceedings, volume 9236 of *LNCS*, pages 170–186. Springer, 2015. (Cited on pages 48 and 49.)

[Fisher 2017] Kathleen Fisher, John Launchbury and Raymond Richards. *The HACMS program: using formal methods to eliminate exploitable bugs*. Philosophical transactions. Series A, Mathematical, physical, and engineering sciences, vol. 375, no. 2104, page 20150401, 10 2017. (Cited on page 2.)

[Fitzgerald 2008] John S. Fitzgerald, Peter Gorm Larsen and Shin Sahara. *VDMTools: advances in support for formal modeling in VDM*. SIGPLAN Notices, vol. 43, pages 3–11, February 2008. (Cited on page 82.)

[Fitzgerald 2016] John Fitzgerald, Claire Ingram and Alexander Romanovsky. *Chapter 1 Concepts of Dependable Cyber-Physical Systems Engineering: Model-Based Approaches*. In Alexander Romanovsky and Fuyuki Ishikawa, Editors, Trustworthy Cyber-Physical Systems Engineering, pages 1–22. Taylor & Francis Group, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742 CRC Press, 2016. (Cited on page 3.)

[Fowler 1999] Martin Fowler and Kent Beck. Refactoring: Improving the design of existing code. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. (Cited on pages 10 and 12.)

[FPGA] FPGA. https://www.xilinx.com. (Cited on page 59.)

[Fränzle 2007] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan and Tobias Schubert. *Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure*. J. Satisf. Boolean Model. Comput., vol. 1, no. 3-4, pages 209–236, 2007. (Cited on page 88.)

[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 88.)

[Frehse 2011] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang and Oded Maler. *SpaceEx: Scalable Verification of Hybrid Systems*. In Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings, pages 379–395, 2011. (Cited on page 88.)

[Friedenthal 2008] Sanford Friedenthal, Alan Moore and Rick Steiner. A practical guide to sysml: Systems modeling language. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008. (Cited on page 3.)

[Fürst 2014] Andreas Fürst, Thai Son Hoang, David Basin, Krishnaji Desai, Naoto Sato and Kunihiko Miyazaki. *Code Generation for Event-B*. In Elvira Albert and Emil Sekerinski, Editors, Integrated Formal Methods, pages 323–338, Cham, 2014. Springer. (Cited on pages 49 and 61.)

[Gamati 2009] Abdoulaye Gamati. Designing Embedded Systems with the SIGNAL Programming Language: Synchronous, Reactive Specification. Springer Publishing Company, Incorporated, 1st edition, 2009. (Cited on page 74.)

[Gamma 1995] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. Design patterns: Elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. (Cited on page 81.)

[Ge 2017] Ning Ge, Arnaud Dieumegard, Eric Jenn, Bruno daAusbourg and Yamine Aït-Ameur. *Formal development process of safety-critical embedded human machine interface systems*. In 11th International Symposium on Theoretical Aspects of Software Engineering, TASE 2017, Sophia Antipolis, France, September 13-15, 2017, pages 1–8, 2017. (Cited on page 98.)

[Gheyi 2004] Rohit Gheyi and Paulo Borba. *Refactoring Alloy Specifications*. Electr. Notes Theor. Comput. Sci., vol. 95, pages 227–243, 2004. (Cited on page 10.)

[Gibson 2021] J. Paul Gibson and Jean-Luc Raffy. *Modelling an E-Voting Domain for the Formal Development of a Software Product Line: When the Implicit Should Be Made Explicit*. In Yamine Aït-Ameur, Shin Nakajima and Dominique Méry, Editors, Implicit and Explicit Semantics Integration in Proof-Based Developments of Discrete Systems: Communications of NII Shonan Meetings, pages 3–18. Springer Singapore, Singapore, 2021. (Cited on page 10.)

[Gigante 2012] Gabriella Gigante and Domenico Pascarella. *Formal Methods in Avionic Software Certification: The DO-178C Perspective*. In Tiziana Margaria and Bernhard Steffen, Editors, Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies, pages 205–215, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. (Cited on page 2.)

[Gonçalves 2007] Bernardo Gonçalves, Giancarlo Guizzardi and José G. Pereira Filho. *An Electrocardiogram (ECG) Domain Ontology*. In The Second Workshop on Ontologies and Metamodels for Software and Data Engineering, 2007. (Cited on page 14.)

[Gonçalves 2011] Bernardo Gonçalves, Giancarlo Guizzardi and José G. Pereira Filho. *Using an ECG Reference Ontology for Semantic Interoperability of ECG Data*. J. of Biomedical Informatics, vol. 44, no. 1, pages 126–136, February 2011. (Cited on page 14.)

[Gr´aßler 2021] Iris Gr´aßler, Dominik Wiechel, Daniel Roesmann and Henrik Thiele. *V-model based development of cyber-physical systems and cyber-physical production systems*. Procedia CIRP, vol. 100, pages 253–258, 2021. 31st CIRP Design Conference 2021 (CIRP Design 2021). (Cited on page 1.)

[Gray 1999] Wayne D. Gray, Philippe Palanque and Fabio Paternó. *Introduction to the Special Issue on Interface Issues and Designs for Safety-Critical Interactive Systems: When There is No Room for User Error*. ACM Trans. Comput.-Hum. Interact., vol. 6, no. 4, pages 309–310, dec 1999. (Cited on pages 31 and 98.)

[Grishchenko 2018] Ilya Grishchenko, Matteo Maffei and Clara Schneidewind. *A Semantic Framework for the Security Analysis of Ethereum smart contracts*. CoRR, vol. abs/1802.08660, 2018. (Cited on page 62.)

[Griswold 1992] William G. Griswold. *Program Restructuring As an Aid to Software Maintenance*. PhD thesis, University of Washington, Seattle, WA, USA, 1992. UMI Order No. GAX92-03258. (Cited on page 10.)

[Gruber 1993] T. R. Gruber. *Towards Principles for the Design of Ontologies Used for Knowledge sharing*. In N. Guarino and R. Poli, Editors, Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Publisher's, 1993. (Cited on page 18.)

[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 page 74.)

[Halchin 2021] Alexandra Halchin. *Development of a Formal Verification Methodology for B Specifications using PERF formal toolkit. Application to safety requirements of railway systems.* PhD thesis, Institut National Polytechnique de Toulouse, Toulouse, France, 2021. (Cited on pages 75, 76, 77 and 80.)

[Hall 2008] Jon Hall, Lucia Rapanotti and Michael Jackson. *Problem Oriented Software Engineering: Solving the Package Router Control Problem*. IEEE Transactions on Software Engineering, vol. 34, no. 2, pages 226–241, 2008. (Cited on page 10.)

[Hamon 2013] Arnaud Hamon, Philippe Palanque, José Luís Silva, Yannick Deleris and Eric Barboni. *Formal Description of Multi-Touch Interactions*. In Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS '13, pages 207–216, New York, NY, USA, 2013. Association for Computing Machinery. (Cited on page 42.)

[Han 2012] K. Han, H. Kang, J. Kim and M. Choi. *Mathematical models for insulin secretion in pancreatic β-cells*. ISLETS, vol. 4, pages 94–107, 2012. (Cited on page 53.)

[Harrison 1990] M. Harrison and H. Thimbleby, Editors. Formal methods in human-computer interaction. Cambridge University Press, USA, 1990. (Cited on pages 31 and 98.)

[Harrison 2014] Michael D. Harrison, Paolo Masci, Jose C. Campos and Paul Curzon. *Demonstrating that medical devices satisfy user related safety requirements*. In 4th International Symposium on Foundations of Healthcare Information Engineering and Systems (FHIES2014), 2014. (Cited on page 98.)

[Harrison 2017] Michael D. Harrison, Paolo Masci, José Creissac Campos and Paul Curzon. *Verification of User Interface Software: The Example of Use-Related Safety Requirements and Programmable Medical Devices*. IEEE Trans. Human-Machine Systems, vol. 47, no. 6, pages 834–846, 2017. (Cited on pages 98 and 119.)

[Hegedus 2021] Csaba Hegedus, Pál Varga and Attila Frankó. *A DevOps Approach for Cyber-Physical System-of-Systems Engineering through Arrowhead*. In 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 902–907, 2021. (Cited on page 1.)

[Heitmeyer 1996] Constance L. Heitmeyer, Ralph D. Jeffords and Bruce G. Labaw. *Automated Consistency Checking of Requirements Specifications*. ACM Trans. Softw. Eng. Methodol., vol. 5, no. 3, pages 231–261, July 1996. (Cited on page 32.)

[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 32.)

[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 32.)

[Henry 1990] Tyson R. Henry, Scott E. Hudson and Gary L. Newell. *Integrating Gesture and Snapping into a User Interface Toolkit*. In Proceedings of the 3rd Annual ACM SIGGRAPH Symposium on User Interface Software and Technology, UIST '90, pages 112–122, New York, NY, USA, 1990. ACM. (Cited on page 98.)

[Henzinger 1997] Thomas A. Henzinger, Pei-Hsin Ho and Howard Wong-Toi. *HyTech: A Model Checker for Hybrid Systems*. International Journal on Software Tools for Technology Transfer, vol. 1, no. 1-2, pages 110–122, 1997. (Cited on pages 32 and 88.)

[Henzinger 2000] Thomas A. Henzinger. *The Theory of Hybrid Automata*. In M. Kemal Inan and Robert P. Kurshan, Editors, Verification of Digital and Hybrid Systems, volume 170 of *NATO ASI Series*, pages 265–292. Springer Berlin Heidelberg, 2000. (Cited on page 88.)

[Hildebrandt 2018] C. Hildebrandt, S. TÃ¶rsleff, B. Caesar and A. Fay. *Ontology Building for Cyber-Physical Systems: A domain expert-centric approach*. In 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), pages 1079–1086, 2018. (Cited on page 4.)

[Hildenbrandt 2018] Everett Hildenbrandt, Manasvi Saxena, Nishant Rodrigues, Xiaoran Zhu, Philip Daian, Dwight Guth, Brandon Moore, Daejun Park, Yi Zhang, Andrei Stefanescu and Grigore Rosu. *KEVM: A Complete Formal Semantics of the Ethereum Virtual Machine*. In 2018 IEEE 31st Computer Security Foundations Symposium (CSF), pages 204–217, 2018. (Cited on page 62.)

[Hirai 2017] Yoichi Hirai. *Defining the Ethereum Virtual Machine for Interactive Theorem Provers*. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y.A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore and Markus Jakobsson, Editors, Financial Cryptography and Data Security, pages 520–535, Cham, 2017. Springer International Publishing. (Cited on page 62.)

[Hoare 1985] C. A. R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985. (Cited on page 2.)

[Holzmann 1997] Gerard J. Holzmann. *The Model Checker SPIN*. IEEE Trans. Softw. Eng., vol. 23, no. 5, pages 279–295, May 1997. (Cited on pages 2 and 17.)

[IEEE-Std-1474.1 1999] IEEE-Std-1474.1. *IEEE Standard for Communications-Based Train Control (CBTC) Performance and Functional Requirements*, 1999. (Cited on page 80.)

[Iliasov 2010] Alexei Iliasov, Elena Troubitsyna, Linas Laibinis and Alexander Romanovsky. *Patterns for Refinement Automation*. In FrankS. de Boer, MarcelloM. Bonsangue, Stefan Hallerstede and Michael Leuschel, Editors, Formal Methods for Components and Objects, volume 6286 of *Lecture Notes in Computer Science*, pages 70–88. Springer Berlin Heidelberg, 2010. (Cited on page 23.)

[Iliasov 2016] Alexei Iliasov, Paulius Stankaitis, David Adjepon-Yamoah and Alexander Romanovsky. *Rodin Platform Why3 Plug-In*. In Michael Butler, Klaus-Dieter Schewe, Atif Mashkoor and Miklos Biro, Editors, Abstract State Machines, Alloy, B, TLA, VDM, and Z, pages 275–281, Cham, 2016. Springer International Publishing. (Cited on page 91.)

[iLock] Prover iLock. https://www.prover.com/software-solutions-rail-control/prover-ilock/. (Cited on pages 2 and 32.)

[Jackson 1993] Michael Jackson and Pamela Zave. *Domain descriptions*. In Proceedings of IEEE International Symposium on Requirements Engineering, RE 1993, San Diego, California, USA, January 4-6, 1993, pages 56–64. IEEE, 1993. (Cited on pages 3, 4, 9 and 18.)

[Jackson 1995] Michael A. Jackson. Software requirements and specifications - a lexicon of practice, principles and prejudices. Addison-Wesley, 1995. (Cited on page 10.)

[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 page 2.)

[Jeannin 2015] J. B. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki and A. Platzer. *Formal verification of ACAS X, an industrial airborne collision avoidance system*. In 2015 International Conference on Embedded Software (EMSOFT), pages 127–136, Oct 2015. (Cited on pages 2 and 32.)

[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 2 and 105.)

[Jiang 2015] Yanjun Jiang. A Tabular Expression to Event-B Language Transformation Tool. Master's thesis, McMaster University, Hamilton, ON, Canada, 2015. (Cited on pages 28, 30, 42 and 106.)

[Jifeng 1994] He Jifeng. *From CSP to Hybrid Systems*. In A. W. Roscoe, Editor, A Classical Mind, pages 171–189. Prentice Hall International (UK) Ltd., 1994. (Cited on page 88.)

[Jones 1986] Clifford B. Jones. Systematic Software Development using VDM. Prentice Hall International Series in Computer Science. Prentice Hall, 1986. (Cited on pages 2, 23, 32 and 49.)

[Jones 1991] Clifford B. Jones, K. D. Jones, Peter Alexander Lindsay and Richard C. Moore. Mural - a formal development support system. Springer, 1991. (Cited on page 49.)

[Julian 2016] Kyle D. Julian, Jessica Lopez, Jeffrey S. Brush, Michael P. Owen and Mykel J. Kochenderfer. *Policy compression for aircraft collision avoidance systems*. In 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), pages 1–10, 2016. (Cited on page 119.)

[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 32, 81 and 105.)

[Kelly 1998] Tim Kelly. *Arguing Safety – A Systematic Approach to Managing Safety Cases*. PhD thesis, University of York, September 1998. (Cited on pages 81 and 120.)

[Kelly 2001] T. P. Kelly and J. A. Mcdermid. *A systematic approach to safety case maintenance*. Reliability Engineering & System Safety, vol. 71, no. 3, pages 271–284, March 2001. (Cited on page 81.)

[Khan 2008] M. Gabriel Khan. Rapid ECG Interpretation . Humana Press, 2008. (Cited on pages 13, 14, 15 and 16.)

[Klein 2009] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch and Simon Winwood. *seL4: Formal Verification of an OS Kernel*. In Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles, SOSP '09, pages 207–220, New York, NY, USA, 2009. ACM. (Cited on pages 2, 23 and 32.)

[Kneuper 1997] Ralf Kneuper. *Limits of formal methods*. Formal Aspects of Computing, vol. 9, no. 4, pages 379–394, 1997. (Cited on pages 2 and 32.)

[Kobayashi 2014] Tsutomu Kobayashi, Fuyuki Ishikawa and Shinichi Honiden. *Understanding and Planning Event-B Refinement through Primitive Rationales*. In Abstract State Machines, Alloy, B, TLA, VDM, and Z - 4th International Conference, ABZ 2014, Toulouse, France, June 2-6, 2014. Proceedings, pages 277–283, 2014. (Cited on page 23.)

[Kobayashi 2016] Tsutomu Kobayashi, Fuyuki Ishikawa and Shinichi Honiden. *Refactoring Refinement Structure of Event-B Machines*. In John Fitzgerald, Constance Heitmeyer, Stefania Gnesi and Anna Philippou, Editors, FM 2016: Formal Methods: 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings, pages 444–459, Cham, 2016. Springer International Publishing. (Cited on page 10.)

[Kopetz 2011] Hermann Kopetz. Real-time systems: Design principles for distributed embedded applications. Springer Publishing Company, Incorporated, 2nd edition, 2011. (Cited on page 3.)

[Krasner 1988] Glenn E. Krasner and Stephen T. Pope. *A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80*. J. Object Oriented Program., vol. 1, no. 3, pages 26–49, August 1988. (Cited on page 46.)

[Kroening 2014] Daniel Kroening and Michael Tautschnig. *CBMC – C Bounded Model Checker*. In Erika Ábrahám and Klaus Havelund, Editors, Tools and Algorithms for the Construction and Analysis of Systems, pages 389–391, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. (Cited on pages 23 and 32.)

[LabVIEW] LabVIEW. `www.ni.com/labview/`. (Cited on page 59.)

[Lahiri 2018] Shuvendu K. Lahiri, Shuo Chen, Yuepeng Wang and Isil Dillig. *Formal Specification and Verification of Smart Contracts for Azure Blockchain*. CoRR, vol. abs/1812.08829, 2018. (Cited on page 62.)

[Le 2018] Ton Chanh Le, Lei Xu, Lin Chen and Weidong Shi. *Proving Conditional Termination for Smart Contracts*. In Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts, BCC '18, pages 57–59, New York, NY, USA, 2018. Association for Computing Machinery. (Cited on page 62.)

[Lecomte 2007] Thierry Lecomte, Thierry Servat and Guilhem Pouzancre. *Formal Methods in Safety-Critical Railway Systems*. In 10th Brasilian Symposium on Formal Methods, pages 29–31, 2007. (Cited on pages 23 and 32.)

[Lecomte 2017] Thierry Lecomte, David Déharbe, Étienne Prun and Erwan Mottin. *Applying a Formal Method in Industry: A 25-Year Trajectory*. In SBMF, volume 10623 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2017. (Cited on pages 23 and 32.)

[Lecrubier 2016] Vincent Lecrubier. *A formal language for designing, specifying and verifying critical embedded human machine interfaces*. Theses, INSTITUT SUPERIEUR DE L'AERONAUTIQUE ET DE L'ESPACE (ISAE) ; UNIVERSITE DE TOULOUSE, June 2016. (Cited on page 98.)

[Lee 2006] 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 23, 32 and 105.)

[Lee 2008] Edward A. Lee. *Cyber Physical Systems: Design Challenges*. In 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), pages 363–369, 2008. (Cited on pages 1 and 2.)

[Lee 2016] Edward A. Lee. *Fundamental Limits of Cyber-Physical Systems Modeling*. ACM Trans. Cyber Phys. Syst., vol. 1, no. 1, pages 3:1–3:26, 2016. (Cited on page 1.)

[Leroy 2009] Xavier Leroy. *Formal Verification of a Realistic Compiler*. Commun. ACM, vol. 52, no. 7, pages 107–115, 2009. (Cited on pages 74 and 75.)

[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 37, 40, 42, 44, 57, 94, 96, 100, 111 and 116.)

[Li 2006] Jiaxu Li, Yang Kuang and Clinton C. Mason. *Modeling the glucose-insulin regulatory system and ultradian insulin secretory oscillations with two explicit time delays*. Journal of Theoretical Biology, vol. 242, no. 3, pages 722–735, 2006. (Cited on pages 54 and 55.)

[Li 2017] Xin Li, Sonia Bilbao, Tamara Martín-Wanton, Joaquim Bastos and Jonathan Rodriguez. *SWARMs Ontology: A Common Information Model for the Cooperation of Underwater Robots*. Sensors, vol. 17, no. 3, 2017. (Cited on page 120.)

[Liu 1995] Shaoying Liu, Victoria Stavridou and Bruno Dutertre. *The practice of formal methods in safety-critical systems*. Journal of Systems and Software, vol. 28, no. 1, pages 77–87, 1995. (Cited on page 119.)

[Liu 2010] Jiang Liu, Jidong Lv, Zhao Quan, Naijun Zhan, Hengjun Zhao, Chaochen Zhou and Liang Zou. *A Calculus for Hybrid CSP*. In Kazunori Ueda, Editor, Programming Languages and Systems - 8th APLAS Symposium, volume 6461 of *LNCS*, pages 1–15. Springer, 2010. (Cited on page 88.)

[Lukina 2018] Anna Lukina, Ashish Tiwari, Scott A. Smolka, Lukas Esterle, Junxing Yang and Radu Grosu. *Resilient Control and Safety for Cyber-Physical Systems*. In 2018 IEEE Workshop on Monitoring and Testing of Cyber-Physical Systems (MT-CPS), pages 16–17, 2018. (Cited on page 120.)

[Magnaudet 2018] Mathieu Magnaudet, Stéphane Chatty, Stéphane Conversy, Sébastien Leriche, Celia Picard and Daniel Prun. *Djnn/Smala: A Conceptual Framework and a Language for Interaction-Oriented Programming*. Proc. ACM Hum.-Comput. Interact., vol. 2, no. EICS, pages 12:1–12:27, June 2018. (Cited on page 42.)

[Maibaum 2008] Tom Maibaum and Alan Wassyng. *A Product-Focused Approach to Software Certification*. Computer, vol. 41, no. 2, pages 91–93, 2008. (Cited on pages 5 and 81.)

[Maiden 1993] Neil A. M. Maiden and Alistair G. Sutcliffe. *Requirements engineering by example: an empirical study*. In Proceedings of IEEE International Symposium on Requirements Engineering, RE 1993, San Diego, California, USA, January 4-6, 1993, pages 104–111. IEEE Computer Society, 1993. (Cited on pages 31 and 97.)

[Maisel 2006] William H. Maisel, Megan Moynahan, Bram D. Zuckerman, Thomas P. Gross, Oscar H. Tovar, Donna-Bea Tillman and Daniel B. Schultz. *Pacemaker and ICD Generator Malfunctions: Analysis of Food and Drug Administration Annual Reports*. JAMA, vol. 295, no. 16, pages 1901–1906, 2006. (Cited on pages 1 and 105.)

[Mammar 2006] Amel Mammar and Regine Laleau. *From a B formal specification to an executable code: application to the relational database domain*. Info. & Soft. Technology 2006, vol. 48, no. 4, 2006. (Cited on page 74.)

[Martinie 2012] Célia Martinie, Philippe A. Palanque, David Navarre and Eric Barboni. *A Development Process for Usable Large Scale Interactive Critical Systems: Application to Satellite Ground Segments*. In Marco Winckler, Peter Forbrig and Regina Bernhaupt, Editors, Human-Centered Software Engineering - 4th International Conference, HCSE 2012, Toulouse, France, October 29-31, 2012. Proceedings, volume 7623 of *Lecture Notes in Computer Science*, pages 72–93. Springer, 2012. (Cited on page 97.)

[Massink 2012] Mieke Massink, Diego Latella, Andrea Bracciali, Michael D. Harrison and Jane Hillston. *Scalable context-dependent analysis of emergency egress models*. Formal Asp. Comput., vol. 24, no. 2, pages 267–302, 2012. (Cited on page 98.)

[MathWorks 2021] The MathWorks. *Simulink User's Guide*, 2021. (Cited on pages 37 and 116.)

[MATLAB] MATLAB. `https://www.mathworks.com/`. (Cited on pages 59 and 116.)

[McCarthy 1959] John McCarthy. *LISP: A Programming System for Symbolic Manipulations*. In Preprints of Papers Presented at the 14th National Meeting of the Association for Computing Machinery, ACM '59, pages 1–4, New York, NY, USA, 1959. Association for Computing Machinery. (Cited on page 49.)

[McComb 2004] Tim McComb. *Refactoring Object-Z Specifications*. In Michel Wermelinger and Tiziana Margaria-Steffen, Editors, Fundamental Approaches to Software Engineering: 7th International Conference, FASE 2004. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004. Proceedings, pages 69–83, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. (Cited on page 10.)

[McDermid 1991] John McDermid. Software Engineer's Reference Book. CRC Press, Inc., Boca Raton, FL, USA, 1991. (Cited on page 2.)

[MD 1975] Siperstein MD. *The glucose tolerance test: a pitfall in the diagnosis of Diabetes Mellitus*. Adv Intern Med, vol. 20, pages 297–323, 1975. (Cited on page 56.)

[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 119.)

[Miller 2010] Steven P. Miller, Michael W. Whalen and Darren D. Cofer. *Software Model Checking Takes off*. Commun. ACM, vol. 53, no. 2, pages 58–64, February 2010. (Cited on pages 2 and 32.)

[Mitra 2005] Sayan Mitra and Myla Archer. *PVS Strategies for Proving Abstraction Properties of Automata*. Electronic Notes in Theoretical Computer Science, vol. 125, no. 2, pages 45–65, 2005. Proceedings of the 5th International Workshop on Strategies in Automated Deduction (Strategies 2004). (Cited on page 49.)

[Mitsch 2014] Stefan Mitsch, Jan-David Quesel and André Platzer. *Refactoring, Refinement, and Reasoning*. In Cliff Jones, Pekka Pihlajasaari and Jun Sun, Editors, FM 2014: Formal Methods: 19th International Symposium, Singapore, May 12-16, 2014. Proceedings, pages 481–496, Cham, 2014. Springer International Publishing. (Cited on page 12.)

[Moura 2021] Leonardo de Moura and Sebastian Ullrich. *The Lean 4 Theorem Prover and Programming Language*. In André Platzer and Geoff Sutcliffe, Editors, Automated Deduction – CADE 28, pages 625–635, Cham, 2021. Springer International Publishing. (Cited on page 49.)

[Muñoz 1999] César Muñoz and John Rushby. *Structural embeddings: Mechanization with method*. In International Symposium on Formal Methods, pages 452–471. Springer, 1999. (Cited on pages 48 and 49.)

[Murata 1989] T. Murata. *Petri nets: Properties, analysis and applications*. Proceedings of the IEEE, vol. 77, no. 4, pages 541–580, 1989. (Cited on page 43.)

[Myers 1993] Brad A. Myers. *Why Are Human-Computer Interfaces Difficult to Design and Implement?* Rapport technique, Carnegie Mellon University, Pittsburgh, PA, USA, 1993. (Cited on pages 31 and 97.)

[Navarre 2001a] David Navarre, Philippe Palanque, Rémi Bastide and Ousmane Sy. *A Model-Based Tool for Interactive Prototyping of Highly Interactive Applications*. In Proceedings of the 12th International Workshop on Rapid System Prototyping, RSP'01, page 136, USA, 2001. IEEE Computer Society. (Cited on page 42.)

[Navarre 2001b] David Navarre, Philippe A. Palanque, Fabio Paternò, Carmen Santoro and Rémi Bastide. *A Tool Suite for Integrating Task and System Models through Scenarios*. In 8th International Workshop on Interactive Systems: Design, Specification, and Verification (DSV-IS), pages 88–113, 2001. (Cited on page 98.)

[Ngo 2015] Van-Chan Ngo, Jean-Pierre Talpin, Thierry Gautier, Loïc Besnard and Paul Le Guernic. *Modular Translation Validation of a Full-sized Synchronous Compiler Using Off-the-shelf Verification Tools*. In Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems, SCOPES '15, pages 109–112, New York, NY, USA, 2015. ACM. (Cited on page 74.)

[Ning Ge 2017] Eric Jenn Ning Ge Arnaud Dieumegard and Laurent Voisin. *Correct-by-construction Specification to Verified Code*. Ada-Europe 2017, 2017. (Cited on page 74.)

[Nipkow 2002] Tobias Nipkow, Markus Wenzel and Lawrence C. Paulson. Isabelle/hol: A proof assistant for higher-order logic. Springer-Verlag, 2002. (Cited on pages 2, 17, 49, 75 and 116.)

[NIST 2002] NIST. *Software Errors Cost U.S. Economy $ 59.5 Billion Annually*, 2002. (Cited on page 114.)

[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 32 and 105.)

[Opdyke 1992] William F. Opdyke. *Refactoring Object-oriented Frameworks*. PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1992. UMI Order No. GAX93-05645. (Cited on pages 10 and 12.)

[Ordioni 2018] Julien Ordioni, Nicolas Breton and Jean-Louis Colaço. *HLL v.2.7 Modelling Language Specification*. Rapport technique, RATP, 2018. (Cited on page 74.)

[Organization 2011] International Standard Organization. *ISO 26262: Road Vehicles – Functional Safety*, 2011. (Cited on page 1.)

[Ostroumov 2011] Sergey Ostroumov and Leonidas Tsiopoulos. *VHDL Code Generation from Formal Event-B Models*. In 2011 14th Euromicro Conference on Digital System Design, pages 127–134, 2011. (Cited on page 60.)

[Overture] Overture. *Overture: Formal Modelling in VDM*. http://www.overturetool.org/. (Cited on pages 23 and 32.)

[Owre 1992] Sam Owre, John M. Rushby and Natarajan Shankar. *PVS: A Prototype Verification System*. In Deepak Kapur, Editor, Automated Deduction - CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, NY, USA, Proceedings, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. Springer, 1992. (Cited on pages 2, 17, 49, 98 and 116.)

[Palanque 1994] Philippe A. Palanque and Rémi Bastide. *Petri net based Design of User-driven Interfaces Using the Interactive Cooperative Objects Formalism*. In Design, Specification and Verification of Interactive Systems, Proc. of the First International Eurographics Workshop, Italy, pages 383–400, 1994. (Cited on page 98.)

[Palanque 1996] P. Palanque, R. Bastide and V. Sengès. *Validating interactive system design through the verification of formal task and system models*. In Leonard J. Bass and Claus Unger, Editors, Engineering for Human-Computer Interaction: Proceedings of the IFIP TC2/WG2.7 working conference on engineering for human-computer interaction, Yellowstone Park, USA, August 1995, pages 189–212, Boston, MA, 1996. Springer US. (Cited on page 98.)

[Palanque 1997] Philippe Palanque and Fabio Paterno. Formal methods in human-computer interaction. Springer-Verlag, Berlin, Heidelberg, 1st edition, 1997. (Cited on page 98.)

[Palanque 2009] Philippe A. Palanque, Jean-François Ladry, David Navarre and Eric Barboni. *High-Fidelity Prototyping of Interactive Systems Can Be Formal Too*. In Human-Computer Interaction. New Trends, 13th International Conference, HCI International 2009, San Diego, CA, USA, Part I, pages 667–676, 2009. (Cited on page 98.)

[Parnas 1992] David Lorge Parnas. *Tabular Representation of Relations*. Rapport technique, McMaster University, 1992. (Cited on page 23.)

[Paul van der Walt 2012] Paul van der Walt. Reflection in Agda. Master's thesis, Utrecht University, 2012. (Cited on page 49.)

[Peters 2003] George A Peters and Barbara J Peters. Automotive vehicle safety. CRC Press, 2003. (Cited on page 1.)

[Petit-Doche 2015] Marielle Petit-Doche, Nicolas Breton, Roméo Courbis, Yoann Fonteneau and Matthias Güdemann. *Formal Verification of Industrial Critical Software*. In Manuel Núñez and Matthias Güdemann, Editors, FMICS 2015, pages 1–11. Springer International, 2015. (Cited on page 74.)

[Pierra 2004] Guy Pierra. *The PLIB Ontology-Based Approach to Data Integration*. In Renè Jacquart, Editor, Building the Information Society, pages 13–18, Boston, MA, 2004. Springer US. (Cited on page 19.)

[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 88.)

[Pnueli 1999] Amir Pnueli, Ofer Shtrichman and Michael Siegel. *Translation Validation: From SIGNAL to C*. In Correct System Design, Recent Insight and Advances, pages 231–255. Springer-Verlag, 1999. (Cited on page 74.)

[Pop 2009] Sebastian Pop, Pierre Jouvelot and George André Silber. *In and Out of SSA : a Denotational Specification*. In Workshop Static Single-Assignment Form Seminar, 2009. (Cited on page 74.)

[Pulina 2010] Luca Pulina and Armando Tacchella. *An Abstraction-Refinement Approach to Verification of Artificial Neural Networks*. In Tayssir Touili, Byron Cook and Paul Jackson, Editors, Computer Aided Verification, pages 243–257, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. (Cited on page 119.)

[Pulina 2012] Luca Pulina and Armando Tacchella. *Challenging SMT Solvers to Verify Neural Networks*. AI Commun., vol. 25, no. 2, pages 117–135, apr 2012. (Cited on page 119.)

[Quesel 2016] Jan-David Quesel, Stefan Mitsch, Sarah M. Loos, Nikos Aréchiga and André Platzer. *How to model and prove hybrid systems with KeYmaera: a tutorial on safety*. Int. J. Softw. Tools Technol. Transf., vol. 18, no. 1, pages 67–91, 2016. (Cited on page 88.)

[Rathore 2020] Heena Rathore, Amr Mohamed and Mohsen Guizani. *A Survey of Blockchain Enabled Cyber-Physical Systems*. Sensors, vol. 20, no. 1, 2020. (Cited on page 121.)

[Riccobene 2004] Elvinia Riccobene and Patrizia Scandurra. *Towards an Interchange Language for ASMs*. In Wolf Zimmermann and Bernhard Thalheim, Editors, Abstract State Machines 2004. Advances in Theory and Practice, 11th International Workshop, ASM 2004, Lutherstadt Wittenberg, Germany, May 24-28, 2004. Proceedings, volume 3052 of *Lecture Notes in Computer Science*, pages 111–126. Springer, 2004. (Cited on page 48.)

[Rivera 2017] Victor Rivera, Néstor Cataño, Tim Wahls and Camilo Rueda. *Code generation for Event-B*. Int. J. Softw. Tools Technol. Transf., vol. 19, no. 1, pages 31–52, 2017. (Cited on page 61.)

[Rochard 2000] B. P. Rochard and F. Schmid. *A review of methods to measure and calculate train resistances*. Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit, vol. 214, no. 4, pages 185–199, 2000. (Cited on pages 89 and 92.)

[RTCA/DO-178B 1992] EUROCAE/ED-12B RTCA/DO-178B. *Software Considerations in Airborne Systems and Equipment Certification*, December 1, 1992. (Cited on page 2.)

[RTCA/DO-178C 011] RTCA/DO-178C. *Software Considerations in Airborne Systems and Equipment Certification*, (December 13, 2011). (Cited on page 2.)

[RTCA/DO-333 011] RTCA/DO-333. *Formal Methods Supplement to DO-178C and DO-278A* , (December 13, 2011). (Cited on page 2.)

[Rumbaugh 2004] James Rumbaugh, Ivar Jacobson and Grady Booch. Unified modeling language reference manual, the (2nd edition). Pearson Higher Education, 2004. (Cited on page 3.)

[Rushby 1993] John Rushby. *Formal Methods and the Certification of Critical Systems*. Rapport technique SRI-CSL-93-7, Computer Science Laboratory, SRI International, Menlo Park, CA, December 1993. Also issued under the title *Formal Methods and Digital Systems Validation for Airborne Systems* as NASA Contractor Report 4551, December 1993. (Cited on page 2.)

[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 page 32.)

[Ryabtsev 2009] Michael Ryabtsev and Ofer Strichman. *Translation Validation: From Simulink to C*. In Computer Aided Verification, pages 696–701, 2009. (Cited on page 74.)

[Sangiorgi 1998] Davide Sangiorgi. *On the bisimulation proof method*. Mathematical Structures in Computer Science, vol. 8, no. 5, pages 447–479, 1998. (Cited on page 79.)

[Savicks 2014] Vitaly Savicks, Michael Butler and John Colley. *Co-simulating Event-B and Continuous Models via FMI*. In Proceedings of the 2014 Summer Simulation Multiconference, SummerSim '14, pages 37:1–37:8, San Diego, CA, USA, 2014. Society for Computer Simulation International. (Cited on page 41.)

[Scacchi 2002] Walt Scacchi. *Process Models in Software Engineering*. In J.J. Marchiniak, Editor, Encyclopedia of Software Engineering, 2nd Edition. John Wiley and Sons, Inc, New York, USA, 2002. (Cited on page 32.)

[Schneider 2001] K. Schneider. *Embedding imperative synchronous languages in interactive theorem provers*. In Proceedings Second International Conference on Application of Concurrency to System Design, pages 143–154, 2001. (Cited on page 78.)

[Schneider 2015] Sigurd Schneider, Gert Smolka and Sebastian Hack. *A Linear First-Order Functional Intermediate Language for Verified Compilers*. In Christian Urban and Xingyuan Zhang, Editors, Interactive Theorem Proving, pages 344–358, Cham, 2015. Springer International Publishing. (Cited on page 74.)

[Schranz 2021] Melanie Schranz, Gianni A. Di Caro, Thomas Schmickl, Wilfried Elmenreich, Farshad Arvin, Ahmet Sekercioglu and Micha Sende. *Swarm Intelligence and cyber-physical systems: Concepts, challenges and future trends*. Swarm and Evolutionary Computation, vol. 60, page 100762, 2021. (Cited on page 120.)

[Servat 2006] Thierry Servat. *BRAMA: A New Graphic Animation Tool for B Models*. In Proceedings of the 7th International Conference on Formal Specification and Development in B, B'07, pages 274–276, Berlin, Heidelberg, 2006. Springer-Verlag. (Cited on page 108.)

[Sheard 2002] Tim Sheard and Simon Peyton Jones. *Template Meta-Programming for Haskell*. SIGPLAN Not., vol. 37, no. 12, pages 60–75, dec 2002. (Cited on page 49.)

[Shneiderman 2016] Ben Shneiderman, Catherine Plaisant, Maxine Cohen, Steven Jacobs and Niklas Elmqvist. Designing the user interface - strategies for effective human-computer interaction, 6th edition. Pearson, 2016. (Cited on page 97.)

[Silber 2007] Hanna E. Silber, Petra M. Jauslin, Nicolas Frey, Ronald Gieschke, Ulrika S. H. Simonsson and Mats O. Karlsson. *An Integrated Model for Glucose and Insulin Regulation in Healthy Volunteers and Type 2 Diabetic Patients Following Intravenous Glucose Provocations*. The Journal of Clinical Pharmacology, vol. 47, no. 9, pages 1159–1171, 2007. (Cited on pages 53 and 55.)

[Silva 2012] Renato Silva and Michael Butler. *Shared Event Composition/Decomposition in Event-B*. In Bernhard K. Aichernig, Frank S. de Boer and Marcello M. Bonsangue, Editors, Formal Methods for Components and Objects, pages 122–141. Springer, 2012. (Cited on page 49.)

[Sites 1974] Richard L. Sites. *Clean Termination of Computer Programs*. Ph.D. dissertation, Stanford University, Stanford, California, June 1974. (Cited on page 65.)

[Smith 2009] Michael Smith. Abaqus/standard user's manual, version 6.9. Dassault Systèmes Simulia Corp, United States, 2009. (Cited on page 58.)

[Snickerdoodle] Snickerdoodle. `http://krtkl.com`. (Cited on page 59.)

[Solidity Documentation 2023] Solidity Documentation. https://solidity.readthedocs.io, 2023. (Cited on pages 62, 63 and 122.)

[Sommerville 2004] Ian Sommerville. Software engineering (7th edition). Pearson Addison Wesley, 2004. (Cited on pages 32 and 39.)

[Souyris 2009] Jean Souyris, Virginie Wiels, David Delmas and Hervé Delseny. *Formal Verification of Avionics Software Products*. In Ana Cavalcanti and Dennis R. Dams, Editors, FM 2009: Formal Methods, pages 532–546, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. (Cited on pages 2 and 32.)

[Sozeau 2020] M. Sozeau, A. Anand, S. Boulier, C. Cohen, Y. Forster, F. Kunze, G. Malecha, N. Tabareau and T. Winterhalter. *The MetaCoq Project*. J. Autom. Reason., vol. 64, no. 5, pages 947–999, 2020. (Cited on page 48.)

[Spiegelhalter 2020] David Spiegelhalter. *Should We Trust Algorithms?* Harvard Data Science Review, vol. 2, no. 1, jan 31 2020. https://hdsr.mitpress.mit.edu/pub/56lnenzj. (Cited on page 119.)

[Spivey 1989] John Michael Spivey. The z notation - a reference manual. Prentice Hall International Series in Computer Science. Prentice Hall, 1989. (Cited on page 2.)

[Stankaitis 2019] Paulius Stankaitis, Alexei Iliasov, Yamine Aït Ameur, Tsutomu Kobayashi, Fuyuki Ishikawa and Alexander Romanovsky. *A Refinement Based Method for Developing Distributed Protocols*. IEEE 19th International Symposium on High Assurance Systems Engineering (HASE), pages 90–97, 2019. (Cited on page 91.)

[Steve 2009] Wright Steve. *Automatic Generation of C from Event-B*. In Workshop on Integration of Model-based Formal Methods and Tools, 2009. (Cited on page 60.)

[Stewart 2015] Gordon Stewart, Lennart Beringer, Santiago Cuellar and Andrew W. Appel. *Compositional CompCert*. SIGPLAN Not., vol. 50, no. 1, pages 275–287, January 2015. (Cited on pages 23 and 32.)

[Storey 1994] Andrew C. Storey and Howard P. Haughton. A strategy for the production of verifiable code using the B Method, pages 346–365. Springer Berlin, 1994. (Cited on page 74.)

[Strecker 2002] Martin Strecker. *Formal Verification of a Java Compiler in Isabelle*. In Andrei Voronkov, Editor, Automated Deduction—CADE-18, pages 63–77. Springer Berlin Heidelberg, 2002. (Cited on page 74.)

[Strunk 2008] Elisabeth A. Strunk and John C. Knight. *The essential synthesis of problem frames and assurance cases*. Expert Systems, vol. 25, no. 1, pages 9–27, 2008. (Cited on page 82.)

[Stump 2016] Aaron Stump. Verified functional programming in agda. Association for Computing Machinery and Morgan & Claypool, 2016. (Cited on page 49.)

[Su 2014a] Wen Su and Jean-Raymond Abrial. *Aircraft Landing Gear System: Approaches with Event-B to the Modeling of an Industrial System*. In ABZ 2014: The Landing Gear Case Study - Case Study Track, Held at the 4th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z, Toulouse, France, June 2-6, 2014. Proceedings, pages 19–35, 2014. (Cited on page 2.)

[Su 2014b] Wen Su, Jean-Raymond Abrial and Huibiao Zhu. *Formalizing hybrid systems with Event-B and the Rodin Platform*. Science of Computer Programming, vol. 94, Part 2, pages 164–202, 2014. Abstract State Machines, Alloy, B, VDM, and Z Selected and extended papers from ABZ 2012. (Cited on pages 32 and 88.)

[Su 2017] Wen Su and Jean-Raymond Abrial. *Aircraft Landing Gear System: Approaches with Event-B to the Modeling of an Industrial System*. Int. J. Softw. Tools Technol. Transf., vol. 19, no. 2, pages 141–166, April 2017. (Cited on page 47.)

[Sun 2009] Jun Sun, Yang Liu, Jin Song Dong and Jun Pang. *PAT: Towards Flexible Verification under Fairness*. In Ahmed Bouajjani and Oded Maler, Editors, Computer Aided Verification, pages 709–714, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. (Cited on page 2.)

[Sutcliffe 2020] Alistair Sutcliffe, Pete Sawyer, Gemma Stringer, Samuel Couth, Laura J. E. Brown, Ann Gledson, Christopher Bull, Paul Rayson, John A. Keane, Xiao-Jun Zeng and Iracema Leroi. *Known and unknown requirements in healthcare*. Requir. Eng., vol. 25, no. 1, pages 1–20, 2020. (Cited on pages 31 and 97.)

[Szegedy 2014] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow and Rob Fergus. *Intriguing properties of neural networks*. In Yoshua Bengio and Yann LeCun, Editors, 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, 2014. (Cited on page 119.)

[Sztipanovits 2007] Janos Sztipanovits. *Composition of Cyber-Physical Systems*. In 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07), pages 3–6, 2007. (Cited on page 5.)

[Sztipanovits 2019] Janos Sztipanovits, Xenofon Koutsoukos, Gabor Karsai, Shankar Sastry, Claire Tomlin, Werner Damm, Martin Fränzle, Jochem Rieger, Alexander Pretschner and Frank Köster. *Science of design for societal-scale cyber-physical systems: challenges and opportunities*. Cyber-Physical Systems, vol. 5, no. 3, pages 145–172, 2019. (Cited on page 2.)

[Taha 1997] Walid Taha and Tim Sheard. *Multi-Stage Programming with Explicit Annotations*. SIGPLAN Not., vol. 32, no. 12, pages 203–217, dec 1997. (Cited on page 49.)

[Tatibouët 2003] Bruno Tatibouët, Antoine Requet, Jean-Christophe Voisinet and Ahmed Hammad. Java Card Code Generation from B Specifications, pages 306–318. Springer Berlin, 2003. (Cited on page 74.)

[Tolmach 2021] Palina Tolmach, Yi Li, Shang-Wei Lin, Yang Liu and Zengxiang Li. *A Survey of Smart Contract Formal Specification and Verification*. ACM Comput. Surv., vol. 54, no. 7, July 2021. (Cited on page 62.)

[Urban 2021] Caterina Urban and Antoine Miné. *A Review of Formal Methods applied to Machine Learning*. CoRR, vol. abs/2104.02466, 2021. (Cited on page 119.)

[U.S. Food and Drug Administration 2010] U.S. Food and Drug Administration. *Total Product Life Cycle: Infusion Pump - Premarket Notification [510(k)] Submissions*, April 2010. (Cited on page 6.)

[van Lamsweerde 2009] Axel van Lamsweerde. Requirements engineering - from system goals to UML models to software specifications. Wiley, 2009. (Cited on page 10.)

[Wagner 2014] Stefan Wagner. *Scrum for Cyber-Physical Systems: A Process Proposal*. In Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, RCoSE 2014, pages 51–56, New York, NY, USA, 2014. Association for Computing Machinery. (Cited on page 1.)

[Wang 2002] P. Wang, A. Kramer, Mark N. A. and David L. Hayes. *Timing Cycles for Biventricular Pacing*. Pacing and Clinical Electrophysiology, vol. 25, pages 62–75, 2002. (Cited on page 110.)

[Wassyng 2011] Alan Wassyng, Tom Maibaum, Mark Lawford and Hans Bherer. *Software Certification: Is There a Case against Safety Cases?* In Radu Calinescu and Ethan Jackson, Editors, Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems, pages 206–227, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. (Cited on page 81.)

[Wassyng 2013] Alan Wassyng. *Though This Be Madness, Yet There Is Method in It?* In Proc. FormaliSE, pages 1–7. IEEE, 2013. (Cited on page 32.)

[Whalen 2005] Michael W. Whalen, John D. Innis, Steven P. Miller and Lucas G. Wagner. *Window Manager Analysis*, 2005. (Cited on page 2.)

[Whiteside 2011] Iain Whiteside, David Aspinall, Lucas Dixon and Gudmund Grov. *Towards Formal Proof Script Refactoring*. In James H. Davenport, William M. Farmer, Josef Urban and Florian Rabe, Editors, Intelligent Computer Mathematics: 18th Symposium, Calculemus 2011, and 10th International Conference, MKM 2011, Bertinoro, Italy, July 18-23, 2011. Proceedings, pages 260–275, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. (Cited on page 10.)

[Wirth 1971] Niklaus Wirth. *Program development by stepwise refinement*. Commun. ACM, vol. 14, pages 221–227, April 1971. (Cited on page 32.)

[Wood 2014] Gavin Wood. *Ethereum: A secure decentralised generalised transaction ledger EIP-150 REVISION*. http://gavwood.com/paper.pdf, 2014. (Cited on pages 62 and 122.)

[Woodcock 2002] Jim Woodcock and Ana Cavalcanti. *The Semantics of Circus*. In Proceedings of the 2nd International Conference of B and Z Users on Formal Specification and Development in Z and B, ZB '02, pages 184–203, Berlin, Heidelberg, 2002. Springer-Verlag. (Cited on page 2.)

[Woodcock 2009] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui and John Fitzgerald. *Formal Methods: Practice and Experience*. ACM Comput. Surv., vol. 41, no. 4, oct 2009. (Cited on page 119.)

[Wu 2021] Changshun Wu, Yliès Falcone and Saddek Bensalem. *Customizable Reference Runtime Monitoring of Neural Networks using Resolution Boxes*. CoRR, vol. abs/2104.14435, 2021. (Cited on page 119.)

[Yaghoubi Shahir 2012] Hamed Yaghoubi Shahir, Roozbeh Farahbod and Uwe Glässer. *Refactoring Abstract State Machine Models*. In John Derrick, John Fitzgerald, Stefania Gnesi, Sarfraz Khurshid, Michael Leuschel, Steve Reeves and Elvinia Riccobene, Editors, Abstract State Machines, Alloy, B, VDM, and Z: Third International Conference, ABZ 2012, Pisa, Italy, June 18-21, 2012. Proceedings, pages 345–348, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. (Cited on page 10.)

[Yamamoto 2013] Shuichiro Yamamoto and Yutaka Matsuno. *An Evaluation of Argument Patterns to Reduce Pitfalls of Applying Assurance Case*. In Proceedings of the 1st International Workshop on Assurance Cases for Software-Intensive Systems, ASSURE '13, pages 12–17. IEEE Press, 2013. (Cited on page 83.)

[Yang 2013] Zhibin Yang, Jean-Paul Bodeveix and Mamoun Filali. *A comparative study of two formal semantics of the SIGNAL language*. Frontiers of Computer Science, vol. 7, no. 5, pages 673–693, Oct 2013. (Cited on page 78.)

[Yu 1997] Eric S. K. Yu. *Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering*. In 3rd IEEE International Symposium on Requirements Engineering (RE'97), January 5-8, 1997, Annapolis, MD, USA, pages 226–235. IEEE Computer Society, 1997. (Cited on page 10.)

[Zaloshnja 2004] E Zaloshnja, T Miller, F Council and B Persaud. *Comprehensive and human capital crash costs by maximum police-reported injury severity within selected crash types*. In Annual proceedings / Association for the Advancement of Automotive Medicine. Association for the Advancement of Automotive Medicine, volume 48, pages 251–263, 2004. (Cited on page 1.)

[Zave 1997] Pamela Zave and Michael Jackson. *Four Dark Corners of Requirements Engineering*. ACM Trans. Softw. Eng. Methodol., vol. 6, no. 1, pages 1–30, 1997. (Cited on pages 3, 9 and 18.)

[Zhao 2017] Xiangfu Zhao, Zhongyu Chen, Xin Chen, Yanxia Wang and Changbing Tang. *The DAO attack paradoxes in propositional logic*. In 2017 4th International Conference on Systems and Informatics (ICSAI), pages 1743–1746, 2017. (Cited on pages 62 and 122.)

[Zhu 2020] Jian Zhu, Kai Hu, Mamoun Filali, Jean-Paul Bodeveix and Jean-Pierre Talpin. *Formal Verification of Solidity contracts in Event-B*. CoRR, vol. abs/2005.01261, 2020. (Cited on page 62.)