Anton Tarasyuk

Formal Development and Quantitative Verification of Dependable Systems

Turku Centre *for* Computer Science

TUCS Dissertations
No 156, January 2013

Formal Development and Quantitative Verification of Dependable Systems

Anton Tarasyuk

To be presented, with the permission of the Department of Information Technologies of the Åbo Akademi University, for public criticism in Auditorium Gamma on January 28, 2013, at 12 noon.

Turku Centre for Computer Science
Åbo Akademi University
Department of Information Technologies
Joukahaisenkatu 3-5, 20520 Turku
Finland

2013

Supervisors

Docent Elena Troubitsyna
Department of Information Technologies
Åbo Akademi University
Joukahaisenkatu 3-5 A, 20520 Turku
Finland

Docent Linas Laibinis
Department of Information Technologies
Åbo Akademi University
Joukahaisenkatu 3-5 A, 20520 Turku
Finland

Reviewers

Professor Dominique Méry
LORIA & Université de Lorraine
F-54506 Vandoeuvre lès Nancy
France

Professor Aad van Moorsel
School of Computing Science
Newcastle University
Newcastle upon Tyne, NE1 7RU
United Kingdom

Opponent

Professor Aad van Moorsel
School of Computing Science
Newcastle University
Newcastle upon Tyne, NE1 7RU
United Kingdom

ISBN 978-952-12-2832-2
ISSN 1239-1883

To my mother

Моей маме посвящается

It is good to have an end to journey towards; but it is the journey that matters, in the end.

Ursula K. Le Guin, "The Left Hand of Darkness"

Abstract

Modern software-intensive systems are becoming increasingly complex. Yet we are observing the pervasive use of software in such critical infrastructures as transportation systems, healthcare, telecommunication, energy production, etc. Consequently, we tend to place increasing reliance on computer-based systems and the software that they are running. The degree of reliance that we can justifiably place on a system is expressed by the notion of dependability.

Designing highly-dependable systems is a notoriously difficult task. It requires rigorous mathematical methods to prevent design errors and guarantee the correct and predictable system behaviour. However, fault prevention via rigorous engineering still cannot ensure avoidance of all faults. Hence we need powerful mechanisms for tolerating faults, i.e., the solutions that allow the system to confine the damage caused by fault occurrence and guarantee high reliability and safety. Traditionally, such dependability attributes are assessed probabilistically. However, the current software development methods suffer from discontinuity between modelling the functional system behaviour and probabilistic dependability evaluation. To address these issues, in the thesis we aim at establishing foundations for a rigorous dependability-explicit development process. In particular, we propose a semantic extension of Event-B – an automated state-based formal development framework – with a possibility of quantitative (probabilistic) reasoning. Event-B and its associated development technique – refinement – provide the designers with a powerful framework for correct-by-construction systems development. Via abstract modelling, proofs and decomposition it allows the designers to derive robust system architectures, ensure predictable system behaviour and guarantee preservation of important system properties.

We argue that the rigorous refinement-based approach to system development augmented with probabilistic analysis of dependability significantly facilitates development of complex software systems. Indeed, the proposed probabilistic extension of Event-B allows the designers to quantitatively assess the effect of different fault tolerance mechanisms and architectural solutions on system dependability. Moreover, it enables the stochastic reasoning

about the impact of component failures and repairs on system reliability and safety from the early design stages. The proposed enhanced version of the standard Event-B refinement allows the designers to ensure that the developed system is not only correct-by-construction but also dependable-by-construction, i.e., it guarantees that refinement improves (or at least preserves) the probabilistic measure of system dependability.

The proposed extension has been validated by a number of case studies from a variety of application domains, including service-oriented systems, aerospace, railways and communicating systems. We believe that the research presented in the thesis contributes to creating an integrated dependability-explicit engineering approach that facilitates rigorous development of complex computer-based systems.

Sammanfattning

Programvaruintensiva system ökar allt mera i komplexitet, men trots detta används programvara i kritisk infrastruktur inom områden såsom transport, telekommunikation, hälsovård och energiproduktion. Vi litar allt mera på datorbaserade system och deras programvara, och graden till vilken vi kan lita på ett system beskriver vi med uttrycket pålitlighet.

Att utveckla starkt pålitliga system är erkänt svårt, och kräver rigorösa matematiska metoder för att förhindra designfel och garantera att systemet uppför sig korrekt och förutsebart. Dessa metoder kan dock inte garantera att alla fel undviks, och därför behövs kraftfulla mekanismer för feltolerans, dvs. lösningar som gör det möjligt för systemet att förhindra spridningen av den skada som felet orsakade samt garantera hög pålitlighet och säkerhet. Dessa egenskaper utvärderas traditionellt probabilistiskt, men i de utvecklingsmetoder som används i dag finns det ingen kontinuitet mellan modellering av systemets funktionalitet och probabilistisk utvärdering av pålitligheten. I denna avhandling är vår målsättning att etablera grunder för en rigorös utvecklingsmetod med explicit pålitlighet. Specifikt föreslår vi ett semantiskt tillägg till Event-B – ett automerat ramverk för tillståndsbaserad formell utveckling – som tillåter ett kvantitativt (probabilistiskt) resonemang. Event-B och dess tillhörande utvecklingsteknik – precisering – ger ett kraftfullt ramverk för utveckling av system som är korrekta genom konstruktionen. Detta gör det möjligt att via abstrakt modellering, bevis och dekomposition skapa robusta systemarkitekturer och försäkra sig om att systemets uppförande är förutsebart och att viktiga egenskaper hos systemet bevaras under systemets exekvering.

Vi hävdar att utvecklingen av pålitliga komplexa system underlättas av rigorös preciseringsbaserad systemutveckling kombinerad med probabilistisk analys av pålitlighet. Det föreslagna probabilistiska tillägget till Event-B gör det möjligt att kvantitativt uppskatta effekten av olika feltoleransmekanismer och arkitekturbaserade lösningar på systemens pålitlighet. Dessutom möjliggör det redan från ett tidigt stadium stokastiskt resonemang om vilken effekt komponentfel och -reparationer har på systemets pålitlighet och säkerhet. Den föreslagna förbättrade versionen av preciseringen i Event-B gör det möjligt att säkerställa att systemet som utvecklas inte enbart är

korrekt genom konstruktionen, utan också pålitligt genom konstruktionen, dvs. garanterar att preciseringen förbättrar, eller åtminstone bibehåller det sannolikhetsbaserade måttet av systemets pålitlighet.

Det föreslagna tillägget har validerats av ett antal fallstudier inom olika applikationsområden, vilka inkluderar serviceinriktade system, flygindustri, spårtrafik och kommunikationssystem. Vi tror att forskningen som presenteras i denna avhandling medverkar till att skapa en integrerad utvecklingsteknik som har explicit pålitlighet och underlättar rigorös utveckling av komplexa datorbaserade system.

Acknowledgements

First and foremost, I would like to extend my heartfelt gratitude to both my supervisors, Docent Elena Troubitsyna and Docent Linas Laibinis. Without your continuous encouragement, never-ending patience, and invaluable support this dissertation would never have come into existence. I have been truly enjoying every day of work with you, and it is very important for me that I can call you not only my supervisors but also my dear friends.

I am most grateful to Professor Aad van Moorsel and Professor Dominique Mèry who kindly accepted to review this dissertation and whose valuable comments improved both quality and readability of its introductory part. I owe my special thanks to Professor Aad van Moorsel for also agreeing to act as an opponent at the public defence of the thesis.

I extend my sincere thanks to all members of the Department of Information Technologies at Åbo Akademi University and Turku Centre for Computer Science for providing excellent working environment and friendly atmosphere. I am especially grateful to my colleagues at the Distributed Systems Laboratory and, above all, to Professor Kaisa Sere, who has always been the driving force behind the scientific success of our laboratory. I wish to thank Petter Sandvik and Pontus Boström, who kindly agreed to help me with the Swedish version of the abstract. I also want to gratefully acknowledge my debt to all my co-authors, Elena Troubitsyna, Linas Laibinis, Inna Pereverzeva, Timo Latvala and Laura Nummila. I have learnt a great deal from our inspiring discussions and professional collaboration.

I would like to acknowledge the generous financial support provided to me by the Department of Information Technologies at Åbo Akademi University, the Academy of Finland, European Commission, Turku Centre for Computer Science and the Nokia foundation.

Last but not the least, I would like to thank my family and my friends for their love and support throughout these years. Without you my life would be less than it has become. Above all, I owe my most special thanks to my mother, Aila Vistbacka. Thank you for your love and endless trust that you gave me. This dissertation is dedicated to you.

Anton Tarasyuk
Turku, December 2012

List of original publications

I Anton Tarasyuk, Elena Troubitsyna and Linas Laibinis, Integrating Stochastic Reasoning about Critical System Properties into Modelling and Verification in Event-B. (submitted to *Science of Computer Programming*).

II Anton Tarasyuk, Elena Troubitsyna and Linas Laibinis, Towards Probabilistic Modelling in Event-B. In: Dominique Méry, Stephan Merz (Eds.), *Proceedings of 8th International Conference on Integrated Formal Methods (iFM 2010)*, LNCS 6396, 275–289, Springer, 2010.

III Anton Tarasyuk, Elena Troubitsyna and Linas Laibinis, Formal Modelling and Verification of Service-Oriented Systems in Probabilistic Event-B. In: J. Derrick et al. (Eds.), *Proceedings of 9th International Conference on Integrated Formal Methods (iFM 2012)*, LNCS 7321, 237–252, Springer, 2012.

IV Anton Tarasyuk, Inna Pereverzeva, Elena Troubitsyna, Timo Latvala and Laura Nummila, Formal Development and Assessment of a Reconfigurable On-board Satellite System, In: Frank Ortmeier, Peter Daniel (Eds.), *Proceedings of 31st International Conference on Computer Safety, Reliability and Security (SAFECOMP 2012)*, LNCS 7612, 210–222, Springer, 2012.

V Anton Tarasyuk, Elena Troubitsyna and Linas Laibinis, Quantitative Verification of System Safety in Event-B. In: Elena Troubitsyna (Ed.), *Proceedings of 3rd International Workshop on Software Engineering for Resilient Systems (SERENE 2011)*, LNCS 6968, 24–39, Springer, 2011.

VI Anton Tarasyuk, Elena Troubitsyna and Linas Laibinis, Augmenting Formal Development of Control Systems with Quantitative Reliability Assessment. In: *Proceedings of 2nd International Workshop on Software Engineering for Resilient Systems (SERENE 2010)*, ACM, 2010.

VII Anton Tarasyuk, Elena Troubitsyna and Linas Laibinis, From Formal Specification in Event-B to Probabilistic Reliability Assessment. In: *Proceedings of 3rd International Conference on Dependability (DEPEND 2010)*, 24–31, IEEE Computer Society Press, 2010.

VIII Anton Tarasyuk, Elena Troubitsyna and Linas Laibinis, Quantitative Reasoning about Dependability in Event-B: Probabilistic Model Checking Approach. In: Luigia Petre, Kaisa Sere, Elena Troubitsyna (Eds.), *Dependability and Computer Engineering: Concepts for Software-Intensive Systems*, 459–472, IGI Global, 2012.

Contents

Part I

Research Summary

Chapter 1

Introduction

Dependability is one of the most essential properties of computer-based systems. It can be defined as the ability of a system to deliver a service that can be justifiably trusted or, alternatively, as the ability of a system to avoid service failures that are more frequent or more severe than is acceptable [7, 8]. It is widely recognised that a high degree of system dependability can only be achieved if it is explicitly addressed through the entire development process. In this thesis, we aim to establish foundations for a dependability-explicit development process based on the integration of the Event-B method and quantitative analysis of dependability.

The thesis consists of two main parts. The first part contains a summary of the research reported in the thesis, while the second part consists of reprints of the original publications. The research summary is structured as follows. In Chapter 1, we introduce the concept of dependability, overview our formal framework – the Event-B method, and briefly discuss the theory of Markov processes and their application to dependability analysis. In Chapter 2, we formulate the research objectives and describe our research methods applied to achieve these objectives. Chapter 3 contains a detailed summary of the research results published in each original paper. Finally, in Chapter 4, we overview the related work in the field, give some concluding remarks and outline possible directions for our future work.

1.1 Dependability Concept

Dependability and its Attributes The notion of dependability allows for various interpretations that encompass a wide range of system properties. Nevertheless, it is commonly accepted that dependability can be characterised by the following key attributes [79, 7, 8]:

- *reliability:* the ability of a system to deliver correct service under given conditions for a specified period of time;

- *availability:* the ability of a system to be in a state to deliver correct service under given conditions at a specified instant of time;
- *maintainability:* the ability of a system to be restored to a state in which it can deliver correct service, when the maintenance is performed in accordance with stated procedures and resources;
- *safety:* the ability of a system not to incur, under given conditions, any critical failures;
- *confidentiality:* the absence of unauthorised disclosure of information;
- *integrity:* the absence of improper system alteration.

Threats to Dependability Generally, no system is absolutely dependable. Various *threats* may affect a system during its functioning and prevent it from delivering the intended service. The threats that may impair system dependability are commonly categorised into *failures*, *errors* and *faults* [7, 8].

The system is said to have a *failure* when its actual behaviour deviates from the intended one specified in design documents. This type of a threat occurs in the *external* system behaviour, i.e., it becomes observable to the user. In contrast, an *error* is usually associated with an incorrect *internal* system state and may lead to a subsequent service failure. (Yet it is not necessary that every error causes a failure.) Finally, a *fault* is an event that might cause an error. It is usually associated with a physical defect or malfunction of a system component. Traditionally, faults are classified as follows [7, 8]:

- *transient faults:* the faults that may occur and then disappear after some period of time;
- *permanent faults:* the faults that remain in the system until they are repaired;
- *intermittent faults:* the reoccurring transient faults.

Means for Dependability There are four categories of the means intended to cope with threats to dependability: *fault prevention*, *fault tolerance*, *fault removal* and *fault forecasting* [7, 8].

The main purpose of *fault prevention* (also known as *fault avoidance*) techniques is to avoid occurrence or introduction of faults during the development process. The fault prevention is a proactive process. It is performed at the design stage, i.e., before the designed system goes operational.

Fault tolerance methods are used to design a system in such a way that it is capable of functioning despite the presence of faults. Fault tolerance is usually achieved by some form of redundancy that allows the system either to mask or to detect a fault. While improving dependability, redundancy always increases the complexity and the cost of the overall system. Hence a thorough evaluation of the reliability/redundancy trade-off is an essential part of the fault tolerance methods. Fault tolerance is implemented in two main steps – *error detection* and *system recovery*. Error detection aims at identifying the presence of errors, while system recovery attempts to eliminate the detected errors (error handling) and to prevent faults from re-activation (fault handling).

Fault removal is a set of techniques for identifying and removing the causes of errors. The fault removal process is performed during the development stage as well as during the operational life of a system. The fault removal process at the development stage starts with system verification, which is followed by the diagnosis and correction steps. At the operational stage, corrective and preventive maintenance of the system is performed. Traditionally, the static and dynamic forms of verification are distinguished. The static verification includes proof-based and model checking approaches, while dynamic verification includes various testing techniques.

Fault forecasting aims at evaluation of the impact of fault occurrence and activation on the system behaviour. Such an evaluation has qualitative and quantitative aspects. The qualitative analysis helps to designate and classify failure modes as well as identify combinations of faults of components that may potentially lead to a system failure. The quantitative (or probabilistic) analysis is carried out in order to assess to what extent certain attributes of dependability are satisfied.

Formal development of dependable systems The development of dependable systems is a complex and challenging process. There is a wide range of techniques proposed to address various issues in dependable system design. The choice of techniques is guided by general-purpose and domain-specific standards (e.g., IEC 61508, ISO 26262, IEC 62278, etc.). In general, the higher the criticality of a system, the more rigorous techniques are required for its development. Traditionally, rigorous software-engineering approaches are called *formal methods* – a collection of mathematically-based methods for system modelling and verification.

In this thesis, we put forward a formal approach to development and verification of dependable systems. Within our approach, we do not only capture the nominal system behaviour but also aim at modelling fault tolerance as an intrinsic part of the system behaviour. Moreover, we demonstrate how to integrate quantitative assessment of the specified fault tolerance mechanisms and their impact on system dependability into the formal development

process. This goal requires a scalable formal technique that would allow us to cope with model complexity as well as explicitly address various dependability aspects throughout the entire development cycle. This consideration has motivated our choice of *Event-B* [2] – a rigorous, state-based method supporting the correct-by-construction system development – as a formal framework for modelling and verification of dependable systems.

1.2 Event-B Method

The B method [1, 73] is a formal approach to industrial development of dependable software. The method has been successfully applied to the development of several complex real-life applications [71, 29, 55]. Event-B is a modelling framework derived from the B method. The framework was inspired by the Action Systems formalism [11, 12, 13] – a formal approach to model parallel, distributed, and reactive systems.

Event-B employs a top-down *refinement-based approach* to system development. In Event-B, the development starts from an abstract formal specification, which is transformed into a final system implementation via a number of correctness-preserving *refinement steps*. The idea of (program) refinement was introduced by Back [10] and Morgan [63], and later has been developed into in the refinement calculus [14] – a mathematical theory based on high order logic and the lattice theory.

In Event-B, a system model is defined using the notion of an *abstract state machine* [2]. An abstract state machine encapsulates the model state, represented as a collection of model variables, and defines a set of feasible operations on this state. Therefore, it describes the dynamic part of a modelled system. A machine may also have an accompanying component, called *context*, which defines the static part of the model. In particular, it can include user-defined constants and sets as well as their properties given as a list of model axioms. A general form of an Event-B model is given in Figure 1.1.

Any machine is uniquely identified by its name Mch. The model variables, v, are declared in the **Variables** clause and initialised by the special (obligatory) event $Init$. Each model variable is strongly typed by the constraining predicate $\mathcal{I}$ given in the **Invariants** clause. The invariant clause may also contain other predicates defining properties that must be preserved during system execution.

The dynamic behaviour of a system is defined by a set of atomic events specified in the **Events** clause. Generally, an event can be defined as follows:

$$evt \mathrel{\widehat{=}} \textbf{any } a \textbf{ where } G \textbf{ then } S \textbf{ end},$$

where a is the list of local variables, the guard G is a conjunction of predicates over the local variables a and state variables v, while the action S is a

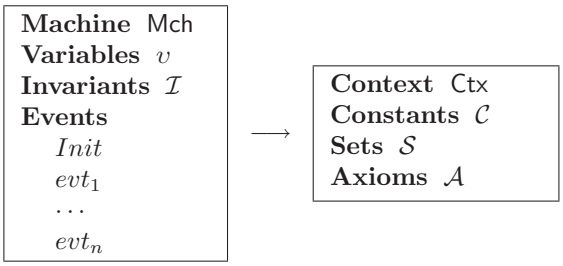


Figure 1.1: Event-B machine and context

state assignment. If the list a is empty, an event can be described simply as

$$evt \mathrel{\widehat{=}} \textbf{when } G \textbf{ then } S \textbf{ end}.$$

The occurrence of events represents the observable behaviour of the system. The guard of an event unambiguously defines the conditions under which its action can be executed, i.e., when the event is *enabled*. If several events are simultaneously enabled, any of them can be chosen for execution nondeterministically. If none of the events is enabled then the system deadlocks. It is worth to notice that any refined Event-B machine satisfies the *relative deadlock freedom* property. It means that the initial system model defines all possible deadlock states (if any) and no new deadlocks can be introduced during the refinement process.

In general, the action of an event is a parallel composition of variable assignments. The assignments can be either deterministic or nondeterministic. A deterministic assignment, $x := E(x, y)$, has the standard syntax and meaning. A nondeterministic assignment is denoted either as $x :\in Q$, where Q is a set of values, or $x :| Q(x, x', y)$, where Q is a predicate relating the initial values of the variables x and y to some final value of x (which is traditionally denoted as x'). As a result of such a nondeterministic assignment, x can get any value belonging to or satisfying Q.

The semantics of Event-B actions is defined using so called *before-after predicates* [2]. A before-after predicate describes the relationship between the system states before and after execution of an event. Figure 1.2 shows the definition of before-after predicate for all three types of Event-B assignment. Here x and y are disjoint lists of variables, while x', y' represent their values in the after-state.

The initial abstract specification describes the most essential behaviour and properties of a system. The refinement process gradually transforms the initial specification into a detailed system model. Each refinement step typically introduces new variables and events into a more abstract model. The introduced new events correspond to stuttering steps that are not visible at

Action (S)	$BA(S)$
$x := E(x, y)$	$x' = E(x, y) \wedge y' = y$
$x :\in Q$	$x' \in Q \wedge y' = y$
$x :\mid Q(x, x', y)$	$Q(x, x', y) \wedge y' = y$

Figure 1.2: Before-after predicates (BA)

the abstract level. The old, abstract model events may be also refined to reduce their nondeterminism and define computations over the new variables. Moreover, the Event-B formal development supports *data refinement*, allowing us to replace some abstract variables with their concrete counterparts. In that case, the invariant of a refined machine is extended (conjuncted) with so called *gluing invariant* that formally defines the relationship between the abstract and concrete variables.

The logical consistency of system models and correctness of the refinement process are verified by mathematical proofs. To verify correctness of a refinement step, we need to prove a number of *proof obligations* defined for the refined model. The list of proof obligations can be found in [2]. Automatic generation and demonstration of the required proof obligations is significantly facilitated by the Rodin platform [72, 3] – an integrated development environment for Event-B. The available tool support makes Event-B relevant in an industrial setting.

The refinement-based approaches to system development, in particular Event-B, have demonstrated their worth in the development of dependable systems from various domains. Traditionally, these approaches are used to ensure the functional correctness of a system. In other words, while developing a system in Event-B, we can mathematically prove that it satisfies the desired *functional* requirements. However, Event-B does not currently provide the support for quantitative verification of *non-functional* system properties that often play a crucial role in guaranteeing system dependability. This thesis is an attempt to overcome this limitation by integrating the probabilistic reasoning about dependability into the formal development by refinement. To achieve this goal, we will extensively rely on *Markov analysis* – one of the best-known methods used in probabilistic analysis of complex dependable systems.

1.3 Discrete-time Markov Processes

To enable probabilistic reasoning about an Event-B model, we aim at representing its behaviour by a *stochastic Markov process*. A stochastic process is

called a Markov process if it satisfies the *Markov property*: the future state of the process depends only on its present state and thus independent of the states that precede it [48]. Markov processes constitute an important class of stochastic processes, which are widely used in dependability engineering. In this section, we examine two types of discrete-time Markov process and briefly present their basic concepts.

Discrete-time Markov chains The most well-known Markov process is a *discrete-time Markov chain* (DTMC) [48, 35]. It is a stochastic process with a discrete state space and discrete time, i.e., the system modelled by a DTMC is observed only at discrete instances of time.

Definition 1 *Let $X_1, X_2, X_3, \ldots$ be a sequence of random variables with a finite or countable set of outcomes Σ. The stochastic process $\{X_t \,|\, t \in \mathbb{N}\}$ is a discrete-time Markov chain if it satisfies the following Markov property:*

$$\mathbf{Pr}\{X_n = \sigma_n \,|\, X_{n-1} = \sigma_{n-1}, \ldots, X_1 = \sigma_1\} = \mathbf{Pr}\{X_n = \sigma_n \,|\, X_{n-1} = \sigma_{n-1}\},$$

where $\forall j \in \mathbb{N} \cdot \sigma_j \in \Sigma$.

Remark 1 *The probability*

$$p_{ij}(n) = \mathbf{Pr}\{X_n = \sigma_j \,|\, X_{n-1} = \sigma_i\}$$

is called the n-th step transition probability for a DTMC $\{X_t \,|\, t \in \mathbb{N}\}$. If $p_{ij}(n)$ does not depend on n, i.e.,

$$p_{ij}(n) = p_{ij},$$

then $\{X_t \,|\, t \in \mathbb{N}\}$ is a time-homogeneous (or stationary) DTMC.

In this thesis, we consider only time-homogeneous Markov chains with finite state spaces. The *transition (probability) matrix* of such a DTMC is the matrix $\mathbf{P}$ with entries p_{ij}. The matrix $\mathbf{P}$ allows us to define the n-step transition probabilities of any DTMC [48, 35]:

Theorem 1 *Let $\mathbf{P}$ be the transition matrix of a DTMC. The ij-th entry $p_{ij}^{(n)}$ of the matrix $\mathbf{P}^n$ gives the probability that the Markov chain, starting in state σ_i, will be in state σ_j after n steps.*

The *initial probability vector* is the vector $\pi(0) = [p_i^{(0)}] = [\mathbf{Pr}\{X_0 = \sigma_i\}]$. Any DTMC process is fully defined by its transition matrix and initial probability vector [48, 35]:

Theorem 2 *Let* $\mathbf{P}$ *be the transition matrix of a DTMC, and let* $\pi(0)$ *be its initial probability vector. Then the probability that the chain is in state* σ_j *after* n *steps is the* j*-th entry in the vector*

$$\pi(n) = \pi(0) \cdot \mathbf{P}^n.$$

Now let us to illustrate the use of the transition matrix of a DTMC for dependability assessment by a simple example.

Example 1 In this example, we demonstrate the use of DTMC for safety assessment. Let us consider an example not from real life but from the life of the Land of Oz. Let us assume that people of the Land of Oz are anticipating some event of great importance, the success of which strongly depends on weather at the day of the event. It is known that the correlation between weather and the success of the event is measured by the following probability distribution:

	Rain	Nice	Snow
Success	0.85	1.00	0.95
Failure	0.15	0.00	0.05

Safety is generally measured by the probability that a system, under given conditions, will not cause any critical failures, i.e., the probability that no hazardous event occurs during the entire system life cycle [79]. The system, safety of which we want to analyse, is very simple. There is a single hazard associated with the system – the failure of the event, and the only hazardous situation is bad weather at the day of the event. We say that the system is safe if the event can be arranged successfully regardless of the weather conditions. Our goal is to compute the probability of this.

One rainy Monday morning it was announced that the event is scheduled on Sunday (we assume that one week is required to make the final preparations). According to [48, 35], the weather in the Land of Oz is rather depressing – people there never have two nice days in a row. "If they have a nice day, they are just as likely to have snow as rain the next day. If they have snow or rain, they have an even chance of having the same the next day. If there is change from snow or rain, only half of the time is this a change to a nice day" [35]. Clearly, in the Land of Oz the weather changes according to a three-state DTMC with the following transition matrix:

$$\mathbf{P} = \begin{array}{c} \\ \text{Rain} \\ \text{Nice} \\ \text{Snow} \end{array} \begin{array}{c} \begin{array}{ccc} \text{Rain} & \text{Nice} & \text{Snow} \end{array} \\ \begin{pmatrix} 0.5 & 0.25 & 0.25 \\ 0.5 & 0 & 0.5 \\ 0.25 & 0.25 & 0.5 \end{pmatrix} \end{array}.$$

Since we know the Monday's weather, using Theorem 2 we can forecast the weather on Sunday:

$$\begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \times \mathbf{P}^6 = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0.4 & 0.2 & 0.4 \\ 0.4 & 0.2 & 0.4 \\ 0.4 & 0.2 & 0.4 \end{pmatrix} = \begin{array}{ccc} \text{Rain} & \text{Nice} & \text{Snow} \\ (\ 0.4 & 0.2 & 0.4 \ ). \end{array}$$

Therefore, according to the formula of total probability, the probability of the event success is

$$\mathbf{Pr}\{\text{Success}\} = 0.4 \cdot 0.85 + 0.2 \cdot 1 + 0.4 \cdot 0.95 = 0.92$$

and does not depend on the today's weather. As a result, we can conclude that the considered system is safe with the probability 0.92.

In real life, the safety analysis is not so trivial. A DTMC representing the behaviour of a system is usually significantly more complex due to the integrated mechanisms for fault tolerance and repair of system components. Moreover, many safety-critical systems have more than one associated hazard.

Identification of all possible hazards associated with the system under development is an important part of the safety analysis. While reasoning about system safety, it is often convenient to treat hazards as terminating system states and define safety as an ability of a system to *terminate in a safe state*. Next we overview the theoretical results on which we rely to reason about terminating systems.

Absorbing DTMC The states of a Markov chain can be divided into equivalence classes. Two states σ_i and σ_j belong to the same equivalence class if the process can go from σ_i to σ_j (not necessarily in one step) and vice versa. In this case, we say that σ_i and σ_j *communicate*. A state of a Markov chain that does not communicate with any other state of the chain is called *absorbing*. Next we define a special class of discrete-time Markov chains called *absorbing Markov chains* [35].

Definition 2 *A state σ_i of a Markov chain is called absorbing if and only if $p_{ii} = 1$, i.e., it is impossible lo leave it. A Markov chain is called absorbing if it has at least one absorbing state and, if from each state which is not absorbing, it is possible to reach an absorbing state (not necessarily in one step).*

Definition 3 *In an absorbing Markov chain, a state which is not absorbing is called transient.*

Let us now consider an arbitrary absorbing DTMC, and let $\{\sigma_1, \ldots, \sigma_r\}$ and $\{\sigma_{r+1}, \ldots, \sigma_m\}$ be its sets of transient and absorbing states correspondingly. Then the transition matrix $\mathbf{P}$ will have the following canonical form:

$$\mathbf{P} = \begin{array}{c} \\ r \\ m-r \end{array} \begin{array}{c} \begin{array}{cc} r & m-r \end{array} \\ \left(\begin{array}{c|c} \mathbf{Q} & \mathbf{R} \\ \hline \mathbf{O} & \mathbf{I} \end{array} \right) \end{array}.$$

Here $\mathbf{Q}$ is an $r \times r$ matrix that models the process as long as it stays in transient states. $\mathbf{R}$ is a nonzero $r \times (m-r)$ matrix that models the transitions from transient to absorbing states. Finally, $\mathbf{O}$ is an $(m-r) \times r$ zero matrix and $\mathbf{I}$ is an $(m-r) \times (m-r)$ identity matrix [48].

Definition 4 *For any absorbing DTMC we define its fundamental matrix to be* $\mathbf{N} = (\mathbf{I} - \mathbf{Q})^{-1}$. *The entry* n_{ij} *of* $\mathbf{N}$ *gives the expected number of times that the process spends in the transient state* σ_j *if it is started in the transient state* σ_i.

It is well-known that for any absorbing DTMC the probability to eventually reach an absorbing state is 1. The following theorem helps us to compute the corresponding probability distribution of absorbing states [48]:

Theorem 3 *Let* b_{ij} *be the probability that an absorbing DTMC will be absorbed in the absorbing state* σ_j *if it starts in the transient state* σ_i. *Let* $\mathbf{B}$ *be the matrix with entries* b_{ij}. *Then*

$$\mathbf{B} = \mathbf{N} \cdot \mathbf{R}.$$

Theorem 3 plays an important role in the dependability analysis. In particular, by partitioning the terminating (absorbing) system states into the classes of safe and unsafe states, we can evaluate which terminating state is likely to capture the process as well as the overall probability that the process terminates in a safe system state.

Markov Decision Processes A general framework for Markov decision processes (MDP) can be considered as an extension of the one for DTMC. In addition to process states and transition probabilities, an MDP has *actions* and *rewards* [80, 69]. As before, we consider only time-homogeneous finite Markov processes.

More precisely, a Markov Decision Process is a discrete time stochastic control process $\{X_t \,|\, t \in \mathbb{N}\}$. By choosing actions, the decision maker has the opportunity to influence the behaviour of a probabilistic system as it evolves through time. For each state σ_i, there exist a feasible action space A_i, which is finite and does not depend on t. As a result of choosing action $a \in A_i$ in state σ_i:

1. the decision maker receives a reward $r(\sigma_i, a)$, and
2. the system successor state σ_j is determined by the transition probability $p_{ij}^{(a)}$.

Thus, the next state σ_j depends on the current state σ_i and the decision maker's action a. However, successor state is conditionally independent of all previous states and actions, and hence the state transitions of an MDP satisfy the Markov property. In our work, we consider only a simple form of MDP where all rewards are equal.

Generally, the goal of the decision maker is to choose a sequence of actions that causes the system to perform optimally with respect to some predetermined criterion. A *decision rule* prescribes a procedure for action selection in each state. The decision rules range from deterministic Markov to randomised history dependent, depending on how they incorporate past information and how they select actions (for more details, see, e.g., [80, 69]). A *policy* specifies the decision rule to be used in all states. It provides the decision maker with a prescription for action selection for any possible future system state. The main problem of the Markov decision theory is to find a policy for the decision maker, i.e., a function u that specifies the action $a = u(\sigma_i)$, that the decision maker will choose when the process is in the state σ_i. Note that an MDP with the defined policy behaves as a DTMC because the policy fixes the action for each state. However, in this thesis we do not focus on finding optimal policies. Instead, we allow the decision maker to choose any available action nondeterministically and aim at verification of the system behaviour for all possible policies.

1.4 Continuous-time Markov Processes

In this subsection, we consider one particular class of Markov processes – *continuous-time Markov chains* (CTMC) [68, 20]. To be precise, a continuous-time Markov chain is a *semi-Markov process* because, in addition to transition probabilities, we also take into account the probability distribution of the *sojourn time* that the process spends in its current state before the next transition occurs.

Definition 5 *The stochastic process* $\{X(t) \mid t \in \mathbb{R}^+\}$ *with finite or countable set of outcomes* Σ *is a continuous-time Markov chain if, for* $t_0 < t_1 < \cdots < t_n$, $t_i \in \mathbb{R}^+$, *it satisfies the following Markov property:*

$$\mathbf{Pr}\{X(t_n) = \sigma_n \mid X(t_{n-1}) = \sigma_{n-1}, \ldots, X(t_0) = \sigma_0\} = \mathbf{Pr}\{X(t_n) = \sigma_n \mid X(t_{n-1}) = \sigma_{n-1}\},$$

where $\forall j \in \mathbb{N} \cdot \sigma_j \in \Sigma$ *and* $\mathbb{R}^+$ *is the set of non-negative real numbers.*

Remark 2 *The transition probability function of a CTMC* $\{X(t) \mid t \in \mathbb{R}^+\}$ *is defined as*

$$p_{ij}(s, s+t) = \mathbf{Pr}\{X(s+t) = \sigma_j \mid X(s) = \sigma_i\}$$

for $s \geq 0$ *and* $t > 0$. *If* $p_{ij}(s, s+t)$ *does not depend on the moments* s *and* $s+t$ *but only on the length of interval* t, *i.e.,*

$$p_{ij}(s, s+t) = p_{ij}(t),$$

then $\{X(t) \mid t \in \mathbb{R}^+\}$ *is a time-homogeneous (or stationary) CTMC.*

As before, we consider only time-homogeneous Markov chains with finite state spaces. The transition probability function is assumed to be continuous at $t = 0$:

$$\lim_{t \to 0} p_{ij}(t) = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$

Moreover, let $\pi_j(t) = \mathbf{Pr}\{X(t) = \sigma_j\}$ be the *unconditional state probability* at time t, then

$$\pi_j(t) = \sum_{\sigma_i \in \Sigma} p_{ij}(t) \cdot \pi_i(0).$$

The transition probability function satisfies the Chapman-Kolmogorov equation [68, 20]: for any $\sigma_i, \sigma_j \in \Sigma$ and $s, t \in \mathbb{R}^+$

$$p_{ij}(s+t) = \sum_{\sigma_k \in \Sigma} p_{ik}(s) \cdot p_{kj}(t).$$

Similarly to the discrete-time setting, we say that two states σ_i and σ_j communicate if there exist such numbers s and t that $p_{ij}(t) > 0$ and $p_{ji}(s) > 0$. The definitions of absorbing and transient states are the same as for a DTMC.

Continuous time is more difficult to reason about because there is no equivalent to the one-step transition matrix of a DTMC. Indeed, in a CTMC, the transition probability is the function of elapsed time, not of the number of elapsed steps. The role of one-step probabilities in the continuous-time setting play *transition rates* (*or transitions intensities*):

$$q_i = \lim_{t \to 0} \frac{1 - p_{ii}(t)}{t} \qquad \text{and} \qquad q_{ij} = \lim_{t \to 0} \frac{p_{ij}(t)}{t}.$$

For $i \neq j$, q_{ij} is called the transition rate from (the current) state σ_i to (a successor) state σ_j.

The *transition rate matrix* (also called the *intensity matrix* or the *infinitesimal generator matrix*) of a CTMC is the matrix $\mathbf{Q}$ with entries q_{ij} where

$$q_{ii} = -q_i = -\sum_{j \neq i} q_{ij}.$$

Usually, while formulating a CTMC, we can define the transition rate matrix $\mathbf{Q}$. Therefore, to find the transition probabilities

$$p_{ij}(t) = \mathbf{Pr}\{X(s+t) = \sigma_j \mid X(s) = \sigma_i\},$$

we need to find the solutions of the forward Kolmogorov differential equations (for time-homogeneous CTMC)[1] [68, 20]:

$$\frac{dp_{ij}(t)}{dt} = -q_j \cdot p_{ij}(t) + \sum_{k \neq j} p_{ik}(t) \cdot q_{kj}.$$

The differential equations for the state probabilities $\pi_j(t)$ are

$$\frac{d\pi_j(t)}{dt} = -q_j \cdot \pi_j(t) + \sum_{k \neq j} \pi_k(t) \cdot q_{kj}. \tag{1.1}$$

Now let us to illustrate how the Kolmogorov equations can be used for reliability assessment.

Example 2 Let us consider a simple redundant system that consists of two identical repairable components. The system reaches the failure state when both components have failed. No component repair is possible in this state. Moreover, we assume that initially both components are operational.

Let λ and μ be correspondingly the failure and repair rates of each individual component, and let σ_i, for $i \in 0..2$, be the system state where i components have failed. Then σ_0 is the initial system state, while σ_2 is the unique absorbing state of the system. Overall, the system behaves as it is shown in Figure 1.3.

In engineering, reliability is measured by the probability that a system is able to deliver correct service under given conditions for a certain period of time $[0, t]$ [79, 64].

To assess the reliability of the systems modelled by absorbing Markov processes, we usually consider the process behaviour up to the moment that it enters the set of absorbing states. Therefore, reliability of our repairable

[1] Or, equally, the backward Kolmogorov equations that we omit in this brief overview of Markov processes

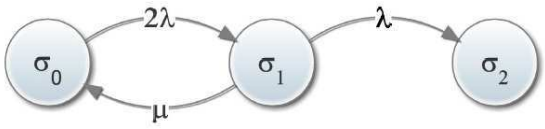


Figure 1.3: Reliability of a repairable system: state transition diagram

system can be defined as $R(t) = 1 - \pi_2(t)$. We will use the differential equations (1.1) to find it. Thus, $\pi_0(0) = 1$, $\pi_1(0) = \pi_2(0) = 0$, and

$$\begin{aligned}
\frac{d\pi_0(t)}{dt} &= -2\lambda\pi_0(t) + \mu\pi_1(t) \\
\frac{d\pi_1(t)}{dt} &= 2\lambda\pi_0(t) - (\lambda + \mu)\pi_1(t) \\
\frac{d\pi_2(t)}{dt} &= \lambda\pi_1(t)
\end{aligned}$$

To find the solutions, we can apply the Laplace transform to both sides of all three equations. From the reduced equations, we obtain the Laplace transform $F_2(s)$ of $\pi_2(s)$:

$$F_2(s) = \frac{2\lambda^2}{s^3 + (3\lambda + \mu)s^2 + 2\lambda^2 s}.$$

Finally, after inverting the Laplace transform, we can obtain $\pi_2(t)$ and then derive the reliability function:

$$R(t) = \frac{s_1 e^{s_2 t} - s_2 e^{s_1 t}}{s_1 - s_2}, \text{ where } s_{1,2} = \frac{-(3\lambda + \mu) \pm \sqrt{\lambda^2 + 6\lambda\mu + \mu^2}}{2}.$$

Provided the numerical values of λ and μ are given, we can use the obtained formula to compute the system reliability at any moment t.

Finally, let us consider the distribution of the state sojourn times of a homogeneous CTMC in detail. The Markov property is the memoryless property of a stochastic process. It is a well-known fact that the *exponential distribution* is the only continuous distribution that satisfies the memoryless property. Therefore, the random variable Y_i, denoting the sojourn time in state σ_i, is exponentially distributed with parameter q_i, i.e.,

$$\mathbf{Pr}\{Y_i \geq t\} = \begin{cases} 1 - e^{-q_i t}, & t \geq 0 \\ 0, & t < 0 \end{cases} \qquad \text{and} \qquad \mathbf{E}[Y_i] = \frac{1}{q_i}.$$

Since $q_i = \sum_{j \neq i} q_{ij}$, the parameter of Y_i is cumulated by all the transition rates outgoing from σ_i.

1.5 Markov Analysis in Dependability Engineering

Markov analysis is one of the most powerful methods in dependability engineering. It allows the developer to reduce the quantitative analysis of complex, dynamic and highly distributed systems to the mathematically well-established state modelling problem.

The main advantage of Markov analysis is that it provides the means for analysis of *repairable systems*. Markov models fully cover modelling of all three types of system fault given in Section 1.1. Moreover, the Markov analysis techniques enable dependability analysis of the systems with strong dependencies between failures of multiple components or between component failures and failure rates (probabilities). In contrast, many of widely used techniques in dependability engineering, e.g., Fault Tree Analysis [78], often require the system components to be totally independent. The Markov analysis techniques are also well suited to handle rare events and allow the developer to perform the analysis of such events within a reasonable amount of time.

The application of Markov analysis to dependability assessment is extensively covered in the literature, see, e.g., [79, 64]. Generally, it consists of the following three major steps:

1. Defining the system state space and partitioning it into two (disjoint) classes of *operational* and *non-operational* states. The operational states are those where the system is functioning properly, i.e., is capable to provide the intended services. The remaining states are non-operational. In an absorbing Markov process, operational and non-operational system states usually correspond to, respectively, the transient and absorbing states of the process.

2. Modelling of all possible state transitions together with their probabilities (rates). To accomplish this, the developer has to identify the cause (or condition) of each transition, e.g., a failure of a component, a repair of a component, some monitoring action, etc., as well as the dependencies between different components and their state transitions.

3. Computation of the model solutions. Depending on the type of an analysed property, the *steady-state* (long-run) and *transient* (time-dependent) analysis techniques are distinguished [20]. The steady-state analysis results in the steady-state (or average) probabilities that the system will be in a certain state after a long time interval. This type of analysis is usually much easier to perform and, in many practical cases, it provides adequate accuracy. On the other hand, the transient analysis results in the exact probabilities that the system

will be in a certain state at a specific time instant. The transient analysis is usually performed for highly critical systems.

As any other method in dependability engineering, Markov analysis also has certain disadvantages. Its main drawback is the state explosion problem – the exponential growth of number of states as the size of a modelled system increases.

Thus, obtaining symbolic solutions (for the state probabilities as functions of time) of Kolmogorov equations by taking and inverting their Laplace transforms is realistic for only small case-studies because the computational complexity of the method drastically increases together with the cardinality of the model state space. The same problem arises in the discrete-time setting. Indeed, the matrix methods are traditionally used for dependability assessment of discrete-time Markov models. However, the application of matrix methods to analysis of large-scale models becomes infeasible. Instead, the numeric, simulation and model-checking techniques are often applied for reliability analysis of Markov models. Most of these methods to some extent depend on the theory of Markov processes discussed in Sections 1.3 and 1.4.

In this thesis, we are mostly concerned with the transient analysis of system reliability and safety, except for some cases of the system safety assessment. To bridge the gap between the refinement-based development in Event-B and the quantitative verification of system reliability and safety, in the next chapter we show how to augment Event-B models with probabilities and represent the behaviour of a modelled system by a Markov process.

Integrating dependability engineering and formal modelling According to the surveys of the industrial use of formal methods [81, 19], the main area of their application is the development of dependable systems. The transport sector is the largest single application domain (16 of 62 considered industrial projects are related to the transport sector). Together with such critical areas as defence, healthcare, nuclear and financial sectors, it makes more than 50% of all the projects that have employed formal techniques. Moreover, the surveys also show that the popularity of formal methods, especially their use at early stages of system specification and design, is gradually increasing. Since Markov analysis remains one of the main methods in dependability engineering, there is a clear need for integrating it into formal approaches to system development. This has motivated the research presented in this thesis.

Chapter 2

Probabilistic Reasoning about Event-B Models

2.1 Research Objectives

Formal development in Event-B allows us to ensure that a resulting detailed specification adheres to its abstract counterpart. In other words, it guarantees that the services provided by the system are functionally correct with respect to its specification. However, in the current process of refinement, the non-functional system requirements, in particular dependability attributes, are usually abstracted away. This deprives the designers of a common semantic model that would allow them to evaluate the impact of the chosen design decisions on system dependability. In engineering, this impact is usually measured by probability. More precisely, while developing any kind of critical computer system, the developer has to assess to what extent the system under development satisfies its dependability requirements. Moreover, it is important to perform such kind of analysis at the early stages of development. Indeed, postponing the system dependability evaluation to the later development stages can lead to a major system redevelopment if dependability requirement are not met.

To enable quantitative verification of an Event-B model, we aim at representing the Event-B specification of the functional system behaviour by a Markov process. To achieve this, we introduce a new operator into the Event-B specification language – the *quantitative probabilistic choice*. Furthermore, we show how the behaviour of probabilistically-enriched Event-B models can be mapped to various kinds of Markov processes.

To facilitate dependability-explicit development in the probabilistic Event-B, we *strengthen the notion of Event-B refinement* by requiring that a refined model, besides being a proper functional refinement of its more abstract counterpart, also satisfies a number of quantitative constraints. These con-

straints ensure that the refined model improves (or at least preserves) the current probabilistic measures of system dependability attributes. In our work, these additional constraints are usually derived from the fundamental properties of Markov processes.

To validate the proposed approaches, we have conducted a number of case studies spanning over different types of software system, e.g., control and monitoring systems, service-oriented systems, etc. The case studies include, in particular, formal development and quantitative assessment of a fault tolerant satellite system (see Paper IV) and formal modelling together with integrated safety analysis of a radio-based railway crossing controller (see Paper V). Since a part of this thesis was carried out within the European Commission Information and Communication Technologies FP7 project DEPLOY (Industrial Deployment of System Engineering Methods Providing High Dependability and Productivity) [46], some of the considered case studies have been provided by the project industrial partners or inspired by our joint work in the project.

2.2 Discrete-time Models

Next we outline our approach to formal modelling and verification of discrete-time systems. We first demonstrate how to extend the language and semantics of Event-B in such a way that the extended models would enable dependability analysis using the theory of Markov processes. We accomplish this task by first introducing the *quantitative probabilistic choice* operator, denoted $\oplus|$, into the Event-B modelling language. The operator notation coincides with that of the *qualitative* probabilistic assignment introduced by Hallerstede and Hoang in [39]. However, its semantics is different in a sense that the quantitative choice operator contains precise probabilistic information about how likely a particular choice should be made.

The new probabilistic choice operator can be introduced into a model to replace a nondeterministic choice (assignment) in the event actions. It has been shown that any probabilistic choice statement always refines its demonic nondeterministic counterpart [60]. Hence such an extension is not interfering with the established refinement process.

For instance, an event nondeterministically modelling a failure of a system component

$$evt \mathrel{\widehat{=}} \textbf{when } G \textbf{ then } failure :\in BOOL \textbf{ end} \tag{2.1}$$

can be refined by a probabilistic one as follows

$$evt \mathrel{\widehat{=}} \textbf{when } G \textbf{ then } failure \oplus| \; TRUE \mathbin{@} p; FALSE \mathbin{@} 1-p \textbf{ end}, \tag{2.2}$$

where $p \in [0, 1]$ is the probability of failure occurrence. The variable *failure* can be understood as a discrete random variable with two possible outcomes

and the probability mass function defined by the right hand side of the $\oplus|$ operator. Generally, while refining a nondeterministic assignment over the set of possible values Q, one can introduce a probabilistic choice with the same set of outcomes Q. In practice, the set Q is usually finite, though it can also be countable. Clearly, when Q is a singleton set, $\oplus|$ becomes the standard deterministic assignment.

After refining all the nondeterministic assignments of an Even-B model by their probabilistic counterparts, we can consider the resulting model as a Markov process – a DTMC or a simple form of a MDP, depending on the presence of a nondeterministic choice between simultaneously enabled events in the model.

In the previous section, we defined reliability as the probability that a system $\mathcal{X}$ is able to deliver correct service under given conditions for a certain period of time $[0, t]$. Formally, it can be specified as

$$R(t) = \mathbf{Pr}\{\mathcal{X} \textit{ not failed over time } [0, t]\}.$$

To verify such a time-bounded reachability property, we also need to introduce into Event-B the notion of time, which is not explicitly supported by the framework at the moment. To achieve this, we focus on the modelling of systems that exhibit a *cyclic behaviour*, i.e., the systems that iteratively execute a predefined sequence of steps. Typical representatives of cyclic systems are control and monitoring systems. For instance, one iteration of a control system usually includes reading the sensors that monitor the controlled physical processes, processing the obtained sensor values, and finally setting the actuators according to a predefined control algorithm. In principle, the system could operate in this way indefinitely long. However, unforeseen conditions in the operating environment or component failures may affect the normal system functioning and lead to a shutdown.

Once the desired structure and the control flow requirements of cyclic systems are formally defined, we can can use the notion of an iteration of a cyclic system as a *discrete unit of time* defining a unified time scale for every refined Event-B model. Moreover, we model cyclic systems in such a way that, while reasoning about system reliability, it is sufficient to consider only the initial and final states of each system iteration. Such an approach allows us to partition the system state space into two sets of *observable* and *unobservable* states (see Paper I for more details). The main advantage of the achieved partitioning is that it significantly reduces the size of the model. On the other hand, this complicates computation of the transition probabilities of the (reduced) underlying Markov process.

Furthermore, to model failures of the system components, we distinguish between the *operational* and *non-operational* observable system states, which directly correspond to the transient and absorbing states of the underlying Markov process. If the underlying Markov model is a DTMC, we

can rely on its probability transition matrix (Section 1.3) to evaluate the system reliability. If the underlying stochastic process is a MDP, we have to assess the worst case scenario reliability over all possible policies, i.e., to find the policy that minimises the number of steps leading to absorption.

An introduction of the quantitative probabilistic choice (2.2) allows us to assess not only system reliability but also its safety. In this thesis, we consider safety in a context of the system behaviour in a hazardous situation. Hazard – a potentially dangerous situation – may occur as a result of certain combinations of component failures. One of the developers' goals is to identify all possible hazardous situations and specify the expected system behaviour in such situations. The main advantage of the approach we present in the thesis is that the safety assessment becomes an intrinsic part of formulating and verifying Event-B safety invariants, i.e., the invariants specifying the required (safe) system behaviour at each particular execution stage (see Paper V for more details).

2.3 Continuous-time Models

We will now briefly describe our approach to formal modelling and verification of continuous-time systems. The continuous-time models can be especially relevant when the designer needs to make an assumption about explicit duration of system activities, which are represented by events of an Event-B model in our case.

By defining the probabilistic choice operator in the continuous-time setting, we augment possible state transformations with constant transition rates $\lambda_i \in \mathbb{R}^+$. In the continuous-time setting, the event (2.1) modelling a faulty component can be refined as follows

$$evt \mathrel{\widehat{=}} \textbf{when } G \textbf{ then } \textit{failure} \oplus| \; TRUE \mathbin{@} \lambda_1; FALSE \mathbin{@} \lambda_2 \textbf{ end}. \quad (2.3)$$

Similar to the case of discrete time, the right hand side of the assignment may have more than two outcomes. Moreover, according to the theory of CTMC briefly outlined in Section 1.4, $\sum_i \lambda_i$ is the parameter of *the exponentially distributed sojourn time* that the system spends in the current state before it transits to the next one. In such a way, we can replace a nondeterministic choice between the possible successor states by the probabilistic choice associated with the (exponential) race condition. The probabilistic assignment in (2.3) can be then considered as an analogue of that of (2.2), where

$$p = \frac{\lambda_1}{\lambda_1 + \lambda_2}.$$

It is important to note that, in the continuous-time setting, any two simultaneously enabled events participate in the same race condition. This

means that, after refining all nondeterministic assignments of an Even-B model by their probabilistic counterparts, we completely eliminate demonic nondeterminism from the model. As s result, such a probabilistically augmented Event-B machine becomes a CTMC. The transient methods for reliability assessment of the systems modelled by CTMC are discussed in Sections 1.4 and 1.5. However, for industrial-size models, obtaining the system reliability by applying and inverting the Laplace transform of the corresponding Kolmogorov equations is an extremely computationally extensive task. Often it becomes too complex or even unfeasible to solve it analytically. Instead, for probabilistic analysis of Event-B models, both discrete- and continuous-time, it is convenient to rely on probabilistic model checking techniques [16] that we discuss later.

2.4 Extending the Notion of Refinement

Quantitative refinement The formal development of dependable systems can be facilitated by quantitative evaluation of possible design alternatives conducted at early design stages. To achieve this, we also have to extend our formal development technique – refinement. Specifically, we have to strengthen the notion of refinement by additionally requiring that a refined system model preserves or improves the probabilistic measures of the desired dependability attributes.

Let $\mathcal{P}$ denote the probabilistic measure of some dependability attribute that depends on time t. Then, for fully probabilistic discrete-time (continuous-time) models, i.e., models without nondeterminism, we can strengthen the definition of Event-B refinement in a following way:

Definition 6 *Let $\mathcal{M}_a$ and $\mathcal{M}_c$ be two probabilistic Event-B models. We say that $\mathcal{M}_c$ is a valid refinement of $\mathcal{M}_a$ if and only if*

1. *$\mathcal{M}_c$ is an Event-B refinement of $\mathcal{M}_a$ ($\mathcal{M}_a \sqsubseteq \mathcal{M}_c$), and*

2. *$\forall t \in \mathbb{N}\,(\mathbb{R}^+) \cdot \mathcal{P}_a(t) \leq \mathcal{P}_c(t)$.*

Remark 3 *For discrete-time models with nondeterminism (i.e., MDP models), in the second refinement condition of Definition 6 we have to evaluate the minimum probability $\mathcal{P}$ over all possible system behaviours.*

In this thesis, we usually consider $\mathcal{P}$ to be system reliability and responsiveness, thus Definition 6 essentially means that the refined model $\mathcal{M}_c$ is at least as reliable (responsive) as the abstract model $\mathcal{M}_a$. Moreover, we also define the notion of *partial* quantitative refinement, where the second condition of Definition 6 does not hold everywhere but for a finite time interval

of the length T, i.e., $\forall t \in [0, T]$. While the partial refinement is a significantly weaker property, it nevertheless might be vital when the intended operational time of the system under development does not exceed T.

Even though our definition of the quantitative refinement looks quite natural, it is important to check that it is consistent with the traditional Event-B refinement process. Indeed, this process allows the developer to introduce new events to a refined model. In the continuous-time setting, after probabilistic augmentation, execution of each such event takes some time, which means that the execution time of a refined system can be significantly longer than the execution time of its more abstract predecessor. In other word, it means that the abstract and refined models have different time scales. In this situation, the proposed extended definition of refinement may become confusing. To avoid this, we often need to find acceptable trade-offs between certain system characteristics. For instance, if $\mathcal{P}$ is the reliability of a system then, during the refinement process, we should not only guarantee that $\mathcal{P}$ is not decreasing, but also that the system performance remains at the acceptable level. There is no such a problem in the discrete-time setting as we usually strictly define an iteration of the execution cycle as a unit of time and guarantee that it persists as such during the refinement process. In other words, all new model events are defined on unobservable system states within an iteration (see Paper I for more details).

2.5 Means for Modelling and Verification

Complexity of modern computer systems puts a great emphasis on tools for automatic analysis and verification of models. Next we overview the main platforms for modelling and verifying dependable systems that we have used in the thesis.

The Rodin platform [72, 3] provides the developers with an automated tool support that enables formal verification of Event-B models and the refinement process. Despite the fact that Rodin lacks the functionality required to verify the quantitative refinement condition of Definition 6, the reliance on the theory of Markov processes allows the developer to use a large variety of other software tools to facilitate this task. The most well-known examples of such tools are the MATLAB programming environment [58] with Statistics and MDP Toolboxes [59, 24], the Möbius modelling tool [28] and probabilistic model checking platforms [50, 47, 26, 38].

In our work, we have often used the PRISM probabilistic symbolic model checker [50] in conjunction with Rodin to prove the strengthened refinement of Event-B models. Next we consider Rodin and PRISM in more detail.

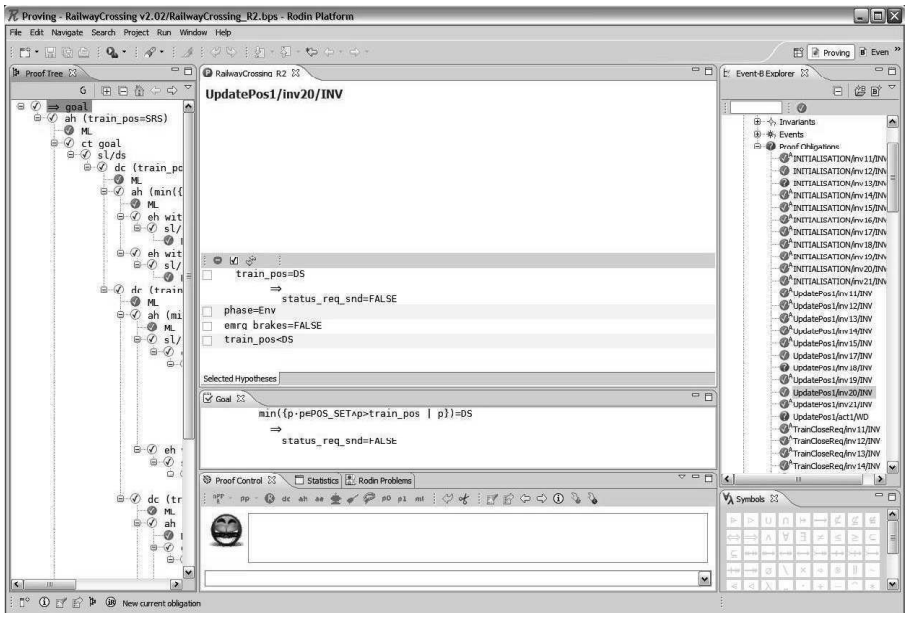


Figure 2.1: The Rodin platform: Proving perspective

Rodin platform In the previous chapter, we have already discussed the Event-B method, its specification language and the main principles of system development in Event-B. Here we provide a short overview of its tool support – the Rodin platform. The Rodin platform is an Eclipse-based integrated development environment for Event-B that provides effective support for modelling and verification by mathematical proof. The platform is open source and is further extendable with plug-ins. More information about available extensions can be found in [33].

The Roding platform has two major purposes. It provides the means for modelling in Event-B, i.e., specifying Event-B constructs – machines and contexts, as well as for proving the consistency, refinement and other formulated properties of models. Well-formedness of Event-B constructs (e.g., type checking, lexical and syntactical analysis) is verified by a *static checker*. The *proof obligation generator* takes the well-formed Event-B constructs as inputs and generates the required proof obligations. Finally, the *proof manager* tries to automatically prove the generated proof obligations as well as maintains existing proofs associated with them. When the proof obligations cannot be discharged automatically, the user can attempt to discharge them interactively using a collection of available proof tactics within the Proving perspective, shown in Figure 2.1.

Typical examples of the Event-B proof obligations are invariant preservation, guard strengthening in refinement (ensuring that when a concrete event is enabled then so is the corresponding abstract one), simulation (en-

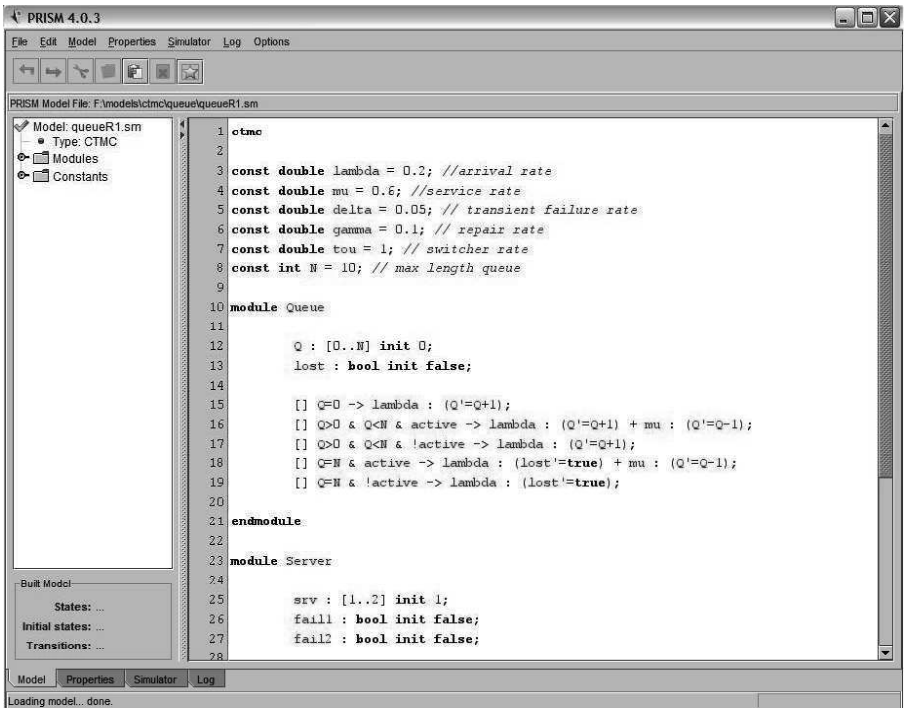


Figure 2.2: The PRISM GUI: model editor

suring that the actions of a concrete event are not contradictory with those of the abstract event), etc. The full list of proof obligations and their formal definitions can be found in [2].

PRISM model checker As we mentioned before, finding analytical solutions while directly relying on the theory of Markov processes is often computationally infeasible. Probabilistic model checking is a fast growing area of research that provides means for verification of Markov models. One of the leading tools in the area is PRISM.

The PRISM model checker is a software tool for formal modelling and verification of systems that exhibit probabilistic behaviour. It provides support for analysis of all three considered in the thesis types of Markov process – DTMC, MDP and CTMC. Moreover, it supports modelling of (priced) probabilistic timed automata and stochastic games (as a generalisation of MDP) [51]. The state-based modelling language of PRISM relies on the *reactive modules* formalism of Alur and Henzinger [6]. A PRISM model consists of a number of *modules* which can interact with each other. The behaviour of each module is described by a set of guarded commands that are quite similar to Event-B events. The latter fact significantly simplifies transformation of Event-B machines to the corresponding PRISM specifica-

tions.

While analysing a PRISM model, one can define a number of temporal logic properties to be evaluated by the tool. The dependability properties that we are interested in verifying are the time-bounded reachability and reward properties. In the property specification language of PRISM, such properties can be formulated using the supported temporal logics – PCTL (Probabilistic Computation Tree Logic) [41] for discrete-time models and CSL (Continuous Stochastic Logic) [9, 17] for continuous-time models. A detailed survey and the specification patterns for probabilistic properties can be found in [36].

PRISM is free and open source tool released under GPL license. It is developed in Java and C++ programming languages [50]. It has been successfully applied in many domains including distributed coordination algorithms, wireless communication protocols, security, dependability and biological models, etc. A screenshot of the GUI version of PRISM is shown in Figure 2.2.

In this chapter, we gave a general overview of the research presented in the thesis. The research results have been published in a number of papers collected in the Part II of the thesis. Next we present a detailed summary of the research results published in each paper along with the statement on the contribution of the thesis author.

Chapter 3

Summary of Publications

The thesis consists of eight publications. A short summary of each publication is presented in this chapter.

Paper I: Integrating Stochastic Reasoning about Critical System Properties into Modelling and Verification in Event-B

In this paper, we propose an approach to integrating stochastic reasoning about reliability and responsiveness of cyclic systems into Event-B modelling. We formally specify the general requirements that should be verified to ensure that the system under construction exhibits the cyclic behaviour. This allows us to strictly control the execution flow of a cyclic system and to use the notion of a system iteration as a discrete unit of time that unambiguously defines a unified time scale for any Event-B machine in the refinement chain. Since we formalise the control flow requirements in a general form that does not depend on a specific Event-B model, they can be easily generated as additional proof obligations (theorems) that must be discharged within the Rodin proving environment. We also propose an extension of the Event-B language with the quantitative probabilistic choice construct and define the semantics for the extended framework as discrete-time Markov processes. We demonstrate how to define reliability and responsiveness as the properties of extended Event-B models and integrate explicit stochastic reasoning about these system properties into the Event-B refinement process. Specifically, we strengthen the notion of Event-B refinement for cyclic systems by additionally requiring that the refined model has to preserve or improve the probabilistic measure of a desired dependability attribute. Finally, we show how to apply the Markov analysis to verification of the strengthened quantitative refinement between Event-B models.

Paper II: Towards Probabilistic Modelling in Event-B

Paper II can be considered as the early version of Paper I. It examines modelling of cyclic systems in Event-B and strengthening the notion of refinement from the reliability point of view. Here, to distinguish between system operational and non-operational states, we introduce a new clause called Operational guards into the Event-B specification language. An operational guard is essentially a shorthand notation implicitly adding the corresponding guard conditions to all the events enabled in the operational states (except the initialisation event). As model invariants, operational guards are inherited in all refined machines of the model. Similarly to Paper I, we separately consider fully probabilistic discrete-time models (modelled by DTMC) and the models that combine both probabilistic and nondeterministic behaviour (modelled by MDP). Unfortunately, the approach proposed for reliability computation of the systems modelled by MDP is not generally correct (the corrected version is presented in Paper I). Nevertheless, the overall approach discussed in this work is credible and have inspired us to write Paper I.

Paper III: Formal Modelling and Verification of Service-Oriented Systems in Probabilistic Event-B

This paper discusses an integration of quantitative assessment of the essential quality of service attributes into the formal modelling process. The paper's technical contribution is two-fold. First, we put forward an approach to creating and verifying a dynamic service architecture in Event-B. Such an Event-B model represents service orchestration explicitly, i.e., it depicts interactions of a service director with the controlled services, the desired sequence of services, simultaneous occurrences of certain services or their mutual exclusion, possible deadlocks conditions as well as the necessary fault tolerance mechanisms. We list the most essential properties of service-oriented systems and formally specify them as the verification conditions for a dynamic service architecture modelled in Event-B. The formalised conditions are to be generated as additional proof obligations by the Rodin platform. Second, we define the quantitative probabilistic choice construct in the continuous-time setting and demonstrate how to augment an Event-B model with stochastic information and transform it into a continuous-time Markov chain. Further, by relying on probabilistic model-checking techniques, we conduct quantitative evaluation of the quality of service attributes of the system under development. We argue that such a verification, i.e. the one that is performed at the architectural level, can be especially useful at the early development stages.

Paper IV: Formal Development and Assessment of a Reconfigurable On-board Satellite System

Ensuring fault tolerance of satellite systems is critical for achieving goals of a space mission. Since the use of redundancy is restricted by the size and the weight of the on-board equipment, the designers need to rely on dynamic system reconfiguration techniques in case of component failures. Paper IV discusses a formal approach to development and assessment of fault tolerant dynamically reconfigurable systems. Specifically, we define the guidelines for step-wise development of such systems in Event-B. Furthermore, we demonstrate how to formally assess a reconfiguration strategy and evaluate whether the chosen fault tolerance mechanism fulfils the reliability and performance objectives. The proposed approach is illustrated by a case study – development and assessment of the reconfigurable satellite Data Processing Unit (DPU). The adopted approach is similar to the one proposed in Paper III. Namely, we enrich the Event-B models of DPU with explicit probabilistic information about reliability of its components and the duration of time required to execute the tasks assigned to these components. Furthermore, using probabilistic model checking, we compare the reliability and performance measures of a system by employing two different fault tolerance mechanisms: the one realising a standard redundancy scheme and the other one that is based on dynamic reconfiguration. The method proposed in the paper not only guarantees correct design of complex fault tolerance mechanisms but also facilitates finding suitable trade-offs between system reliability and performance.

Paper V: Quantitative Verification of System Safety in Event-B

In this paper, we present a method for integrating quantitative safety assessment into the formal system development in Event-B. The main merit of the method is combining logical (qualitative) reasoning about correctness of the system behaviour with probabilistic (quantitative) analysis of its safety. Essentially, the proposed approach sets the guidelines for safety-explicit development in Event-B. Specifically, we structure the functioning of a system according to a number of execution stages that are considered separately in the refinement process. For each such a stage, we describe the expected (safe) system behaviour as a number of (standard) safety invariants. At the final refinement step, we rely on these safety invariants to specify the correct behaviour of the overall system in a hazardous situation. As a result, the refinement process facilitates not only correctness-preserving model transformations but also establishes a logical link between the system safety conditions at different levels of abstraction as well as leads to deriving a logical representation of hazardous conditions. An explicit modelling

of probabilities of component failures, based on the extension of Event-B discussed in Paper I, has allowed us to calculate the likelihood of hazard occurrence. The proposed method is illustrated by a case study – an automatic railway crossing system.

Paper VI: Augmenting Formal Development of Control Systems with Quantitative Reliability Assessment

This work discusses integration of reliability analysis into the formal development of control systems in Event-B. In the paper, in order to enable stochastic reasoning about Event-B models, we exploit the fact that the semantics of an Event-B model is essentially a trace semantics. We propose to augment every event trace of an Event-B machine with probabilistic information, i.e., transform it into the corresponding probabilistic trace. This is achieved by adding the probabilistic weight to every event in a trace. Such an augmentation allows us to represent the behaviour of an Event-B model as a set of observable probabilistic traces, which, in turn, explicitly defines a DTMC (in this paper, we consider only fully-probabilistic systems). Furthermore, we define the quantitative Event-B refinement (from the reliability perspective) as the probabilistic trace refinement. The use of Event-B combined with probabilistic model checking for modelling and verification is validated by a case study – a heater controller.

Paper VII: From Formal Specification in Event-B to Probabilistic Reliability Assessment

Paper VII further exploits the integration of two frameworks: refinement in Event-B and probabilistic model checking. It is a practical work that demonstrates benefits of this integration by modelling and verification of the standard fault tolerance mechanisms used in reliability engineering. Specifically, we show that introducing any of such fault tolerance schemes as Triple Modular Redundancy (TMR), Hot/Cold Standby Spare and the TMR arrangement with a spare is a valid refinement step in Event-B. To evaluate which of them is more optimal from the reliability point of view, we demonstrate how to transform the Event-B specifications into DTMC. We model the mentioned fault tolerance strategies as their PRISM DTMC counterparts and perform the required evaluation using the PRISM model checker.

Paper VIII: Quantitative Reasoning about Dependability in Event-B: Probabilistic Model Checking Approach

Paper VIII is a continuation of Paper VII. In this paper, we provide the general modelling guidelines for the Event-B to PRISM model transformation. Specifically, we discuss the semantic correspondences between Event-B

events, containing different types of actions, and the respective combinations of the guarded commands and modules of the PRISM specification language. In this paper, we rely on the notion of the operational guard (see Paper II) to distinguish between the operational and non-operational system states. The operational guard of an Event-B model is also used to formulate the corresponding reliability properties in PRISM. Finally, we illustrate the use of the Event-B in conjunction with the PRISM model checker by an example – a simple monitoring system.

Author's contribution: The main author of all the included papers. Responsible for the Event-B development of most case studies described in the papers, except for the models of DPU presented in Paper IV (the models have been developed by M.Sc. Inna Pereverzeva). Responsible for the probabilistic modelling and analysis of all the conducted case studies. Has made a major contribution to investigating the theoretical results proposed in the papers, yet under outstanding technical and theoretical supervision of Docent Elena Troubitsyna and Docent Linas Laibinis.

Chapter 4

Conclusions

4.1 Related Work

Formal modelling and quantitative verification of dependable systems is an area of active research. It is rather unfeasible to describe all the work conducted in the field. Therefore, we are going to overview only the topics that are specifically relevant to the approaches proposed in the thesis. In particular, we consider the approaches to modelling reachability in Event-B, formalisation of the system execution flow in Event-B, and a probabilistic extension of the refinement calculus. Moreover, we summarise a number of methods employing the application of probabilistic model checking to dependability assessment. Finally, we briefly describe the industrial use of the formal techniques combining mechanised theorem proving with the quantitative reasoning about system dependability.

Reachability in Event-B Traditionally, in Event-B, all essential properties that must be preserved by the system under development are specified as model invariants. However, this allows the developer to reason only about the properties that hold with probability one. Verification of reachability properties has a similar limitation – Event-B allows the developer to prove only the total correctness property, i.e., the property that a model terminates and delivers the expected result.

There is a solid body of research aimed at extending the set of properties that can be modelled and verified in Event-B. For instance, in [4] Abrial et al. proposed an approach to defining and proving reachability properties in Event-B that is similar to ours. They lift the notion of total correctness of an individual program task to total correctness of the whole program in a sense that none of the tasks of a running program may prevent termination of any other task. In other words, they define reachability as follows: "the termination of each task must be reachable". To prove it, in the initial model

the desired task is modelled as reachable in "one shot". The reliance on convergence of the new events introduced by refinement allows the developer to guarantee that the task is eventually reachable in the refined models. The later work by Hoang and Abrial [42] discusses how to reason about such liveness properties as existence, progress and persistence in Event-B. The persistence property is the same as our (ideal) goal reachability property considered in Paper IV. The authors propose proof rules for verification of the defined properties and, as a result, significantly increase the number of system properties that can be verified in Event-B. However, the papers discussed focus on reasoning about reachability and liveness properties that are preserved by the system with probability one.

Hallerstede and Hoang [39] have proposed an extension of the Event-B framework to model the probabilistic system behaviour. Specifically, they introduce the qualitative probabilistic choice operator to reason about almost certain termination. This operator is used to bound demonic nondeterminism, and to facilitate proving convergence of the new events in Event-B models. In particular, they apply this technique to resolve the contention problem in Fireware protocol. In [82], Yilmaz and Hoang also successfully apply the qualitative probabilistic reasoning in Event-B to formalise the Rabin's choice coordination algorithm. The use of the qualitative probabilistic choice is currently supported by the Rodin tool [32]. However, the presented approach is not suitable for quantitative evaluation of system properties, since the introduced operator does not contain explicit probabilistic information.

Formalising system execution in Event-B Several approaches have been recently proposed to enable explicit reasoning about the dynamic system behaviour in Event-B. Iliasov [43] has proposed a method for expressing use case scenarios as formal verification conditions. These conditions appear in Event-B models as additional proof obligations. Moreover, Iliasov presents a formal semantics of use cases as control flows. The developed extension of the Rodin platform facilitates automatic translation of use cases (given in a diagrammatic way) into the proof obligations of a model.

An integration of CSP and Event-B to facilitate reasoning about the dynamic system behaviour has been proposed by Schneider et al. [74]. In the latter work, CSP is used to provide an explicit control flow for an Event-B model as well as to separate the requirements dependent on the control flow information.

The approach we have taken in the thesis is inspired by these works. We, however, rely solely on Event-B to built a (dynamic) system architecture. Specifically, in Paper III, we have extended the work [53, 52] of Laibinis et al. on a formalisation of Lyra – an UML-based approach for development of service-oriented systems – by defining in Event-B a number of the formal

verification requirements for service orchestration. Moreover, in Paper I, we have used a similar approach to formally define the desired execution flow of a cyclic system in Event-B.

In [5], Ait-Sadoune and Ait-Ameur have addressed the problem of formal validation and verification of services composition. To tackle the problem, the authors propose to extract Event-B specifications from the BPEL (Business Process Execution Language) models expressing services composition and description, and augment the extracted models with relevant invariants and theorems. Such an approach undoubtedly benefits from combining Event-B with BPEL, which is the standard language for defining Web services composition. However, it is not clear how to guarantee correctness of the proposed model transformation, in part because of the fact that the approach allows the developer to introduce new event guards into the extracted models. Moreover, the approach that we present in Paper III combines both proof-based and probabilistic model checking techniques to verification of services composition that allows us to deal with verification of a significantly wider class of system requirements (properties).

Probability and refinement The topic of probabilistic formal modelling has been extensively explored by Morgan et al. in the context of probabilistic refinement calculus [60] – an extension of the standard refinement calculus. The introduced notion of probabilistic data refinement has been used, among other things, for assessment of system dependability (see [61, 60], for instance). Here, probabilistic programs are modelled using expectation transformers and probabilistic data refinement is verified via simulation between datatypes. In [77], a similar approach is taken to enable reasoning about reliability in probabilistic action systems [76] – the extension of the action systems that combine both probabilistic and nondeterministic behaviour. However, proving simulation that implies data refinement between datatypes is an extremely difficult problem, which immediately raises the scalability issue. Moreover, the majority of non-functional system attributes, including those of dependability, explicitly depend on time. However, to the best of our knowledge, the notion of time is not defined in the probabilistic refinement calculus.

In [70], Rao has proposed a generalisation of the UNITY formalism [25] that enables reasoning about probability and parallelism. Specifically, he generalise the weakest precondition semantics of UNITY to define a new predicate transformer – weakest probabilistic precondition. Relying on this extension, he also generalise certain relations of the UNITY to make them amenable for reasoning about probabilistic (parallel) programs. In particular, new probabilistically leads-to relation allows for defining probabilistic progress properties. Similarly to the approach taken in [39], Rao does not aim at computing any kind of probabilistic measures but to reason about the

progress properties that are attained with probability one. The proposed methodology has proved its worth in constructing and proving probabilistic algorithms [70].

A connection between probabilistic reasoning and program refinement has been investigated by Meinicke and Solin [62]. The authors introduce a refinement algebra for reasoning about probabilistic program transformations. In particular, they investigate the data and atomicity refinement rules for probabilistic programs and explore the difference between probabilistic and non-probabilistic programs. They reason about the probabilistic program transformations without introducing a probabilistic choice operator or other explicit probabilistic attributes. Our approach is rather different from the one by Meinicke and Solin. We introduce the quantitative probabilistic choice operator, which explicitly defines concrete probabilistic values for different choices. The introduced probabilistic information is used to verify quantitative non-functional properties of the system and their preservation by refinement. Otherwise, we rely on the existing Event-B refinement framework to guarantee correctness of model transformations.

COMPASS project One of the most intensive and ambitious works on modelling and verification of safety-critical systems is being conducted within the COMPASS project [27]. The goal of this project is to develop a coherent co-engineering approach for system specification and evaluation of system-level correctness, dependability and performability. The main formalism adopted in the approach is Architecture Analysis and Design Language (AADL). Within the project, a formal extended semantics for AADL has been developed. The extended semantics incorporates functional, probabilistic and hybrid aspects of safety-critical systems [22, 21]. In the proposed approach, a system specification consists of two models – the nominal model that describes the system under normal operation and the error model that specifies how the system can fail. These two models are linked through fault injection. The method benefits from a powerful tools support – the COMPASS platform, which is based on various model-checking techniques and provide the means for verification and validation of AADL models. The COMPASS methods and the toolset have been validated by a set of industrial-size case-studies, see [31] for instance. Though initially the framework has been proposed only for the space domain, it is fairly generic and comprehensive per se. In comparison, in this thesis, rather than invent a completely new approach, we extend the Event-B framework and rely on its powerful correct-by-construction development technique. Building a tool support for probabilistic Event-B, e.g., bridging Rodin with existing probabilistic model checking tools, is one of the main directions of our future work.

Probabilistic model checking Probabilistic model checking is widely used for assessment of non-functional system requirements. There are a number of works, for instance, see [49, 15, 57, 23], successfully applying the quantitative model checking techniques to evaluate system dependability and quality of service. These approaches benefit from the existing good tool support for formal modelling and verification of discrete- and continuous-time Markov processes [50, 47]. The principal difference between model checking and our approach stems from the fact that the model checking generally aims at assessing non-functional system attributes of already developed systems. However, postponing the dependability and quality of service evaluation to the later development stages can lead to major system redevelopment, if the non-functional requirement are not met. In our approach, the assessment of non-functional requirements proceeds hand-in-hand with the system development by refinement, which allows us to analyse the behaviour of a designed system at the early stages of development. Despite the discussed differences, we believe that the probabilistic model checking techniques can complement our approach. More specifically, quantitative model checkers can be used in conjunction with Rodin to prove the strengthened (quantitative) refinement of Event-B models.

Quantitative analysis of safety also often relies on probabilistic model checking. For instance, the work reported in [34] presents model-based probabilistic safety assessment based on generating PRISM specifications from Simulink diagrams annotated with the failure logic. The method pFMEA (probabilistic Failure Modes and Effect Analysis) also relies on the PRISM model checker to conduct quantitative analysis of safety [37]. The approach integrates the failure behaviour into a system model, represented by a continuous-time Markov chain, via failure injection. In [65], the authors proposed a method for probabilistic model-based safety analysis for synchronous parallel systems. It has been shown that different types of failures, in particular per-time and per-demand, can be modelled and analysed using probabilistic model checking. In general, the methods based on model checking aim at safety evaluation of already developed systems. They extract a model eligible for probabilistic analysis and evaluate impact of various system parameters on its safety. In Paper V, we have aimed at providing the designers with a safety-explicit development method. As a result, the safety analysis is essentially integrated into the system development by refinement. It allows us to perform the quantitative assessment of safety within proof-based verification of the system behaviour.

Formal methods in the railway and aerospace domains Formal methods are gaining more popularity in the industrial development and verification of safety-critical systems. Specifically, the railway domain is often considered as one of the most productive application areas of formal meth-

ods. In particular, the B Method and Event-B are successfully applied to formal development of railway systems [29, 18, 56, 55]. In [67, 66], safety analysis of a formal model of a radio-based railway crossing controller (that we discuss in Paper V) has been performed using the KIV theorem prover. The same case study has been explored in [30] using statecharts with the goal of proving the required safety properties by model checking. To achieve this goal, the Statemate Verification Environment (STVE) for STATEMATE tool has been developed. However, both these methods concern with only logical (qualitative) reasoning about railway crossing safety and do not include the quantitative safety analysis.

One of the success stories of the DEPLOY project [46] is the use of the Event-B method in the aerospace domain (see, for instance, [45, 44]). In particular, formalisation and verification of interplay between satellite operational mode and the fault-tolerance mechanisms, as well as system dynamic reconfiguration in the presence of component failures have been investigated. The modelling approach and case-study that we discuss in Paper IV are aimed at enhancing this research work with the probabilistic reasoning about dependability of satellite systems. While discussing the application of formal verification techniques in the aerospace domain, it is worth to mention again the COMPASS project and, in particular, work [31]. In this paper, the authors have performed a thorough analysis of a satellite platform covering discrete, real-time, hybrid and probabilistic system aspects. However, despite the notable progress in the area of the industrial application of formal methods, the approaches for integrating theorem proving and quantitative system assessment are still scarce.

4.2 Research Conclusions

Recent advances in formal modelling and verification have demonstrated that application of formal engineering methods becomes essential for ensuring both functional correctness and dependability of complex computer systems. The refinement-based modelling frameworks, such as classical B and Event-B methods, formalise model-driven development process and enable development of systems correct-by-construction. However, these formal approaches suffer from the lack of support for stochastic reasoning about system dependability. The main goal of the research reported in this thesis is to overcome this limitation.

To achieve our research goal, we have proposed a formal approach to development and quantitative verification of dependable systems within the Event-B modelling framework. To enable the probabilistic reasoning about Event-B models, we have extended the formal semantics of the Event-B specification language and demonstrated how the behaviour of a probabilistically-

enriched Event-B model can be represented by a particular kind of Markov process. We have defined the strengthened (quantitative) version of Event-B refinement that takes into account a particular dependability attribute, yet does not contradict to the standard Event-B refinement process. Finally, to enable verification of the quantitative refinement, we have build a connection between the Event-B formalism and the PRISM framework – a state-of-the-art probabilistic model checking technique. We believe that our work establishes sound mathematical foundations for integrating the logical reasoning about functional correctness and the probabilistic reasoning about dependability of a system.

The work presented in the thesis has certain limitations (discussed in more detail in the next section) that will be addressed in the future to build a holistic approach to development of complex dependable systems. Nevertheless, the thesis addresses the following essential aspects of the dependability-explicit system development: refinement-based development of reliable and responsive cyclic (e.g., control and monitoring) systems, modelling and verification of service-oriented systems, achieving fault-tolerance of satellite systems via dynamic reconfiguration, and invariant-based safety analysis of highly-intensive systems.

4.3 Future Work

In our work, we have established the initial foundation for integrating quantitative reasoning about dependability into the formal system development by refinement. However, further advances in both theory and automated tool support are required to create a versatile technique for development of industrial-size dependable systems. The foreseen future directions for our research are outlined below.

At the moment, the probabilistic choice is defined as an operator that can be used only in the body of an event. It would be advantageous to introduce such a choice into the declaration of the event local parameters, to allow them to be initialised according to some probability distribution. Another possible extension would be an introduction of a new **Distributions** clause into a machine, where the designer can define some constant (parametrised) distributions than can be used in more than one event of a model. Thus, a discrete uniform distribution can be defined for any finite set of elements. At the moment, the use of probabilistic choice is somewhat cumbersome and these two features can definitely make it more handy. Moreover, to handle the complexity posed by the size of state space of large-scale systems, we can employ such techniques as lumping and probabilistic bisimulation (see, e.g., [48, 54] for fully probabilistic systems and [40, 75] for the systems that contain both nondeterministic and probabilistic behaviour).

To make an Event-B model amenable to probabilistic assessment, we need to extend the Rodin platform with a dedicated plug-in that would enable the use of the quantitative probabilistic choice operator in Event-B machines. Such a plug-in would help the developer to derive an underlying Markov model from an Event-B specification and to verify the strengthened quantitative refinement. Moreover, it would also bridge the Rodin platform with the existing state-of-the-art techniques for formal verification of Markov models (e.g., probabilistic model checking tools). To derive a Markov model, we need to omit some excessive elements of the Event-B modelling language and also ensure the consistency between the derived model and the initial specification.

Another direction for future research is to use the experience of the automatic theorem generation of [43] to obtain additional proof obligations that would constrain the structure of an Event-B model and define its execution flow. These proof obligations can be generated as theorems in Rodin. Similarly to the approach presented in [43], in Papers I and III we have proposed to formalise the requirements that define the system execution flow in a general form. Providing the automatic tool support for generation of the required proof obligations would significantly facilitate the quantitative system assessment in Event-B.

Bibliography

[1] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 2005.

[2] J.-R. Abrial. *Modeling in Event-B*. Cambridge University Press, 2010.

[3] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: an Open Toolset for Modelling and Reasoning in Event-B. *International Journal on Software Tools for Technology Transfer*, 12(6):447–466, 2010.

[4] J.-R. Abrial, D. Cansell, and D. Méry. Refinement and Reachability in Event-B. In *ZB 2005, Formal Specification and Development in Z and B*, pages 222–241. Springer, 2005.

[5] I. Ait-Sadoune and Y. Ait-Ameur. A Proof Based Approach for Modelling and Verifying Web Services Composition. In *ICECCS 2009, International Conference on Engineering of Complex Computer Systems*, pages 1–10. IEEE, 2009.

[6] R. Alur and T. Henzinger. Reactive Modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[7] A. Avizienis, J.-C. Laprie, and B. Randell. Dependability and its Threats - A taxonomy. In *IFIP Congress Topical Sessions*, pages 91–120, 2004.

[8] A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. Dependable Sec. Comput.*, 1(1):11–33, 2004.

[9] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying Continuous Time Markov Chains. In *CAV'96, International Conference on Computer Aided Verification*, pages 269–276. Springer, 1996.

[10] R. J. R. Back. *On the Correctness of Refinement Steps in Program Development*. PhD thesis, University of Helsinki, Helsinki, 1978.

[11] R. J. R. Back and R. Kurki-Suonio. Decentralization of Process Nets with Centralized Control. In *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 131–142. ACM, 1983.

[12] R. J. R. Back and K. Sere. Stepwise Refinement of Action Systems. *Structured Programming*, 12(1):17–30, 1991.

[13] R. J. R. Back and K. Sere. From Action Systems to Modular Systems. *Software – Concepts and Tools*, 17(1):26–39, 1996.

[14] R. J. R. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Springer-Verlag, 1998.

[15] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Automated Performance and Dependability Evaluation Using Model Checking. In *Performance Evaluation of Complex Systems: Techniques and Tools*, pages 261–289. Springer-Verlag, 2002.

[16] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT press, 2008.

[17] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate Symbolic Model Checking of Continuous-Time Markov Chains. In *CONCUR'99, International Conference on Concurrency Theory*, pages 146–161. Springer, 1999.

[18] P. Behm, P. Benoit, A. Faivre, and J.-M. Meynadier. Météor: A Successful Application of B in a Large Project. In *FM'99, World Congress on Formal Methods in the Development of Computing Systems*, pages 369–387. Springer, 1999.

[19] J. Bicarregui, J. S. Fitzgerald, P. G. Larsen, and J. C. Woodcock. Industrial Practice in Formal Methods: A Review. In *FM'09, World Congress on Formal Methods*, pages 810–813. Springer, 2009.

[20] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains. Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons, 2006.

[21] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri. Safety, Dependability and Performance Analysis of Extended AADL Models. *Comput. J.*, 54(5):754–775, 2011.

[22] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, M. Roveri, and R. Wimmer. A Model Checker for AADL. In T. Touili, B. Cook, and P. Jackson, editors, *CAV'10, Computer Aided Verification*, pages 562–565. Springer, 2010.

[23] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Trans. Softw. Eng.*, 37:387–409, 2011.

[24] I. Chadés, M.-J. Cros, F. Garcia, and R. Sabbadin. Markov Decision Processes (MDP) Toolbox. online at http://www.inra.fr/mia/T/MDPtoolbox/.

[25] K. Mani Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

[26] F. Ciesinski and C. Baier. LiQuor: A tool for Qualitative and Quantitative Linear Time analysis of Reactive Systems. In *QEST'06, International Conference on Quantitative Evaluation of Systems*, pages 131–132. IEEE CS Press, 2006.

[27] Correctness, Modelling and Performance of Aerospace Systems (COMPASS). European Space Agency project. online at http://compass.informatik.rwth-aachen.de/.

[28] T. Courtney, S. Gaonkar, K. Keefe, E. Rozier, and W. H. Sanders. Möbius 2.3: An Extensible Tool for Dependability, Security, and Performance Evaluation of Large and Complex System Models. In *DSN 2009, Dependable Systems and Networks*, pages 353–358. IEEE Computer Society, 2009.

[29] D. Craigen, S. Gerhart, and T. Ralson. Case Study: Paris Metro Signaling System. In *IEEE Software*, pages 32–35, 1994.

[30] W. Damm and J. Klose. Verification of a Radio-Based Signaling System Using the Statemate Verification Environment. *Formal Methods in System Design*, 19(2):121–141, 2001.

[31] M.-A. Esteve, J.-P. Katoen, V. Y. Nguyen, B. Postma, and Y. Yushtein. Formal Correctness, Safety, Dependability and Performance Analysis of a Satellite. In *ICSE'2012, International Conference on Software Engineering*. ACM and IEEE CS Press, 2012.

[32] Event-B and Rodin Documentation Wiki. Qualitative Probability Plug-in. online at http://wiki.event-b.org/index.php/Event-B_Qualitative_Probability_User_Guide.

[33] Event-B and Rodin Documentation Wiki. Rodin Plug-ins. online at http://wiki.event-b.org/index.php/Rodin_Plug-ins.

[34] A. Gomes, A. Mota, A. Sampaio, F. Ferri, and J. Buzzi. Systematic Model-based Safety Assessment via Probabilistic Model Checking.

In *ISoLA'10, International Conference on Leveraging Applications of Formal Methods, Verification, and Validation*, pages 625–639. Springer-Verlag, 2010.

[35] C. M. Grinstead and L. J. Snell. *Introduction to Probability*. American Mathematical Society, 2006.

[36] L. Grunske. Specification patterns for probabilistic quality properties. In *ICSE 2008, International Conference on Software Engineering*, pages 31–40. ACM, 2008.

[37] L. Grunske, R. Colvin, and K. Winter. Probabilistic Model-Checking Support for FMEA. In *QEST'07, International Conference on Quantitative Evaluation of Systems*, 2007.

[38] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang. PARAM: A Model Checker for Parametric Markov Models. In *CAV'10, International Conference on Computer Aided Verification*, pages 660–664. Springer, 2010.

[39] S. Hallerstede and T. S. Hoang. Qualitative probabilistic modelling in Event-B. In J. Davies and J. Gibbons, editors, *IFM 2007, Integrated Formal Methods*, pages 293–312, 2007.

[40] H. Hansson. *Time and Probability in Formal Design of Distributed Systems*. Elsevier, 1995.

[41] H. Hansson and B. Jonsson. A Logic for Reasoning about Time and Reliability. In *Formal Aspects of Computing*, pages 512–535, 1994.

[42] T. S. Hoang and J.-R. Abrial. Reasoning about Liveness Properties in Event-B. In *ICFEM'11, International Conference on Formal Methods and Software Engineering*, pages 456–471. Springer, 2011.

[43] A. Iliasov. Use Case Scenarios as Verification Conditions: Event-B/Flow Approach. In *SERENE 2011, Software Engineering for Resilient Systems*, pages 9–23. Springer-Verlag, 2011.

[44] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala. Developing Mode-Rich Satellite Software by Refinement in Event-B. In *FMICS 2010, Formal Methods for Industrial Critical Systems*, pages 50–66. Springer, 2010.

[45] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, P. Väisänen, D. Ilic, and T. Latvala. Verifying Mode Consistency for On-Board Satellite Software. In *SAFECOMP 2010, International Conference on Computer Safety, Reliability and Security*, pages 126–141. Springer, 2010.

[46] Industrial Deployment of System Engineering Methods Providing High Dependability and Productivity (DEPLOY). IST FP7 IP Project. online at http://www.deploy-project.eu/.

[47] J.-P. Katoen, I. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The Ins and Outs of The Probabilistic Model Checker MRMC. In *QEST 2009, Quantitative Evaluation of Systems*, pages 167–176. IEEE Computer Society, 2009.

[48] J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. D. Van Nostrand Company, 1960.

[49] M. Kwiatkowska, G. Norman, and D. Parker. Controller Dependability Analysis by Probabilistic Model Checking. In *Control Engineering Practice*, pages 1427–1434, 2007.

[50] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *CAV'11, International Conference on Computer Aided Verification*, pages 585–591. Springer, 2011.

[51] M. Kwiatkowska and D. Parker. Advances in Probabilistic Model Checking. In T. Nipkow, O. Grumberg, and B. Hauptmann, editors, *Software Safety and Security - Tools for Analysis and Verification*, volume 33 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 126–151. IOS Press, 2012.

[52] L. Laibinis, E. Troubitsyna, and S. Leppänen. Formal Reasoning about Fault Tolerance and Parallelism in Communicating Systems. In *Methods, Models and Tools for Fault Tolerance*, pages 130–151. Springer-Verlag, 2009.

[53] L. Laibinis, E. Troubitsyna, S. Leppänen, J. Lilius, and Q. Malik. Formal Model-Driven Development fn Communicating Systems. In *ICFEM 2005, International Conference on Formal Engineering Methods*, pages 188–203. Springer-Verlag, 2005.

[54] K. G. Larsen and A. Skou. Bisimulation through Probabilistic Testing. In *Information and Computation 94*, pages 1–28, 1991.

[55] T. Lecomte. Safe and Reliable Metro Platform Screen Doors Control/Command Systems. In *FM'08, International Symposium on Formal Methods*, pages 430–434. Springer, 2008.

[56] T. Lecomte, T. Servat, and G. Pouzancre. Formal Methods in Safety-Critical Railway Systems. In *Brasilian Symposium on Formal Methods*, 2007.

[57] M. Massink, J.-P. Katoen, and D. Latella. Model Checking Dependability Attributes of Wireless Group Communication. In *DSN'04, International Conference on Dependable Systems and Networks*, pages 711–720, 2004.

[58] MathWorks. MATLAB: The Language of Technical Computing. online at http://www.mathworks.se/products/matlab/.

[59] MathWorks. Statistics Toolbox. online at http://www.mathworks.com/products/statistics/.

[60] A. K. McIver and C. C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2005.

[61] A. K. McIver, C. C. Morgan, and E. Troubitsyna. The Probabilistic Steam Boiler: a Case Study in Probabilistic Data Refinement. In *International Refinement Workshop, ANU, Canberra*. Springer, 1998.

[62] L. Meinicke and K. Solin. Refinement algebra for probabilistic programs. In *Formal Aspects of Computing*, volume 22, pages 3–31, 2010.

[63] C. C. Morgan. *Programming from Specification, 2nd ed.* Prentice Hall, 1994.

[64] P. D. T. O'Connor. *Practical Reliability Engineering, 3rd ed.* John Wiley & Sons, 1995.

[65] F. Ortmeier and M. Güdemann. Probabilistic Model-Based Safety Analysis. In *QAPL 2010, Workshop on Quantitative Aspects of Programming Languages*, EPTCS, pages 114–128, 2010.

[66] F. Ortmeier, W. Reif, and G. Schellhorn. Formal Safety Analysis of a Radio-Based Railroad Crossing Using Deductive Cause-Consequence Analysis (DCCA). In *EDCC 2005, European Dependable Computing Conference*, pages 139–151. Springer, 2007.

[67] F. Ortmeier and G. Schellhorn. Formal Fault Tree Analysis: Practical Experiences. In *AVoCS 2006, International Workshop on Automated Verification of Critical Systems*, volume 185 of *ENTCS*, pages 139–151. Elsevier, 2007.

[68] E. Parzen. *Stochastic Processes*. Holden-Day, 1964.

[69] M. Putterman. *Markov Decision Processes. Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2005.

[70] J. R. Rao. *Extension of the UNITY Methodology: Compositionality, Fairness and Probability in Parallelism*. Springer-Verlag, 1995.

[71] Rigorous Open Development Environment for Complex Systems (RODIN). IST FP6 STREP project. online at http://rodin.cs.ncl.ac.uk/.

[72] Rodin. Event-B Platform. online at http://www.event-b.org/.

[73] S. Schneider. *The B-Method: An Introduction*. Palgrave, 2001.

[74] S. Schneider, H. Treharne, and H. Wehrheim. A CSP Approach to Control in Event-B. In *IFM 2010, Integrated Formal Methods*, pages 260–274. Springer, 2010.

[75] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In *Nordic Journal of Computing, 2(2)*, pages 250–273, 1995.

[76] K. Sere and E. Troubitsyna. Probabilities in Action Systems. In *Nordic Workshop on Programming Theory*, 1996.

[77] E. Troubitsyna. Reliability Assessment through Probabilistic Refinement. *Nord. J. Comput.*, 6(3):320–342, 1999.

[78] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *Fault Tree Handbook*. Nuclear Regulatory Commission. NUREG–0492, 1981.

[79] A. Villemeur. *Reliability, Availability, Maintainability and Safety Assessment*. John Wiley & Sons, 1992.

[80] D. J. White. *Markov Decision Processes*. John Wiley & Sons, 1993.

[81] J. C. Woodcock, P. G. Larsen, J. Bicarregui, and J. S. Fitzgerald. Formal methods: Practice and Experience. *ACM Comput. Surv.*, 41(4), 2009.

[82] E. Yilmaz and T. S. Hoang. Development of Rabin's Choice Coordination Algorithm in Event-B. *ECEASST*, 35, 2010.

Part II

Original Publications

Paper I

Integrating Stochastic Reasoning about Critical System Properties into Modelling and Verification in Event-B

Anton Tarasyuk, Elena Troubitsyna and Linas Laibinis

Submitted to *Science of Computer Programming*

Integrating Stochastic Reasoning about Critical System Properties into Modelling and Verification in Event-B

A. Tarasyuk[a,b], E. Troubitsyna[a], L. Laibinis[a]

[a] *Åbo Akademi University, Joukahaisenkatu 3-5 A, 20520 Turku, Finland*
[b] *Turku Centre for Computer Science, Joukahaisenkatu 3-5 B, 20520 Turku, Finland*

Abstract

Formal modelling and verification techniques are widely used for development of critical computer-based systems. Such techniques include Event-B – a state-based formalism that enables development of systems correct-by-construction. While Event-B offers a scalable approach to ensuring functional correctness of a system, it leaves aside modelling of non-functional critical properties, e.g., reliability and responsiveness, that are essential for ensuring dependability of critical systems. Both reliability, i.e., the probability of the system to function correctly over a given period of time, and responsiveness, i.e., the probability of the system to complete execution of a requested service within a given time bound, are defined as quantitative stochastic measures. In this paper we propose an extension of the Event-B semantics to enable stochastic reasoning about non-functional properties of cyclic systems. We define the requirements that a cyclic system should satisfy and introduce the notions of reliability and responsiveness refinement. Such an extension integrates reasoning about functional correctness and stochastic modelling of non-functional characteristics into the formal system development. It allows the designer to ensure that a developed system does not only correctly implement its functional requirements but also satisfies given non-functional quantitative constraints.

Keywords: Event-B, refinement, stochastic reasoning, reliability, responsiveness, cyclic systems, Markov processes

Email addresses: anton.tarasyuk@abo.fi (A. Tarasyuk), elena.troubitsyna@abo.fi (E. Troubitsyna), linas.laibinis@abo.fi (L. Laibinis)

1. Introduction

Formal methods – the mathematically-based approaches that provide the developers with rigorous ways to design and analyse systems – are extensively used in the design of critical computer-based systems. Such methods include Event-B [1, 2] – a formalism derived from the B Method [3] to facilitate development of reactive and distributed systems. Event-B is a rigorous, state-based framework supporting the correct-by-construction system development. While developing a computer-based system in Event-B, we start from an abstract specification that defines the essential behaviour and properties of the system under construction. Via a number of correctness preserving model transformations – refinement steps, the abstract specification is transformed into a specification that is close to the desired implementation. In the development process, correctness of each refinement step is verified by proofs.

Formal development in Event-B allows us to ensure that a resulting detailed specification adheres to its abstract counterpart, i.e., it guarantees that the services provided by the system are functionally correct. However, in the current process of system refinement, the non-functional requirements are abstracted away. This deprives the designers of a common semantic model that would allow them to evaluate the impact of the chosen design decisions on the non-functional requirements. Hence there is a clear need for integration between modelling of functional and non-functional system requirements.

In this paper we extend the Event-B framework to enable stochastic modelling of reliability and responsiveness of cyclic systems. *Reliability* is the probability of the system functioning correctly over a given period of time under a given set of operating conditions [4, 5, 6], while *responsiveness* is the likelihood that the system successfully completes service delivery within a certain time bound [7, 8]. These properties are dual in the sense that reliability defines a probabilistic measure of the system staying operational during a certain time period, while responsiveness gives a probabilistic measure of the system termination within a certain period of time. We rely on the notion of *iteration* as a discrete unit of time defining a unified time scale for cyclic systems, i.e., the systems that iteratively execute a predefined sequence of computational steps. We formally define the conditions that should be verified to ensure that the system under construction is indeed cyclic.

To enable explicit probabilistic reasoning about reliability and responsiveness, we introduce a new language construct – the quantitative proba-

bilistic choice – and define the semantics of extended Event-B models. We show that, in the case of fully probabilistic systems, the underlying model of a probabilistically-enriched Event-B specification is a discrete-time Markov chain [9]. Moreover, in the case of the systems that contain both probabilistic and demonic behaviour, this model becomes a Markov decision process [10, 11].

To enable reliability- and responsiveness-explicit development in probabilistically-augmented Event-B, we strengthen the notion of refinement by requiring that a refined model, besides being a proper functional refinement of its more abstract counterpart, also satisfies a number of quantitative constraints. These constraints ensure that the refined model improves (or at least preserves) the current system reliability or responsiveness. These additional constraints are derived from the fundamental properties of discrete Markov chains and Markov decision processes. We believe that our work establishes sound mathematical foundations for integrating logical reasoning about functional correctness and probabilistic reasoning about critical system properties.

The paper is structured as follows. In Section 2 we overview our formal framework – Event-B. In Section 3 we introduce the notion of cyclic systems, formally define the conditions required to verify their cyclic nature and rigorously define the notion of a system iteration and its properties. In Section 4 we introduce the probabilistic choice operator and give an example of a probabilistic Event-B model. In Sections 5 and 6 we present the strengthened notion of Event-B refinement for both fully probabilistic systems and the systems with nondeterminism. In Section 7 we summarise the presented approach to stochastic reasoning in Event-B. Finally, in Sections 8 and 9 we overview the related work in the field and give some concluding remarks.

2. Introduction to Event-B

Event-B [1] is a formal framework derived from the (classical) B method [3] to model parallel, distributed and reactive systems. The Rodin platform [12] provides tool support for modelling and formal verification by theorem proving in Event-B.

Event-B employs a top-down refinement-based approach to system development. The development starts from an abstract system specification that models the most essential behaviour and properties. Each refinement step introduces a representation of more detailed requirements into the system

model. This results in elaborating on data structures, dynamic behaviour and properties of the model. The logical consistency of system models and correctness of refinement steps are verified by mathematical proofs.

2.1. Event-B Language

In Event-B, a system specification is defined using the notion of an *abstract state machine*. An abstract state machine encapsulates the model state, represented as a collection of model variables, and defines operations on this state via machine *events*. The occurrence of events together with the corresponding state changes represents the system behaviour.

Usually, an Event-B machine has an accompanying component called *context*. A context component can include user-defined carrier sets (types) as well as constants and their properties, which are given as a list of model axioms. In a most general form, an Event-B model can be defined as follows.

Definition 1. *An Event-B model is a tuple* $(\mathcal{C}, \mathcal{S}, \mathcal{A}, \upsilon, \Sigma, \mathcal{I}, \mathcal{E}, Init)$*, where:*

- $\mathcal{C}$ *is a set of model constants;*
- $\mathcal{S}$ *is a set of model sets (types);*
- $\mathcal{A}$ *is a set of axioms over* $\mathcal{C}$ *and* $\mathcal{S}$*;*
- υ *is a set of model variables;*
- Σ *is the model state space, which is defined by all possible valuations of the model variables* υ*;*
- $\mathcal{I}$ *is the model invariant defined as a state predicate, i.e.,* $\mathcal{I} : \Sigma \rightarrow Bool$*;*
- $\mathcal{E}$ *is a non-empty set of model events, where each event* e*,* $e \in \mathcal{E}$*, is defined as a binary state relation, i.e.,* $e : \Sigma \times \Sigma \rightarrow Bool$*;*
- *Init is a predicate defining an non-empty set of model initial states.*

The model variables υ are strongly typed by the constraining predicates specified in the invariant $\mathcal{I}$ and initialised by the values satisfying the predicate *Init*. Furthermore, $\mathcal{I}$ may define other important properties that must be preserved by the system during its execution.

While specifying an event, we rely on the following syntax:

$$e \mathrel{\widehat{=}} \textbf{any } a \textbf{ where } G_e \textbf{ then } R_e \textbf{ end},$$

where e is the event name, a is a list of local variables of the event, and G_e is the *event guard* – a model state predicate $G_e : \Sigma \rightarrow Bool$. The *event action*

$R_e : \Sigma \times \Sigma \to Bool$ is defined as a binary relation expressing the relationship between the system states before and after event execution.

The event guard G_e defines the conditions under which such an execution can be performed, i.e., when the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution nondeterministically.

The event action R_e is usually specified as a parallel composition of state assignments. These assignments can be either deterministic or nondeterministic. A deterministic assignment $x := E(x, y)$, where $x, y \subseteq \upsilon$, has the standard syntax and meaning. A nondeterministic assignment is denoted either as $x :\in S$, where S is a set of values, or $x :| P(x, y, x')$, where P is a predicate relating the initial values of variables x and y to some final value of x, denoted as x'. As a result of such non-deterministic assignments, the variables x can get any value either belonging to S or according to P.

If an event does not have local variables, it can be described simply as

$$e \mathrel{\widehat{=}} \textbf{when } G_e \textbf{ then } R_e \textbf{ end}.$$

Without loss of generality, it suffices to consider only such simple events because any event specified using local variables can be always rewritten in this simple form.

2.2. Event-B Semantics: Model Events

Essentially, an event is a relation describing the corresponding state transformation from σ to σ', such that

$$e(\sigma, \sigma') \;=\; \mathcal{I}(\sigma) \;\wedge\; G_e(\sigma) \wedge R_e(\sigma, \sigma').$$

Here we treat the model invariant $\mathcal{I}$ as an implicit event guard. Note that, due to the possible presence of nondeterminism, the successor state σ' is not necessarily unique.

In other words, the semantics of a single model event is given as a binary relation between pre- and post-states of the event. To clarify this relationship, we define two functions before and after of the type $\mathcal{E} \to \mathbf{2}^{\Sigma}$ in a way similar to [13, 14]:

$$\mathsf{before}(e) = \{\sigma : \Sigma \mid \mathcal{I}(\sigma) \wedge G_e(\sigma)\} \quad \text{and}$$
$$\mathsf{after}(e) = \{\sigma' : \Sigma \mid \exists \sigma : \Sigma \bullet\; \mathcal{I}(\sigma) \wedge G_e(\sigma) \wedge R_e(\sigma, \sigma')\}.$$

The latter definition can be also rewritten as follows:

$$\mathsf{after}(e) = \{\sigma' : \Sigma \mid \exists \sigma : \Sigma \cdot \ \sigma \in \mathsf{before}(e) \wedge R_e(\sigma, \sigma')\}.$$

Clearly, for a given event $e \in \mathcal{E}$ and any state $\sigma \in \Sigma$, e is enabled in σ if and only if $\sigma \in \mathsf{before}(e)$.

To consider event execution starting from a particular pre-state σ, we also introduce a "narrowed down" with respect to a fixed pre-state σ, version of the function after:

$$\mathsf{after}_\sigma(e) = \{\sigma' : \Sigma \mid \mathcal{I}(\sigma) \wedge G_e(\sigma) \wedge R_e(\sigma, \sigma')\}.$$

We can lift the above functions before and after for any set of the given events E, $E \subseteq \mathcal{E}$:

$$\mathsf{before}(E) = \bigcup_{e \in E} \mathsf{before}(e) \quad \text{and} \quad \mathsf{after}(E) = \bigcup_{e \in E} \mathsf{after}(e).$$

In the special case when $E = \mathcal{E}$, the resulting set $\mathsf{before}(\mathcal{E})$ contains all the states when the modelled system is operational, i.e., when at least one event is enabled. Correspondingly, the complement of $\mathsf{before}(\mathcal{E})$ gives us those system states that, once reached, put the system into deadlock:

$$\mathsf{deadlocks}(\mathcal{E}) = \Sigma \setminus \mathsf{before}(\mathcal{E}).$$

2.3. Event-B Semantics: Initial Model

The semantics of an entire Event-B model is completed by formulating a number of conditions – *proof obligations*, expressed in the form of logical sequents. In this paper we present only several of the most important proof obligations that should be verified for the initial and refined models. The full list of proof obligations can be found in [1].

In this paper we will heavily rely on the semantic functions before and after defined above. To keep our formalisation consistent and concise, we formulate all the presented proof obligations in terms of these functions.

The initial Event-B model should satisfy the event feasibility and invariant preservation properties. For each event e, its feasibility means that, whenever the event is enabled (in some particular state σ), its next-state relation is well-defined, i.e., there exists some reachable after-state:

$$\mathcal{A},\ \sigma \in \mathsf{before}(e) \ \vdash \ \exists \sigma' \cdot \sigma' \in \mathsf{after}_\sigma(e) \qquad \text{(FIS)}$$

Each event e of an Event-B model should also preserve the model invariant:

$$\mathcal{A},\, \sigma' \in \mathsf{after}(e) \;\vdash\; \mathcal{I}(\sigma') \qquad \text{(INV)}$$

Since the initialisation event has no initial state and guard, its invariant preservation proof obligation is simpler:

$$\mathcal{A},\, \mathit{Init}(\sigma') \;\vdash\; \mathcal{I}(\sigma') \qquad \text{(INIT)}$$

2.4. Event-B Semantics: Refinement

Each Event-B refinement step typically introduces new variables and events into a more abstract model. The introduced new events correspond to stuttering steps that are not visible at the abstract level. The old, abstract model events may be also refined to reduce their nondeterminism and provide access to the new variables.

Moreover, Event-B formal development supports data refinement, allowing us to replace some abstract variables with their concrete counterparts. In that case, the invariant of a refined machine is extended (conjuncted) with a so called *gluing invariant* that formally defines the relationship between the abstract and concrete variables.

Let Σ_a, $\mathcal{I}_a$ and $\mathcal{E}_a$ be respectively the state space, invariant, and events of the abstract model. Similarly, let Σ_c, $\mathcal{I}_c$ and $\mathcal{E}_c$ be respectively the state space, invariant, and events of the (concrete) refined model. Finally, let $\mathcal{J}$ be the gluing invariant between Σ_a and Σ_c. To verify correctness of a refinement step, we need to prove a number of proof obligations for the refined model.

The first three proof obligations focus on the connection between the abstract events and their concrete refined versions. Let us assume that an abstract event $e_a \in \mathcal{E}_a$ is refined by a concrete event $e_c \in \mathcal{E}_c$.

The first proof obligation states that the refined event e_c should stay feasible:

$$\mathcal{A},\, \mathcal{I}_a(\sigma_a),\, \mathcal{J}(\sigma_a, \sigma_c),\, \sigma_c \in \mathsf{before}(e_c) \;\vdash\; \exists \sigma_c' \cdot\; \sigma_c' \in \mathsf{after}_{\sigma_c}(e_c) \qquad \text{(REF\_FIS)}$$

The guard of the refined event e_c can be only strengthened in a refinement step:

$$\mathcal{A},\, \mathcal{I}_a(\sigma_a),\, \mathcal{J}(\sigma_a, \sigma_c),\, \sigma_c \in \mathsf{before}(e_c) \;\vdash\; \sigma_a \in \mathsf{before}(e_a) \qquad \text{(REF\_GRD)}$$

The refined event e_c should preserve the concrete invariant $\mathcal{I}_c$. Moreover, its "execution" cannot be contradictory to the one of the abstract event e_a:

$$\mathcal{A},\, \mathcal{I}_a(\sigma_a),\, \mathcal{J}(\sigma_a, \sigma_c),\, \sigma_c' \in \mathsf{after}_{\sigma_c}(e_c) \;\vdash$$

$$\mathcal{I}_c(\sigma'_c) \ \wedge \ \exists \sigma'_a \cdot \sigma'_a \in \mathsf{after}_{\sigma_a}(e_a) \ \wedge \ \mathcal{J}(\sigma'_a, \sigma'_c) \qquad \text{(REF\_INV)}$$

To verify that all the concrete events (both old and new) do not introduce additional deadlocks into the model, we need to prove *relative deadlock freedom*:

$$\mathcal{A},\, \sigma_a \in \mathsf{before}(\mathcal{E}_a),\, \mathcal{J}(\sigma_a, \sigma_c),\, \mathcal{I}_c(\sigma_c) \ \vdash \ \sigma_c \in \mathsf{before}(\mathcal{E}_c) \qquad \text{(REF\_DLF)}$$

Finally, we should demonstrate that the new events do not collectively diverge, i.e., they eventually return control to the old events. This is typically achieved by providing a natural number state expression (*variant*) and showing that each new event decreases it. Let $nvar \in \Sigma_c \to \mathbb{N}$ be the provided variant expression. Let also $e \in \widehat{\mathcal{E}}_c$ be a new concrete event, where $\widehat{\mathcal{E}}_c \subset \mathcal{E}_c$ is a set of new events of the refined model. Then the non-divergence proof obligation for the event e can be presented as follows:

$$\mathcal{A},\, \sigma'_c \in \mathsf{after}_{\sigma_c}(e) \ \vdash \ nvar(\sigma'_c) < nvar(\sigma_c) \qquad \text{(REF\_VAR)}$$

The Event-B refinement process allows us to gradually introduce implementation details, while preserving functional correctness during stepwise model transformation. The model verification effort, in particular, automatic generation and demonstration of the required proof obligations, is significantly facilitated by the provided tool support – the Rodin platform.

Event-B facilitates correct-by-construction development of critical systems. However, to ensure system dependability we should guarantee that the system is not only functionally correct but also meet desired non-functional requirements to system reliability, safety, responsiveness, etc. Since many of these properties depend on time, in the next section we will demonstrate how reliance on the notion of iteration allows us to implicitly introduce a model of time into Event-B specifications of cyclic systems. In its turn, it sets a scene for stochastic modelling of quantitative critical properties.

3. Modelling of Cyclic Systems in Event-B

There is a large class of systems that exhibit a cyclic behaviour, i.e., the systems that iteratively execute a predefined sequence of steps. Typical representatives of cyclic systems are control and monitoring systems. For instance, one iteration of a control system usually includes reading the sensors that monitor the controlled physical processes, processing the obtained sensor values, and finally setting the actuators according to a predefined

control algorithm. In principle, the system could operate in this way indefinitely long. However, unforeseen conditions in the operating environment or component failures may affect the normal system functioning and lead to a shutdown.

3.1. The Event-B Structure and Control Flow of a Cyclic System

Let us start by describing the desired structure and control flow properties of cyclic systems we aim at modelling in Event-B. After completing computations performed at each iteration, the status of a cyclic system should be re-evaluated to decide whether it can continue its operation. Therefore, it is convenient to split a formal model of a cyclic system into two parts. The first part focuses on modelling the *computation* performed at each iteration, e.g., for a control system it would include reading sensor outputs, processing them, setting actuators, etc. The second part consists of a set of *controlling* actions that initiate a new system iteration as well as analyse the system status once an iteration is completed.

Based on this observation, we assume that the events of the initial Event-B model of a cyclic system can be partitioned into three groups. The first group, called e_0, models the computational part of a cyclic system. The remaining two groups, called IN and OUT, model the controlling system actions at the beginning and the end of each system iteration respectively. Without losing generality, from now on we will treat all such groups of events as single events e_0, IN, and OUT (because they can always be merged into the corresponding single events, see, e.g., [1]).

Each iteration of the modelled cyclic system can be represented by the following control flow on model events:

$$IN \longrightarrow e_0 \longrightarrow OUT$$

During the Event-B refinement process, any of the events IN, OUT and e_0 can be refined. For simplicity, in this paper we assume that refinement will focus on elaborating the computation modelled by e_0. This reflects the refinement style that is frequently adopted in Event-B: the detailed representation of an algorithm is introduced in a number of new events $e_1, e_2, ..., e_n$ preceding e_0. In other words, these events model intermediate calculations on new data structures necessary to produce the system result in e_0. Then the control flow of a single iteration of a refined system looks as follows:

$$IN \longrightarrow e_1, ..., e_n \longrightarrow e_0 \longrightarrow OUT$$

Please note that we do not restrict in any way the execution order of the new events $e_1, e_2, ..., e_n$, i.e., any event branching or looping is possible. However, according to the Event-B semantics, $e_1, e_2, ..., e_n$ are not allowed to diverge (see the proof obligation rule REF_VAR), i.e., the new events will eventually return control to the event e_0.

3.2. Formal Requirements for Cyclic Systems

Let us now formally define requirements imposed on the abstract and refined Event-B models of a cyclic system. Similarly to Section 2, we denote by $\mathcal{E}$ the set of all model events. Moreover, let $\widehat{\mathcal{E}}$ be all the new model events introduced during the refinement process. In other words, $\widehat{\mathcal{E}} = \{e_1, e_2, ..., e_n\}$, $\widehat{\mathcal{E}} \subset \mathcal{E}$, and $\mathcal{E} = \{IN, e_0, OUT\} \ \cup \ \widehat{\mathcal{E}}$.

Next we formulate a number of formal requirements that a model of a cyclic system has to satisfy. These properties can be generated and verified as additional proof obligations for a particular model under consideration. Since the properties proved for an Event-B model are preserved for any of its refinements, it is often sufficient to verify these additional proof obligations for newly introduced events only.

$$\{\sigma \in \Sigma \mid Init(\sigma)\} \subseteq \mathsf{before}(IN) \tag{1}$$

$$\mathsf{after}(IN) \subseteq \mathsf{before}(e_0) \cup \mathsf{before}(\widehat{\mathcal{E}}) \tag{2}$$

$$\mathsf{after}(\widehat{\mathcal{E}}) \subseteq \mathsf{before}(\widehat{\mathcal{E}}) \cup \mathsf{before}(e_0) \tag{3}$$

$$\mathsf{after}(e_0) \subseteq \mathsf{before}(OUT) \tag{4}$$

$$\mathsf{after}(OUT) \subseteq \mathsf{before}(IN) \cup \mathsf{deadlocks}(\mathcal{E}) \tag{5}$$

$$\forall e, f \in \{IN, OUT, e_0\} \cdot e \neq f \Rightarrow \mathsf{before}(e) \cap \mathsf{before}(f) = \varnothing \tag{6}$$

$$\forall e \in \{IN, OUT\} \cdot \mathsf{before}(e) \cap \mathsf{before}(\widehat{\mathcal{E}}) = \varnothing \tag{7}$$

The requirement (1) states that the system initialisation should enable the event IN. The requirements (2)–(5) stipulate the desired execution order of the involved model events, informally presented in the control flow given above. Specifically, the event IN must be followed by the event e_0 or any of the new model events (the requirement (2)). The new events may loop or terminate by enabling e_0 (the requirement (3)). The event e_0 is followed by the event OUT (the requirement (4)). Finally, the event OUT may enable a start of a new iteration by the event IN or put the whole system into a deadlock (the requirement (5)).

The last two requirements (6) and (7) stipulate that the guards of the events IN, OUT and e_0 as well as IN, OUT and any new event from $\widehat{\mathcal{E}}$ should be disjoint, i.e., they cannot be enabled at the same time. This allows us to guarantee that the presented control flow is strictly followed.

The presented formulation of additional proof obligations ensuring a specific control flow of events is inspired by the approach given in [13].

3.3. Example: Abstract Event-B Model of a Cyclic System

In this section we present an example of modelling a simple cyclic system in Event-B. It can be easily shown that the modelled system satisfies the given cyclic system requirements (1)–(7).

Figure 1 shows an abstract Event-B model (the machine CS) of such a cyclic system. The controlling events IN and OUT model the start and the end of an iteration, while e_0 is an event abstractly modelling its body – the computation. The boolean variable *active* indicates whether execution of a system iteration is under way, while the variable res abstractly models the result returned by a single iteration. Here, the value ND (meaning "not defined") indicates that the system output has not been produced yet.

Upon execution of the body event e_0, res may obtain one of two values: OK (successful execution) or NOK (a critical system failure). This output is evaluated by the event OUT. After its iteration, the system may stay operational or terminate. In the first case, the system proceeds to its next iteration. Otherwise, the system deadlocks.

We will use the model CS and its refinements as a running example of this paper.

3.4. Observable States and Iterations

While reasoning about quantitative properties of a cyclic system, we are usually interested in the number of iterations that the system can perform before it terminates. This observation allows us to focus only on those system states where a system iteration starts and finishes. We call such system states *observable*. We also distinguish an important subset of the observable states called *operational* states. Usually, essential properties of the system (such as dependability, performance and safety properties) can be guaranteed only while the system stays in the operational states.

Machine CS
Variables $active, res$
Invariants $active \in BOOL,\ res \in \{OK, NOK, ND\}$
Events

$Initialisation \mathrel{\widehat{=}}$
begin
$active, res := FALSE, ND$
end

$OUT \mathrel{\widehat{=}}$
when
$active = TRUE \wedge res \neq ND$
then
$active := FALSE$
$res :| res' \in \{res, ND\} \wedge res' = ND \Leftrightarrow res = OK$
end

$IN \mathrel{\widehat{=}}$
when
$active = FALSE \wedge res = ND$
then
$active := TRUE$
end

$e_0 \mathrel{\widehat{=}}$
when
$active = TRUE \wedge res = ND$
then
$res :\in \{OK, NOK\}$
end

Figure 1: Event-B model of a cyclic system

Definition 2 (Observable states). *For Event-B models satisfying the requirements (1)–(7), we define the observable system states as a set containing all the states where an iteration of a cyclic system may start or finish, i.e.,*

$$\Sigma_{obs} = \mathsf{before}(IN) \cup \mathsf{after}(OUT).$$

From the requirement (5), we can also conclude that

$$\Sigma_{obs} \subseteq \mathsf{before}(IN) \cup \mathsf{deadlocks}(\mathcal{E}). \tag{8}$$

Since $\mathsf{before}(IN) \cap \mathsf{deadlocks}(\mathcal{E}) = \varnothing$, (8) also suggests that the set of observable states can be partitioned into two disjoint subsets of operational and non-operational (or terminating) states:

$$\Sigma_{obs} = \Sigma_{op} \cup \Sigma_{nop},$$

where $\Sigma_{op} = \mathsf{before}(IN)$ and $\Sigma_{nop} = \mathsf{after}(OUT) \backslash \mathsf{before}(IN) \subseteq \mathsf{deadlocks}(\mathcal{E})$.

States that are not in Σ_{obs} are called *unobservable*. Intuitively, introduction of the system observable states Σ_{obs} means that, for the external observer of a cyclic system, the core part of a cyclic system is a "black box"

and only starting and ending points of iterations are visible. Moreover, since in this paper we aim at stochastic reachability analysis of cyclic systems, we assume that the set Σ_{obs} is finite.

Before we formally define the notion of an iteration for the proposed generalised Event-B model of a cyclic system, let us to formulate one useful lemma.

Lemma 1. *If an Event-B model satisfies the requirements (1)–(7), all the model events from* $\{e_0\} \cup \widehat{\mathcal{E}}$ *are defined on unobservable system states only, i.e.,*

$$\forall e \in \{e_0\} \cup \widehat{\mathcal{E}} \cdot (\mathsf{before}(e) \cup \mathsf{after}(e)) \cap \Sigma_{obs} = \varnothing.$$

Proof. We only show the proof for the event e_0. The corresponding proof for any e such that $e \in \widehat{\mathcal{E}}$ is similar.

From the requirement (6), we immediately have that

$$\mathsf{before}(e_0) \cap \mathsf{before}(IN) = \varnothing.$$

Moreover, by the definition of $\mathsf{deadlocks}(\mathcal{E})$, the following is true:

$$\mathsf{before}(e_0) \cap \mathsf{deadlocks}(\mathcal{E}) = \varnothing.$$

From these two propositions and the property (8), we have

$$\mathsf{before}(e_0) \cap \Sigma_{obs} = \varnothing. \tag{9}$$

Similarly, from the requirements (4) and (6), we can conclude that

$$\mathsf{after}(e_0) \cap \mathsf{before}(IN) = \varnothing.$$

From the requirement (4) and the definition of $\mathsf{deadlocks}(\mathcal{E})$, we also get

$$\mathsf{after}(e_0) \cap \mathsf{deadlocks}(\mathcal{E}) = \varnothing.$$

From the last two propositions and the property (8), we have

$$\mathsf{after}(e_0) \cap \Sigma_{obs} = \varnothing. \tag{10}$$

Finally, from (9) and (10), we conclude that

$$(\mathsf{before}(e_0) \cup \mathsf{after}(e_0)) \ \cap \ \Sigma_{obs} = \varnothing.$$

■

Each iteration of a cyclic system maps the current operational system state $\sigma \in \Sigma_{op}$ into a subset of Σ_{obs}. The resulting set of states represents all possible states that can be reached due to the system nondeterministic behaviour. Formally, we can define it as follows:

Definition 3 (Iteration). *An iteration of a cyclic system is a total function mapping the set of operational states to the powerset of observable system states*

$$iter \in \Sigma_{op} \rightarrow \mathbf{2}^{\Sigma_{obs}},$$

such that, for any subsequent observable states σ_i and σ_{i+1} in a system execution trace,

$$\sigma_{i+1} \in iter(\sigma_i).$$

Theorem 1. *If an Event-B model satisfies the requirements (1)–(7), the modelled system is cyclic and its iteration function iter can be defined on all the system operational states.*

Proof. Let us first consider the case when $\widehat{\mathcal{E}} = \varnothing$.

In that case, the requirements (1)–(7) guarantee that the system repeatedly executes the events IN, e_0, and OUT in the fixed order. Moreover, Lemma 1 states that the intermediate states, i.e., the pre- and post-states of the event e_0, are unobservable. This means that we can treat the events IN, e_0, and OUT as a single event

$$IN; e_0; OUT,$$

where ";" denotes the relational composition operator.

Then we can define the function $iter$ as

$$iter(\sigma) \;=\; \mathsf{after}_\sigma(IN; e_0; OUT)$$

for any $\sigma \in \mathsf{before}(IN)$.

In the case of $\widehat{\mathcal{E}} \neq \varnothing$, we rely on the proof obligation (REF_VAR), which states that the new events cannot diverge. This allows us to represent the overall execution of the new events as a single composite event $(\widehat{e})^*$, where $\widehat{e} = \bigcup_{e \in \widehat{\mathcal{E}}} e$ and * denotes the transitive relational closure operator.

Moreover, the requirements (1)–(7) guarantee that the system is now repeatedly executed as a composite event

$$IN; (\widehat{e})^*; e_0; OUT,$$

when Lemma 1 again enforces that all the intermediate states from $\bigcup_{e \in \{e_0\} \cup \widehat{\mathcal{E}}} (\mathsf{before}(e) \cup \mathsf{after}(e))$ are unobservable.

Similarly as above, we can now define the function *iter* as

$$iter(\sigma) \;=\; \mathsf{after}_\sigma(IN; (\widehat{e})^*; e_0; OUT)$$

for any $\sigma \in \mathsf{before}(IN)$.

■

Essentially, Theorem 1 postulates that, no matter what refinement steps are taken, the Event-B refinement process will preserve the cyclic nature of a given system, provided that the formulated requirements (1)–(7) are verified. This means that we can use the notion of a system iteration as a *discrete unit of time* defining a unified time scale for any Event-B machine in the refinement chain.

The given requirements for modelling cyclic systems defined above narrow down the class of considered Event-B models. However, this makes such models amenable for integrating stochastic reasoning about the system behaviour. To achieve this goal, we first propose a semantic extension of the Event-B language.

4. Stochastic Modelling in Event-B

Hallerstede and Hoang [15] have extended the Event-B framework with a new operator – *qualitative probabilistic choice*, denoted $\oplus|$. This operator assigns new values to state variables with some positive but generally unknown probability. The proposed extension aims at introducing into Event-B the concept of "almost-certain convergence"– probabilistically certain termination of new event operations introduced by model refinement. The new operator can only replace a nondeterministic choice (assignment) statement in the event actions. It has been shown that any probabilistic choice statement always refines its demonic nondeterministic counterpart [16]. Hence such an extension is not interfering with the established refinement process.

In our previous work [17], we have proposed extending the Event-B modelling language with *quantitative probabilistic choice*, also denoted $\oplus|$. The introduced operator allows us to represent a precise probabilistic information

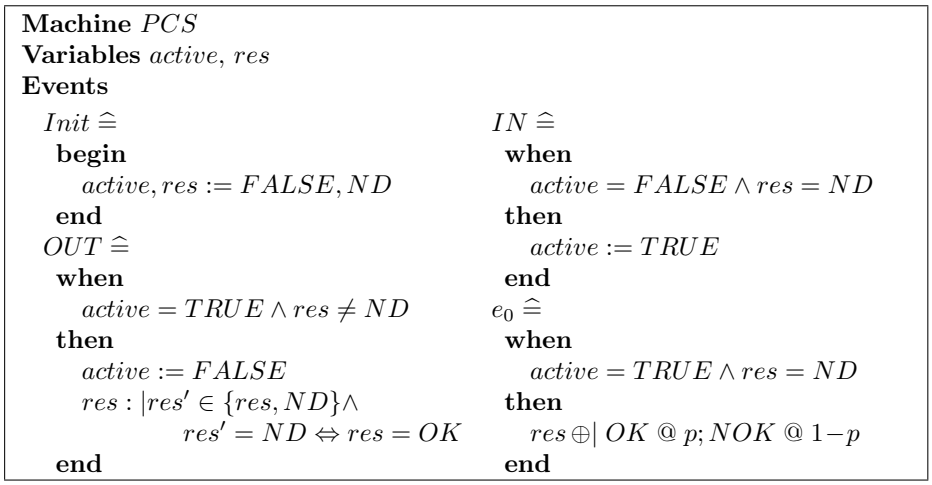


Figure 2: Cyclic system: introducing probabilities

about how likely a particular choice should be made. In other words, it behaves according to some known probabilistic distribution. The quantitative probabilistic choice (assignment) has the following syntax

$$x \oplus| \; x_1 \; @ \; p_1; \ldots; x_n \; @ \; p_n,$$

where $\sum_{i=1}^{n} p_i = 1$. It assigns to the variable x a new value x_i with the corresponding non-zero probability p_i. Similarly to Hallerstede and Hoang, we have restricted the use of the new probabilistic choice operator by introducing it only to replace the existing demonic one. Therefore, we can rely on the Event-B proof obligations to guarantee functional correctness of a refinement step. Moreover, the probabilistic information introduced in new quantitative probabilistic choices can be used to stochastically evaluate certain non-functional system properties as well as their preservation during the refinement process.

To illustrate the proposed extension, in Figure 2 we present a probabilistic refinement of the abstract machine CS. In CS, the result returned by a single iteration is modelled nondeterministically. In the refined model PCS, the nondeterministic choice is replaced by a probabilistic one, where the non-zero constant probabilities p and $1-p$ express how likely the variable res is getting value OK or NOK. According to the theory of probabilistic refinement [16],

the machine PCS is a refinement of the machine CS.

The proposed probabilistic choice operator allows us to introduce a specific probabilistic information into Event-B models and, as a result, model (at least some subset of) probabilistic systems. Our goal, however, is to integrate stochastic reasoning into the entire Event-B development process. In the next section we will show how the notion of Event-B refinement can be strengthened to quantitatively demonstrate that the refined system is "better" (e.g., more reliable or responsive) than its abstract counterpart.

5. Modelling Fully Probabilistic Cyclic Systems

In this section we present a theoretical basis for formal verification of probabilistic cyclic systems in Event-B. We rely on the structure and properties for cyclic systems introduced in Section 3.

5.1. Probability Distribution

Since an Event-B model is essentially a state transition system, we can simulate its execution by producing a tree of reachable states. Each path in such a tree corresponds to one operational trace, while tree branching occurs due the presence of nondeterminism in an Event-B model. If we replace a particular nondeterministic choice by a probabilistic one, we essentially attach concrete weights (probabilities) to separate branches, reflecting how likely a particular branch will be chosen for execution.

Based on that, we can distinguish between two types of modelled systems – fully probabilistic systems, i.e., the systems containing only probabilistic branching, and the systems that behave both probabilistically and nondeterministically. The absence of nondeterminism in a fully probabilistic system additionally imposes a certain restriction on its initialisation event. Specifically, it can be either deterministic or probabilistic assignment.

Let us first consider fully probabilistic systems. The quantitative information present in a probabilistic Event-B model allows us to lift the notion of the system state to that of a probabilistic distribution over the system state:

Definition 4 (Probability distribution). *For the system observable state space Σ_{obs}, the set of distributions over Σ_{obs} is*

$$\overline{\Sigma}_{obs} \;\widehat{=}\; \{\Delta : \Sigma_{obs} \to [0,1] \mid \sum_{\sigma \in \Sigma_{obs}} \Delta(\sigma) = 1\}.$$

Each iteration of a fully probabilistic cyclic system maps some initial operational state to a subset of Σ_{obs} according to some probabilistic distribution, i.e., we can define a single iteration of a probabilistic cyclic system as a total function

$$piter : \Sigma_{op} \rightarrow \overline{\Sigma}_{obs}.$$

There is a simple connection between the iteration *iter* of a cyclic system and its probabilistic counterpart *piter* – if some state σ' can be reached from a current state σ by *piter* with a non-zero probability then it is also reachable by *iter*:

$$\forall \sigma \in \Sigma_{op}, \sigma' \in \Sigma_{obs} \cdot piter(\sigma)(\sigma') > 0 \Rightarrow \sigma' \in iter(\sigma).$$

For example, for our abstract models CS and PCS, both Σ_{op} and Σ_{nop} are singleton sets such that $\Sigma_{op} = \{\sigma_1\} = \{(active = FALSE, res = ND)\}$ and $\Sigma_{nop} = \{\sigma_2\} = \{(active = FALSE, res = NOK)\}$. It is straightforward to see that the iteration *iter* of CS is defined as

$$iter(\sigma_1) = \{\sigma_1, \sigma_2\},$$

while the probabilistic iteration function *piter* for the model PCS is

$$piter(\sigma_1) = \Delta_{\sigma_1}, \text{ such that } \Delta_{\sigma_1}(\sigma_1) = p \text{ and } \Delta_{\sigma_1}(\sigma_2) = 1-p.$$

For any state $\sigma \in \Sigma_{op}$, its distribution Δ_σ (where $\Delta_\sigma = piter(\sigma)$) is calculated from probabilistic choice statements present in a model. However, once the system terminates, it stays in a terminating state forever. This means that, for any state $\sigma \in \Sigma_{nop}$, its distribution Δ_σ is such that $\Delta_\sigma(\sigma) = 1$ and $\Delta_\sigma(\sigma') = 0$, if $\sigma' \neq \sigma$.

5.2. Definition of Quantitative Refinement

While developing a complex software system, the designer often should define critical non-functional constraints, such as required dependability or performance properties. These constraints explicitly describe the desired parameters of the system functioning and must be then taken into account during the development process. In this paper we focus on reasoning about two such constraints – the system *reliability* and *responsiveness (response time)*.

Often, it is not possible to formally guarantee that the system always satisfies a desired critical property. However, we can still assess the probability

that the property is preserved by the system at a certain moment. Currently, Event-B does not explicitly support the notions of time and probability. In the previous sections we proposed a general approach for modelling cyclic systems in Event-B, where the progress of time is modelled by system iterations. Moreover, we proposed the semantic extension of the modelling language that allows us to augment Event-B models with probabilistic information about the system behaviour. Based on this information, we can strengthen the notion of Event-B refinement by additionally requiring that refined models meet reliability and responsiveness requirements with a higher probability.

Let us first consider system reliability. In engineering, reliability is generally measured by the probability that an entity $\mathcal{X}$ can perform a required function under given conditions for the time interval $[0, t]$:

$$R(t) = \mathbf{P}\{\mathcal{X} \textit{ not failed over time } [0, t]\}.$$

Hence, for cyclic systems, reliability can be expressed as the probability that the system remains *operational* during a certain number of iterations. Let $X(t)$ be a function that returns the system state after t-th iteration, where $t \in \mathbb{N}$, and $X(0)$ is an initial system state such that $X(0) \in \Sigma_{op}$. Then we can formally define the system reliability as follows:

$$R(t) = \mathbf{P}\{\Box^{\leq t}\,(X(t) \in \Sigma_{op})\}.$$

Here we use the modal (temporal logic) operator $\Box$, and the formula $(\Box^{\leq t}\,\phi)$ means that ϕ holds *globally* for the first t iterations. It is straightforward to see that this property corresponds to the standard definition of reliability given above. Thus, while modelling a cyclic system, we can strengthen the notion of Event-B refinement from the reliability point of view in the following way:

Definition 5 (Reliability refinement). *Let M_a and M_c be two probabilistic Event-B models of cyclic systems. Moreover, let Σ^a_{op} and Σ^c_{op} be the sets of operational states of M_a and M_c correspondingly. Then we say that M_c is a reliability refinement of M_a if and only if*

1. *M_c is an Event-B refinement of M_a ($M_a \sqsubseteq M_c$), and*
2. $\forall t \in \mathbb{N}_1 \cdot \mathbf{P}\{\Box^{\leq t}\,(X_a(t) \in \Sigma^a_{op})\} \leq \mathbf{P}\{\Box^{\leq t}\,(X_c(t) \in \Sigma^c_{op})\}.$ (11)

The second condition essentially requires that the system reliability cannot decrease during the refinement process.

Dually, for a cyclic system that terminates by providing some particular service to the customers, our goal is to assess the probability that this service will be provided during a certain time interval. This property can be expressed as the probability that $X(t)$ eventually falls into Σ_{nop} within first t iterations, i.e., the probability that the system will *terminate* during the time interval $[0, t]$:

$$Q(t) = \mathbf{P}\{\Diamond^{\leq t}\,(X(t) \in \Sigma_{nop})\}.$$

Here $\Diamond^{\leq t}$ is a modal operator denoting "eventually, within time $[0, t]$".

Therefore, from the responsiveness point of view, we can strengthen the definition of Event-B refinement by also requiring that the refined system should be at least as responsive as the abstract one:

Definition 6 (Responsiveness refinement). *Let M_a and M_c be two probabilistic Event-B models of cyclic systems. Moreover, let Σ^a_{nop} and Σ^c_{nop} be the sets of non-operational states of M_a and M_c correspondingly. Then we say that M_c is a responsiveness refinement of M_a if and only if*

1. *M_c is an Event-B refinement of M_a ($M_a \sqsubseteq M_c$), and*
2. $\forall t \in \mathbb{N}_1 \cdot \mathbf{P}\{\Diamond^{\leq t}\,(X_a(t) \in \Sigma^a_{nop})\} \leq \mathbf{P}\{\Diamond^{\leq t}\,(X_c(t) \in \Sigma^c_{nop})\}.$ (12)

The second condition essentially requires that the system responsiveness cannot decrease during the refinement process.

Remark 1. *If the second, quantitative refinement condition of Definitions 5 and 6 holds not for all t, but for some interval $t \in 1..T$, $T \in \mathbb{N}_1$, we say that M_c is a partial reliability (responsiveness) refinement of M_a for $t \leq T$.*

5.3. Verification of Quantitative Refinement

To verify the first refinement condition of Definitions 5 and 6, we rely on the proof obligation rules that we discussed in Section 2. Event-B tool support – the Rodin platform – provides us with a means for generating and proving of all the required proof obligations, including the formalised requirements (1)–(7) for cyclic systems given in Section 3. However, it lacks the functionality needed to quantitatively verify refinement conditions (11) and (12). In this subsection we give a theoretical background that allows us

to express the probabilistic reachability properties (11) and (12) in terms of the operational and non-operational states of cyclic systems.

Let us now consider in detail the behaviour of a fully probabilistic cyclic system M. We assume that the initial system state σ is determined by some probability distribution Δ_0 over the set of operational states Σ_{op} (which also covers the case of deterministic initialisation). After its first iteration, the system reaches some state $\sigma' \in \Sigma_{obs}$ with the probability $\Delta_\sigma(\sigma')$. At this point, if $\sigma' \in \Sigma_{nop}$, the system terminates. Otherwise, the system starts a new iteration and, as a result, reaches some state σ'' with the probability $\Delta_{\sigma'}(\sigma'')$, and so on. This process is completely defined by its state transition matrix P_M. More precisely, given that the state space Σ_{obs} is finite, we can enumerate it, i.e., assume that $\Sigma_{obs} = \{\sigma_1, \ldots \sigma_n\}$, and define elements of the $n \times n$ transition matrix P_M as

$$\forall i, j \in 1..n \cdot P_M(\sigma_i, \sigma_j) \mathrel{\widehat{=}} \Delta_{\sigma_i}(\sigma_j) = \begin{cases} piter(\sigma_i)(\sigma_j) & \text{if } \sigma_i \in \Sigma_{op}, \\ 1 & \text{if } \sigma_i \in \Sigma_{nop} \text{ and } i = j, \\ 0 & \text{if } \sigma_i \in \Sigma_{nop} \text{ and } i \neq j. \end{cases}$$

In its turn, this matrix unambiguously defines the underlying Markov process – the absorbing discrete time Markov chain [9], with the set of transient states Σ_{op} and the set of absorbing states Σ_{nop}. The state transition matrix of a Markov process together with its initial state allows us to calculate the probability that the defined Markov process, after a given number t of steps, will be in some particular state σ. Using the transition matrix P_M, we now can assess reliability and responsiveness of Event-B models as follows:

Proposition 1. *The reliability refinement condition (11) is equivalent to*

$$\forall t \in \mathbb{N}_1 \cdot \sum_{\sigma \in \Sigma^a_{op}} \left([\Delta^a_0] \cdot P^t_{M_a} \right)(\sigma) \leq \sum_{\sigma \in \Sigma^c_{op}} \left([\Delta^c_0] \cdot P^t_{M_c} \right)(\sigma),$$

where Σ^a_{op} and Σ^c_{op} are the sets of operational states of the systems M_a and M_c respectively, $[\Delta^a_0]$ and $[\Delta^c_0]$ are the initial state distribution row-vectors, and P^t is the matrix P raised to the power t.

Proof. Directly follows from our definition of the observable state space and fundamental theorems of the theory of Markov chains. ∎

Proposition 2. *The responsiveness refinement condition (12) is equivalent to*

$$\forall t \in \mathbb{N}_1 \cdot \sum_{\sigma \in \Sigma^a_{nop}} \left([\Delta^a_0] \cdot P^t_{M_a}\right)(\sigma) \leq \sum_{\sigma \in \Sigma^c_{nop}} \left([\Delta^c_0] \cdot P^t_{M_c}\right)(\sigma),$$

where Σ^a_{nop} and Σ^c_{nop} are the sets of terminating states of the systems M_a and M_c respectively, $[\Delta^a_0]$ and $[\Delta^c_0]$ are the initial state distribution row-vectors, and P^t is the matrix P raised to the power t.

Proof. Similar to Proposition 1. ∎

To illustrate the use of our definitions of quantitative refinement in practice, let us revisit our simple example. To increase the reliability of our cyclic system, we refine the model as follows. In the case when the previously acquired result is not acceptable, the system now retries to obtain a new result. The number of such attempts is bounded by a predefined positive constant $N \in \mathbb{N}_1$. The resulting Event-B model $RPCS$ is presented in Figure 3. Here the variable att represents the number of performed attempts. Moreover, a new event e_1 is introduced to probabilistically model possible multiple retries, where a new variable x stores the outcome of the last attempt. Once x is assigned the value OK or the system reaches the maximum limit of attempts, the value of x is copied to the variable res in the refined version of the event e_0.

The Event-B machine $RPCS$ can be proved to be a probabilistic reliability refinement of its abstract probabilistic model (the machine PCS in Figure 2) according to Definition 5. Indeed, it is easy to see that

$$R_{PCS}(t) = \mathbf{P}\{\Box^{\leq t}\,(X_{PCS}(t) \in \Sigma_{op})\} = p^t,$$

while

$$R_{RPCS}(t) = \mathbf{P}\{\Box^{\leq t}\,(X_{RPCS}(t) \in \Sigma_{op})\} = (1 - (1-p)^N)^t$$

and, finally, for any given values of p and N,

$$\forall t \in \mathbb{N}_1 \cdot R_{PCS}(t) \leq R_{RPCS}(t).$$

The presented simple example demonstrates how we can incorporate formal reasoning about system reliability into the refinement process in Event-B. Since system responsiveness is the (mathematically) dual property, it can be modelled and verified in a similar way.

In the next section we generalise our approach to the systems that combines both nondeterministic and probabilistic behaviour.

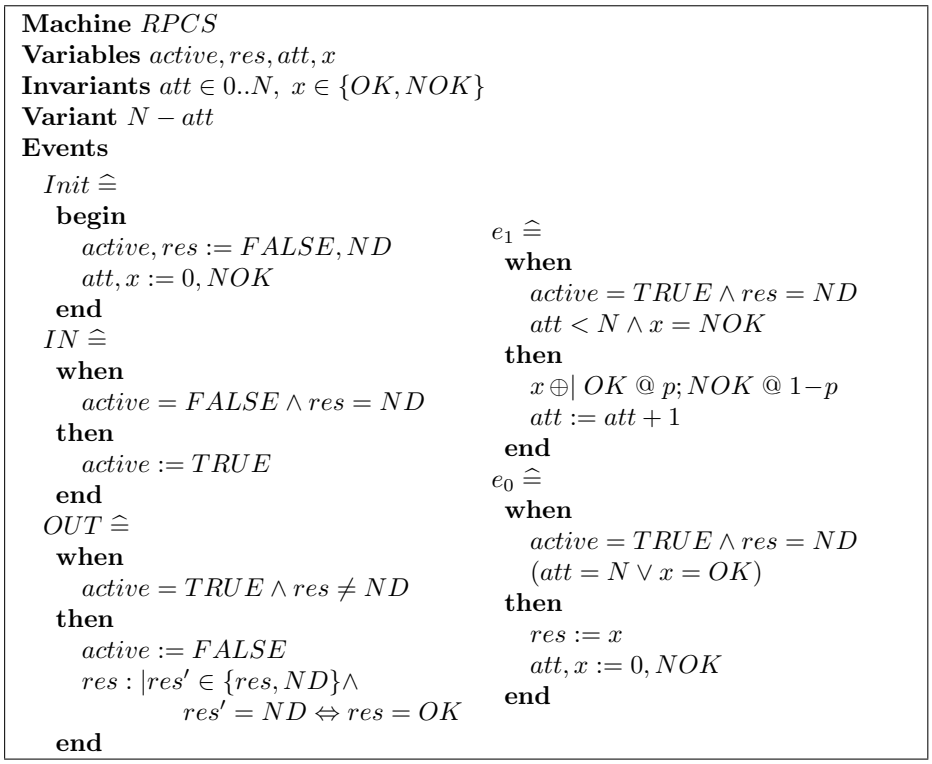


Figure 3: Cyclic system: probabilistic reliability refinement

6. Modelling Probabilistic Cyclic Systems with Nondeterminism

It is not always possible to give precise probabilistic information for all nondeterministic choices in a specification of a cyclic system. As a result, system models often contain a mixture of nondeterministic and probabilistic choices, making reasoning about such systems more complicated. In this section we offer our solution to this problem, which is a generalisation of our approach presented in Section 5.

6.1. Extended Definition of Quantitative Refinement

For a cyclic system containing both probabilistic and nondeterministic choices, we define a single iteration as a total function

$$npiter : \Sigma_{op} \rightarrow \mathbf{2}^{\overline{\Sigma}_{obs}},$$

i.e., this function maps a given observable operational state, $\sigma \in \Sigma_{op}$, into a *set of distributions* over the observable state space Σ_{obs}. The resulting set of distributions is built for all possible combinations of nondeterministic choices (i.e., execution traces) of a single system iteration.

Similarly as in the case of a fully probabilistic system, there is a simple connection between the iteration *iter* of a cyclic system and its nondeterministic-probabilistic counterpart *npiter*. Specifically, if some state σ' can be reached from a current state σ with a non-zero probability according to some distribution from $npiter(\sigma)$ then it is also reachable by *iter*:

$$\forall \sigma \in \Sigma_{op}, \sigma' \in \Sigma_{obs} \cdot (\exists \Delta_\sigma \in npiter(\sigma) \cdot \Delta_\sigma(\sigma') > 0) \Rightarrow \sigma' \in iter(\sigma).$$

Now let us consider the behaviour of some nondeterministic-probabilistic cyclic system M in detail. We can assume that the initial system state σ belongs to the set of operational states Σ_{op}. After its first iteration, the system nondeterministically chooses some distribution Δ_σ from $npiter(\sigma)$ and then, according to this distribution, reaches some state σ' with the non-zero probability $\Delta_\sigma(\sigma')$. At this point, if $\sigma' \in \Sigma_{nop}$, the system terminates. Otherwise, the system starts a new iteration. It is easy to see that the behavioural semantics of a nondeterministic-probabilistic cyclic system in Event-B is defined by a Markov decision process with the absorbing set Σ_{nop} [10, 11].

Nondeterminism has the demonic nature in Event-B [18], i.e., we do not have any control or information about which branch of execution will be chosen. Therefore, while reasoning about system reliability and responsiveness, we have to consider the worst case scenario by always choosing the "worst" of available distributions. From the reliability and responsiveness perspective, it means that, while evaluating these properties of a probabilistic cyclic system with nondeterminism, we need to obtain the lowest bound of the performed evaluation. Therefore, we re-formulate the definitions of the reliability and responsiveness refinement for probabilistic cyclic systems as follows:

Definition 7 (Reliability refinement). *Let M_a and M_c be two nondeterministic-probabilistic Event-B models of cyclic systems. Moreover, let Σ^a_{op} and Σ^c_{op} be the sets of operational states of M_a and M_c correspondingly. Then we say that M_c is a reliability refinement of M_a if and only if*

1. *M_c is an Event-B refinement of M_a ($M_a \sqsubseteq M_c$), and*
2. $\forall t \in \mathbb{N}_1 \cdot \mathbf{P}_{min}\{\Box^{\leq t}\,(X_a(t) \in \Sigma^a_{op})\} \leq \mathbf{P}_{min}\{\Box^{\leq t}\,(X_c(t) \in \Sigma^c_{op})\}, \quad (13)$

where $\mathbf{P}_{min}\{\Box^{\leq t}\,(X(t) \in \Sigma_{op})\}$ *is the minimum probability that the system remains operational during the first* t *iterations.*

Definition 8 (Responsiveness refinement). *Let* M_a *and* M_c *be two nondeterministic-probabilistic Event-B models of cyclic systems. Moreover, let* Σ^a_{nop} *and* Σ^c_{nop} *be the sets of non-operational states of* M_a *and* M_c *correspondingly. Then we say that* M_c *is a responsiveness refinement of* M_a *if and only if*

1. M_c *is an Event-B refinement of* M_a *(*$M_a \sqsubseteq M_c$*), and*
2. $\forall t \in \mathbb{N}_1 \cdot \mathbf{P}_{min}\{\Diamond^{\leq t}\,(X_a(t) \in \Sigma^a_{nop})\} \leq \mathbf{P}_{min}\{\Diamond^{\leq t}\,(X_c(t) \in \Sigma^c_{nop})\}, (14)$

where $\mathbf{P}_{min}\{\Diamond^{\leq t}\,(X(t) \in \Sigma_{nop})\}$ *is the minimum probability that the system terminates during the first* t *iterations.*

Remark 2. *If the second, quantitative refinement condition of Definitions 7 and 8 holds not for all* t*, but for some interval* $t \in 1..T$*,* $T \in \mathbb{N}_1$*, we say that* M_c *is a partial reliability (responsiveness) refinement of* M_a *for* $t \leq T$*.*

6.2. Verification of Quantitative Refinement

To evaluate the worst case reliability and responsiveness for probabilistic systems with nondeterminism, we have to calculate the minimum probabilities participating in (13) and (14). Let us assume that some cyclic system M is in an operational state σ, while *npiter* maps σ to the *finite* set of distributions $\overline{\Delta}_\sigma = \{\Delta^1_\sigma, \Delta^2_\sigma, \dots\}$. If we want to evaluate the worst case reliability of the system for this iteration, we just have to choose the distribution that maps σ to the set of operational states with the minimal probability, i.e., the probability $\min\limits_{\Delta \in \overline{\Delta}_\sigma} \sum\limits_{\sigma' \in \Sigma_{op}} \Delta(\sigma')$.

Similarly, to evaluate the worst case responsiveness of the system, we have to choose the distribution that maps σ to the set of terminating states with the minimal probability, i.e., the probability $\min\limits_{\Delta \in \overline{\Delta}_\sigma} \sum\limits_{\sigma' \in \Sigma_{nop}} \Delta(\sigma')$.

However, when the goal is to evaluate the worst case stochastic behaviour of the system within a time interval $[0, t]$, where $t \geq 1$, the calculation process of the resulting minimal probability becomes more complex. Because of the intricate nature of demonic nondeterminism, we cannot simply rely on the calculated fixed minimal probability for t iterations when calculating it for

$t+1$ iterations. The "demon" does not have to stick to its previous choices, so the minimal probability has to re-calculated anew, now for $t+1$ iterations.

Let us first consider evaluation of system reliability. We define the worst case reliability as a function $r(t, \sigma)$, the arguments of which are the number of system iterations t and some initial state σ. For now, we assume that the initial system state σ is deterministically defined. Later we consider more general cases when σ is given by some initial probability distribution or nondeterministically. The function $r(t, \sigma)$ returns the minimal value of the reliability function $R(t)$ over all possible state distributions.

The definition of $r(t, \sigma)$ is recursive. Two basic cases define the function values for the terminating (absorbing) states and the zero number of iterations respectively:

$$\sigma \in \Sigma_{nop} \Rightarrow \forall t \in \mathbb{N} \cdot r(t, \sigma) = 0,$$
$$\sigma \in \Sigma_{op} \Rightarrow r(0, \sigma) = 1.$$

Generally, for $\sigma \in \Sigma_{op}$, we can recursively define the function $r(t, \sigma)$ in the following way:

$$\forall t \in \mathbb{N}_1, \sigma \in \Sigma_{op} \cdot r(t, \sigma) = \min_{\Delta \in \overline{\Delta}_\sigma} \sum_{\sigma' \in \Sigma_{op}} \Delta(\sigma') \cdot r(t-1, \sigma').$$

Such an approach for defining an "absorbing" function is often used in the works based on Markov decision processes with absorbing sets (see [19] for instance). Please note that the recursive function application essentially traverses all the possible operational state transitions and, based on that, operational state distributions, and then finds the minimal probability of the system staying operational.

Similarly, the stochastic evaluation of system responsiveness is based on the recursive function $q(t, \sigma)$. It returns the minimal probability of the system terminating within the time interval $[0, t]$, when starting in the observable state σ. The basic cases are

$$\sigma \in \Sigma_{nop} \Rightarrow \forall t \in \mathbb{N} \cdot q(t, \sigma) = 1,$$

$$\sigma \in \Sigma_{op} \Rightarrow q(0, \sigma) = 0.$$

Generally, for $\sigma \in \Sigma_{op}$, we can recursively define the function $q(t, \sigma)$ in the

following way:

$$\forall t \in \mathbb{N}_1, \sigma \in \Sigma_{op} \cdot q(t,\sigma) = \min_{\Delta \in \overline{\Delta}_\sigma} \left[\sum_{\sigma' \in \Sigma_{op}} \Delta(\sigma') \cdot q(t-1,\sigma') + \sum_{\sigma' \in \Sigma_{nop}} \Delta(\sigma') \right] = \min_{\Delta \in \overline{\Delta}_\sigma} \sum_{\sigma' \in \Sigma_{obs}} \Delta(\sigma') \cdot q(t-1,\sigma').$$

Now we are ready to revisit our definitions of reliability and responsiveness refinement for probabilistic cyclic systems with nondeterminism. For such a cyclic system M, let us define column-vectors r^t_M and q^t_M with the elements $r^t_M(\sigma) = r(t,\sigma)$ and $q^t_M(\sigma) = q(t,\sigma)$ respectively. Now, assuming that the initial state of the system is not defined deterministically but given instead by some initial state distribution, we can formulate two propositions similar to Propositions 1 and 2 of Section 5:

Proposition 3. *Let us assume that the initial system state is defined according to some probability distribution. Then the reliability refinement condition (13) is equivalent to*

$$\forall t \in \mathbb{N}_1 \cdot [\Delta^a_0] \cdot r^t_{M_a} \ \leq \ [\Delta^c_0] \cdot r^t_{M_c}$$

where $[\Delta^a_0]$ and $[\Delta^c_0]$ are the initial state distribution row-vectors for the systems M_a and M_c respectively.

Proof. Directly follows from our definition of r^t_M. ∎

Proposition 4. *Let us assume that the initial system state is defined according to some probability distribution. Then the responsiveness refinement condition (14) is equivalent to*

$$\forall t \in \mathbb{N}_1 \cdot [\Delta^a_0] \cdot q^t_{M_a} \ \leq \ [\Delta^c_0] \cdot q^t_{M_c}$$

where $[\Delta^a_0]$ and $[\Delta^c_0]$ are the initial state distribution row-vectors for the systems M_a and M_c respectively.

Proof. Directly follows from our definition of q^t_M. ∎

In Section 5 we considered the machine initialisation is to be either deterministic or probabilistic. However, for the systems that contain both probabilistic and nondeterministic behaviour, we can also assume that we do not have precise information about the system initial state, i.e., the initialisation action is of the following form: $\sigma :\in S$, where $S \subseteq \Sigma_{op}$. In this case we can formulate the following two propositions:

Proposition 5. *Let us assume that the initial system state is defined nondeterministically. Then the reliability refinement condition (13) is equivalent to*

$$\forall t \in \mathbb{N}_1 \cdot \min_{\sigma \in S_a} r(t, \sigma) \ \leq \ \min_{\sigma \in S_c} r(t, \sigma)$$

where S_a and S_c are the sets of possible initial states for the systems M_a and M_c respectively.

Proof. Directly follows from our definition of $r(t, \sigma)$ and properties of the demonic nondeterminism. ■

Proposition 6. *Let us assume that the initial system state is defined nondeterministically. Then the responsiveness refinement condition (14) is equivalent to*

$$\forall t \in \mathbb{N}_1 \cdot \min_{\sigma \in S_a} q(t, \sigma) \ \leq \ \min_{\sigma \in S_c} q(t, \sigma)$$

where S_a and S_c are the sets of possible initial states for the systems M_a and M_c respectively.

Proof. Directly follows from our definition of $q(t, \sigma)$ and properties of the demonic nondeterminism. ■

We can also easily show that the proposed approach for modelling of probabilistic systems with nondeterminism is a generalisation of the approach presented in the previous section. Indeed, let us assume that we do not have any nondeterministic choices in our system. This means that, for any state σ, the set $\overline{\Delta}_\sigma$ is a singleton set, i.e., $\forall \sigma \in \Sigma_{op} \cdot \overline{\Delta}_s = \{\Delta_\sigma\}$. Then

$$r(t, \sigma) = \min_{\Delta \in \overline{\Delta}_\sigma} \sum_{\sigma' \in \Sigma_{op}} \Delta(\sigma') \cdot r(t-1, \sigma') = \sum_{\sigma' \in \Sigma_{op}} \Delta_\sigma(\sigma') \cdot r(t-1, \sigma') = R(t)$$

and

$$q(t,\sigma) = \min_{\Delta \in \overline{\Delta}_\sigma} \sum_{\sigma' \in \Sigma_{obs}} \Delta(\sigma') \cdot q(t-1,\sigma') = \sum_{\sigma' \in \Sigma_{obs}} \Delta_\sigma(\sigma') \cdot q(t-1,\sigma') = Q(t).$$

Moreover, for fully probabilistic systems, we can easily prove (by induction) that $\forall t \in \mathbb{N}, \sigma \in \Sigma_{op} \cdot r(t,\sigma) + q(t,\sigma) = 1$.

To demonstrate the use of our extended definitions of quantitative refinement, let us revisit our simple example yet again. Figure 4 shows a model that refines the cyclic system CS, yet combines both nondeterministic and probabilistic behaviour. Here we refine the abstract event e_0 by two events e_{01} and e_{02} that probabilistically assign one of the values OK and NOK to the variable res according to two different probability distributions. However, the choice between these distributions is nondeterministic. This is achieved by nondeterministically assigning to the new variable dst any of the values 1 or 2 in the model initialisation, and later repeating this action in the bodies of events e_{01} and e_{02}.

The model is rather simple. Hence the worst case scenario reliability function $r(t,\sigma)$ is trivially defined as

$$r(t,\sigma) = (\min\{p_1, p_2\})^t$$

for any of two possible initial states σ. More precisely, after the initialisation the "demon" always chooses the distribution that leads to $res = NOK$ with the higher probability. It follows the same strategy while choosing a distribution for the next iteration in the body of the triggered event (e_{01} or e_{02}).

Let us now assume that one wants to refine the machine $NPCS$ by only changing the probabilities in the bodies of events e_{01} and e_{02}, i.e., by replacing the probabilities p_1 and p_2 with some new values s_1 and s_2. Then, according to Definition 7, such a new model refines $NPCS$ if and only if

$$\min\{p_1, p_2\} \leq \min\{s_1, s_2\}.$$

Essentially, it means that the "better" distribution, i.e., the distribution that leads to $res = OK$ with the higher probability, does not play any role in verification of the refinement condition (13). This observation gives some intuition about the demonic nature of nondeterminism in Event-B.

Machine $NPCS$
Variables $active$, res, dst
Invariants $dst \in 1..2$
Events

$Init \mathrel{\widehat{=}}$
begin
$active, res := FALSE, ND$
$dst :\in 1..2$
end

$IN \mathrel{\widehat{=}}$
when
$active = FALSE \wedge res = ND$
then
$active := TRUE$
end

$OUT \mathrel{\widehat{=}}$
when
$active = TRUE \wedge res \neq ND$
then
$active := FALSE$
$res : |res' \in \{res, ND\} \wedge$
$res' = ND \Leftrightarrow res = OK$
end

$e_{01} \mathrel{\widehat{=}}$
when
$active = TRUE \wedge res = ND$
$dst = 1$
then
$res \oplus| \ OK \ @ \ p_1; NOK \ @ \ 1{-}p_1$
$dst :\in 1..2$
end

$e_{02} \mathrel{\widehat{=}}$
when
$active = TRUE \wedge res = ND$
$dst = 2$
then
$res \oplus| \ OK \ @ \ p_2; NOK \ @ \ 1{-}p_2$
$dst :\in 1..2$
end

Figure 4: Cyclic system: combining probabilities with nondeterminism

7. Discussion

In this paper we have defined a formal basis for integrating stochastic reasoning about reliability and responsiveness into formal modelling of cyclic systems in Event-B. Since reliability and responsiveness are functions of time, we first had to address the problem of representing time in Event-B. Instead of modelling time explicitly, i.e., as an intrinsic part of Event-B model, we have decided to reason about time implicitly by relying on the notion of system iteration. This decision was motivated by several reasons. Firstly, the current approaches to modelling time in Event-B are still rather premature. For instance, the approach proposed by Butler et al. [20] significantly complicates the model and hence raises the question of scalability. The approach by Iliasov et al. [21] relies on building an additional model view (called process view) in a dedicated modelling language and, hence, makes it hard to relate probabilistic Event-B and the process view models. Moreover, there

is currently no automatic tool support for verification of timed systems in Event-B. As a result, we have decided to narrow down the class of modelled systems to cyclic systems and rely on the notion of iteration as a discrete unit of time defining the system time scale. We have formally defined the verification conditions ensuring that the abstract as well as all subsequent refined models preserve the cyclic system behaviour.

To enable stochastic reasoning in Event-B, we have introduced the quantitative probabilistic choice operator into the modelling language and also defined the quantitative refinement conditions that a refined model should satisfy. These conditions ensure that responsiveness and reliability are improved (or at least preserved) in the refinement process, as postulated in Definitions 5 – 8. The resulting integration of stochastic reasoning about non-functional system properties with the modelling and verification of the system functionality facilitates the development process. Indeed, it supports early assessment of the chosen design decisions and possible design alternatives with respect to the non-functional critical properties. Our approach to verification of quantitative model refinement is based on time-bounded reachability analysis of the underlying Markov processes.

In our work, we aimed at finding a scalable solution to integrating probabilistic reasoning into the refinement process. To ensure scalability, we need to address two main issues – applicability of the theory to modelling large-scale systems and availability of automatic tool support. To handle complexity posed by the size of the state space of large-scale systems, we can employ such techniques as lumping and probabilistic bisimulation (see e.g., [9, 22] for fully probabilistic systems and [23, 24] for nondeterministic probabilistic systems). To address the second issue, we can rely on the Rodin Platform [12] – an open extendable environment for modelling and verification in Event-B. Openness and extendability of the platform allow us to build a tool support – a dedicated plug-in that would facilitate the calculations presented in Section 5 and Section 6. The theoretical research described in this paper can be seen as a basis for achieving this goal.

8. Related work

Hallerstede and Hoang [15] have proposed an extension of the Event-B framework to model probabilistic system behaviour. Specifically, they have introduced the qualitative probabilistic choice operator to reason about *almost certain termination*. This operator is used to bound demonic nondeter-

minism in order to prove convergence of new events in Event-B models. In particular, this technique is applied to demonstrate convergence of a certain communication protocol. However, the presented approach is not suitable for quantitative evaluation of system properties since the introduced operator does not contain explicit probabilistic information.

A similar topic has been explored by Morgan et al. [25, 16] in the context of refinement calculus. In this approach the notion of probabilistic (data) refinement is defined and used to assess system dependability. The semantics of statements is given in the form of *expectation transformers*. Dependability is assessed within the data refinement process. Probabilistic data refinement is verified by simulation of datatypes. However, establishing simulation that implies data refinement between datatypes is an extremely difficult problem, which raises the scalability issue. Moreover, the majority of non-functional system attributes explicitly depend on time, yet, to the best of our knowledge, the notion of time is not defined in the probabilistic refinement calculus.

Probabilistic model checking is widely used for assessment of non-functional system requirements. There are a number of works, for instance, see [26, 27, 28] successfully applying the quantitative model checking technique to evaluate system dependability and performance. These approaches benefit from a good tool support for formal modelling and verification of discrete- and continuous-time Markov processes [29, 30]. The principal difference between model checking and our approach stems from the fact that the model checking generally aims at assessing non-functional system attributes of already developed systems. However, postponing the dependability and responsiveness evaluation to the later development stages can lead to major system redevelopment if non-functional requirement are not met. In our approach, assessment of non-functional requirements proceeds hand-in-hand with the system development by refinement, which allows us to analyse the behaviour of a designed system on the early stages of development. Despite the discussed differences, we see probabilistic model checking techniques as complementing to our approach. More specifically, quantitative model checkers can be used in conjunction with Rodin to prove the strengthened refinement of Event-B models according to Definitions 5 – 8.

A connection between probabilistic reasoning and program refinement has been investigated by Meinicke and Solin [31]. The authors introduce a refinement algebra for reasoning about probabilistic program transformations. In particular, they investigate the data and atomicity refinement rules for probabilistic programs and explore the difference between probabilistic

and non-probabilistic programs. They reason about probabilistic program transformations without introducing a probabilistic choice operator or other explicit probabilistic attributes. Our approach is rather different to the one by Meinicke and Solin. We introduce the quantitative probabilistic choice operator, which explicitly defines concrete probabilistic values for different choices. The introduced probabilistic information is used to verify quantitative non-functional properties of the system and their preservation by refinement. Otherwise, we rely on the existing Event-B refinement framework to guarantee correctness of model transformations.

9. Conclusions

In this paper we have proposed an approach to integrating stochastic reasoning about reliability and responsiveness of cyclic systems into Event-B modelling. We have made a number of technical contributions. Firstly, we have formally defined the conditions that should be verified to ensure that the system under construction has a cyclic behaviour. Secondly, we have proposed an extension of the Event-B language with the quantitative probabilistic choice construct and defined the proof semantics for the extended framework. Finally, we have demonstrated how to define reliability and responsiveness as the properties of extended Event-B models and integrate explicit stochastic reasoning about non-functional system properties into the Event-B refinement process.

The main novelty of our work is in establishing theoretical foundations for reasoning about probabilistic properties of augmented Event-B models. This result has been achieved by constraining the structure of considered Event-B models and consequently reducing the reasoning about time-dependent properties in general to the reasoning about these properties in terms of iterations. Since the cyclic systems constitute a large class of critical systems, we believe that the imposed restriction does not put significant limitations on the applicability of the proposed approach. Yet it allows us to represent the system models as discrete Markov chains or Markov decision processes. This, in its turn, enables the use of the well-established theory of Markov processes to verify time-bounded reachability properties.

References

[1] J.-R. Abrial, Modeling in Event-B, Cambridge University Press, 2010.

[2] J.-R. Abrial, Extending B without Changing it (for Developing Distributed Systems), in: H. Habiras (Ed.), First Conference on the B method, 1996, pp. 169–190.

[3] J.-R. Abrial, The B-Book: Assigning Programs to Meanings, Cambridge University Press, 2005.

[4] A. Villemeur, Reliability, Availability, Maintainability and Safety Assessment, John Wiley & Sons, 1995.

[5] P. D. T. O'Connor, Practical Reliability Engineering, 3rd ed, John Wiley & Sons, 1995.

[6] A. Avizienis, J.-C. Laprie, B. Randell, Fundamental Concepts of Dependability, 2001, Research Report No 1145, LAAS-CNRS.

[7] K. Trivedi, S. Ramani, R. Fricks, Recent Advances in Modeling Response-Time Distributions in Real-Time Systems, in: Proceedings of the IEEE, Vol. 91(7), 2003, pp. 1023–1037.

[8] W. W. Chu, C.-M. Sit, Estimating Task Response Time with Contentions for Real-Time Distributed Systems, in: Real-Time Systems Symposium, 1988, pp. 272–281.

[9] J. G. Kemeny, J. L. Snell, Finite Markov Chains, D. Van Nostrand Company, 1960.

[10] M. Putterman, Markov Decision Processes. Discrete Stochastic Dynamic Programming, John Wiley & Sons, 2005.

[11] D. J. White, Markov Decision Processes, John Wiley & Sons, 1993.

[12] Rodin, Event-B Platform, online at http://www.event-b.org/.

[13] A. Iliasov, Use Case Scenarios as Verification Conditions: Event-B/Flow Approach, in: SERENE 2011, Software Engineering for Resilient Systems, Springer-Verlag, 2011, pp. 9–23.

[14] A. Tarasyuk, E. Troubitsyna, L. Laibinis, Formal Modelling and Verification of Service-Oriented Systems in Probabilistic Event-B, in: IFM 2012, Integrated Formal Methods, Vol. 7321 of LNCS, Springer, 2012, pp. 237–252.

[15] S. Hallerstede, T. S. Hoang, Qualitative probabilistic modelling in Event-B, in: J. Davies, J. Gibbons (Eds.), IFM 2007, Integrated Formal Methods, 2007, pp. 293–312.

[16] A. K. McIver, C. C. Morgan, Abstraction, Refinement and Proof for Probabilistic Systems, Springer, 2005.

[17] A. Tarasyuk, E. Troubitsyna, L. Laibinis, Towards Probabilistic Modelling in Event-B, in: D. Méry, S. Merz (Eds.), IFM 2010, Integrated Formal Methods, Springer-Verlag, 2010.

[18] R. J. R. Back, J. von Wright, Refinement Calculus: A Systematic Introduction, Springer-Verlag, 1998.

[19] K. Hinderer, T.-H. Waldmann, The Critical Discount Factor for Finite Markovian Decision Processes with an Absorbing Set, in: Mathematical Methods fo Operations Research, Springer-Verlag, 2003, pp. 1–19.

[20] M. Butler, J. Falampin, An approach to modelling and refining timing properties in B, in: Refinement of Critical Systems (RCS), 2002.

[21] Augmenting event-b modelling with real-time verification.

[22] K. G. Larsen, A. Skou, Bisimulation through Probabilistic Testing, in: Information and Computation 94, 1991, pp. 1–28.

[23] H. Hansson, Time and Probability in Formal Design of Distributed Systems, Elsevier, 1995.

[24] R. Segala, N. Lynch, Probabilistic simulations for probabilistic processes, in: Nordic Journal of Computing, 2(2), 1995, pp. 250–273.

[25] A. K. McIver, C. C. Morgan, E. Troubitsyna, The Probabilistic Steam Boiler: a Case Study in Probabilistic Data Refinement, in: International Refinement Workshop, ANU, Canberra, Springer, 1998.

[26] M. Kwiatkowska, G. Norman, D. Parker, Controller Dependability Analysis by Probabilistic Model Checking, in: Control Engineering Practice, 2007, pp. 1427–1434.

[27] C. Baier, B. Haverkort, H. Hermanns, J.-P. Katoen, Automated Performance and Dependability Evaluation Using Model Checking, in: Performance Evaluation of Complex Systems: Techniques and Tools, Springer-Verlag, 2002, pp. 261–289.

[28] M. Massink, J.-P. Katoen, D. Latella, Model Checking Dependability Attributes of Wireless Group Communication, in: DSN'04, International Conference on Dependable Systems and Networks, 2004, pp. 711–720.

[29] PRISM, Probabilistic symbolic model checker, online at http://www.prismmodelchecker.org/.

[30] MRMC, Markov Reward Model Checker, online at http://www.mrmc-tool.org/.

[31] L. Meinicke, K. Solin, Refinement algebra for probabilistic programs, in: Formal Aspects of Computing, Vol. 22, 2010, pp. 3–31.

Paper II

Towards Probabilistic Modelling in Event-B

Anton Tarasyuk, Elena Troubitsyna and Linas Laibinis

Originally published in: Dominique Méry, Stephan Merz (Eds.), *Proceedings of 8th International Conference on Integrated Formal Methods (iFM 2010)*, LNCS 6396, 275–289, Springer, 2010

Towards Probabilistic Modelling in Event-B

Anton Tarasyuk[1,2], Elena Troubitsyna[2], and Linas Laibinis[2]

[1] Turku Centre for Computer Science
[2] Åbo Akademi University
Joukahaisenkatu 3-5 A, 20520 Turku, Finland
{anton.tarasyuk,elena.troubitsyna,linas.laibinis}@abo.fi

Abstract. Event-B provides us with a powerful framework for correct-by-construction system development. However, while developing dependable systems we should not only guarantee their functional correctness but also quantitatively assess their dependability attributes. In this paper we investigate how to conduct probabilistic assessment of reliability of control systems modeled in Event-B. We show how to transform an Event-B model into a Markov model amendable for probabilistic reliability analysis. Our approach enables integration of reasoning about correctness with quantitative analysis of reliability.

Keywords: Event-B, cyclic system, refinement, probability, reliability.

1 Introduction

System development by refinement is a formalised model-driven approach to developing complex systems. Refinement enables correct-by-construction development of systems. Its top-down development paradigm allows us to cope with system complexity via abstraction, gradual model transformation and proofs. Currently the use of refinement is mainly limited to reasoning about functional correctness. Meanwhile, in the area of dependable system development – the area where the formal modelling is mostly demanded – besides functional correctness it is equally important to demonstrate that the system adheres to certain quantitatively expressed dependability level. Hence, there is a clear need for enhancing formal modelling with a capability of stochastic reasoning about dependability.

In this paper we propose an approach to introducing probabilities into Event-B modelling [1]. Our aim is to enable quantitative assessment of dependability attributes, in particular, reliability of systems modelled in Event-B. We consider cyclic systems and show that their behaviour can be represented via a common Event-B modelling pattern. We show then how to augment such models with probabilities (using a proposed probabilistic choice operator) that in turn would allow us to assess their reliability.

Reliability is a probability of system to function correctly over a given period of time under a given set of operating conditions [23,24,17]. It is often assessed using the classical Markov modelling techniques [9]. We demonstrate that Event-B models augmented with probabilities can be given the semantic of a Markov

D. Méry and S. Merz (Eds.): IFM 2010, LNCS 6396, pp. 275–289, 2010.

process (or, in special cases, a Markov chain). Then refinement of augmented Event-B models essentially becomes reliability-parameterised development, i.e., the development that not only guarantees functional correctness but also ensures that reliability of refined model is preserved or improved. The proposed approach allows us to smoothly integrate quantitative dependability assessment into the formal system development.

The paper is structured as follows. In Section 2 we overview our formal framework – Event-B. In Section 3 we introduce a general pattern for specifying cyclic systems. In Section 4 we demonstrate how to augment Event-B models with probabilities to enable formal modelling and refinement of fully probabilistic systems. In Section 5 we generalise our proposal to the cyclic systems that also contain non-determinism. Finally, in Section 6 we overview the related work and give concluding remarks.

2 Introduction to Event-B

The B Method [2] is an approach for the industrial development of highly dependable software. The method has been successfully used in the development of several complex real-life applications [19,5]. Event-B is a formal framework derived from the B Method to model parallel, distributed and reactive systems. The Rodin platform [21] provides automated tool support for modelling and verification (by theorem proving) in Event-B. Currently Event-B is used in the EU project Deploy [6] to model several industrial systems from automotive, railway, space and business domains.

In Event-B a system specification is defined using the notion of an abstract state machine [20]. An abstract machine encapsulates the state (the variables) of a model and defines operations on its state. A general form of Event-B models is given in Fig.1.

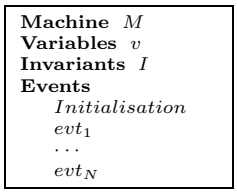


Fig. 1. An Event-B machine

The machine is uniquely identified by its name M. The state variables, v, are declared in the **Variables** clause and initialised in the *init* event. The variables are strongly typed by the constraining predicates I given in the **Invariants** clause. The invariant clause might also contain other predicates defining properties that should be preserved during system execution.

The dynamic behaviour of the system is defined by the set of atomic events specified in the **Events** clause. Generally, an event can be defined as follows:

$$evt \mathrel{\widehat{=}} \textbf{when } g \textbf{ then } S \textbf{ end},$$

where the guard g is a conjunction of predicates over the state variables v and the action S is an assignment to the state variables. If the guard g is $true$, an event can be described simply as

$$evt \mathrel{\widehat{=}} \textbf{begin } S \textbf{ end},$$

In its general form, an event can also have local variables as well as parameters. However, in this paper we use only the simple forms given above.

The occurrence of events represents the observable behaviour of the system. The guard defines the conditions under which the action can be executed, i.e., when the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution nondeterministically. If none of the events is enabled then the system deadlocks.

In general, the action of an event is a parallel composition of assignments. The assignments can be either deterministic or non-deterministic. A deterministic assignment, $x := E(x, y)$, has the standard syntax and meaning. A nondeterministic assignment is denoted either as $x :\in S$, where S is a set of values, or $x :| P(x, y, x')$, where P is a predicate relating initial values of x, y to some final value of x'. As a result of such a non-deterministic assignment, x can get any value belonging to S or according to P.

The semantics of Event-B events is defined using so called before-after (BA) predicates [20]. A before-after predicate describes a relationship between the system states before and after execution of an event, as shown in Fig.2.

Action (S)	$BA(S)$
$x := E(x, y)$	$x' = E(x, y) \ \wedge \ y' = y$
$x :\in S$	$\exists t.\ (t \in Set \wedge x' = t) \ \wedge \ y' = y$
$x :\mid P(x, y, x')$	$\exists t.\ (P(x, t, y) \wedge x' = t) \ \wedge \ y' = y$

Fig. 2. Before-after predicates

where x and y are disjoint lists (partitions) of state variables, and x', y' represent their values in the after state. A before-after predicate for Event-B events is then constructed as follows:

$$BA(evt) \quad = \quad g \wedge BA(S).$$

The formal semantics provides us with a foundation for establishing correctness of Event-B specifications. In particular, to verify correctness of a specification, we need to prove that its initialisation and all events preserve the invariant.

Event-B employs a top-down refinement-based approach to system development. Development starts from an abstract system specification that models the most essential functional requirements. While capturing more detailed requirements, each refinement step typically introduces new events and variables into the abstract specification. These new events correspond to stuttering steps that are not visible at the abstract level. By verifying correctness of refinement, we ensure that all invariant properties of (more) abstract machines are preserved. A detailed description of the formal semantics of Event-B and foundations of the verification process can be found in [20].

3 Modelling of Cyclic Systems in Event-B

In this paper, we focus on modelling systems with cyclic behaviour, i.e. the systems that iteratively execute a predefined sequence of steps. Typical representatives of such cyclic systems are control and monitoring systems. An iteration of a control system includes reading the sensors that monitor the controlled physical processes, processing the obtained sensor values and setting actuators according to a predefined control algorithm. In principle, the system could operate in this way indefinitely long. However, different failures may affect the normal system functioning and lead to a shutdown. Hence, during each iteration the system status should be re-evaluated to decide whether it can continue its operation.

In general, *operational* states of a system, i.e., the states where system functions properly, are defined by some predicate $J(v)$ over the system variables. Usually, essential properties of the system (such as safety, fault tolerance, liveness properties) can be guaranteed only while system stays in the operational states. The predicate $J(v)$ partitions the system state space S into two disjoint classes of states – *operational* (S_{op}) and *non-operational* (S_{nop}) states, where $S_{op} \mathrel{\widehat{=}} \{ s \in S \mid J.s \}$ and $S_{nop} \mathrel{\widehat{=}} S \setminus S_{op}$.

Abstractly, we can specify a cyclic system in Event-B as shown in Fig.3. In the machine CS, the variable st abstractly models the system state, which can be either operational ($J(st)$ is true) or failed ($J(st)$ is false). The event *iter* abstractly models one iteration of the system execution. As a result of this event, the system can stay operational or fail. In the first case, the system can execute its next iteration. In the latter case, the system deadlocks.

Machine CS
Variables st
Invariants
  $st \in STATE$
  ...
Events
  $Initialisation \mathrel{\widehat{=}}$
    begin
      $st :\mid J(st')$
    end
  $iter \mathrel{\widehat{=}}$
    when
      $J(st)$
    then
      $st :\in STATE$
    end

Fig. 3. A cyclic system

The **Invariants** clause (besides defining the variable types) can contain other essential properties of the system. Usually they are stated only over the operational states, i.e., they are of the form:

$$J(st) \quad \Rightarrow \quad ...$$

We can refine the abstract specification CS by introducing specific implementation details. For example, we may explicitly introduce new events modelling

the environment as well as reading the sensors or setting the actuators. The event *iter* can be also refined, e.g., into *detection* operation, which decides whether the system can continue its normal operation or has to shut down due to some unrecoverable failure. However, the Event-B refinement process will preserve the cyclic nature of the system described in the abstract specification CS.

The only other constraint we put on the refinement process is that all the new events introduced in refined models can be only enabled in operational system states, e.g., the event guards should contain the condition $J(v)$. To enforce this constraint, we propose a simple syntactic extension of the Event-B model structure. Specifically, we introduce a new clause **Operational guards** containing state predicates precisely defining the subset of operational system states. This is a shorthand notation implicitly adding the corresponding guard conditions to all events enabled in the operational states (except initialisation). We also assume that, like model invariants, operational guards are inherited in all refined models. By using this new clause, we can rewrite the system CS as follows.

Machine CS
Variables st
Invariants
$st \in STATE$
...
Operational guards
$J(st)$
Events
$Initialisation \mathrel{\widehat{=}}$
begin
$st :| J(st')$
end
$iter \mathrel{\widehat{=}}$
begin
$st :\in STATE$
end

Fig. 4. A cyclic system

In general, the behaviour of some cyclic system M can be intuitively described by the sequential composition $(Initialisation; \mathbf{do}\, J \to E\, \mathbf{do})$, where $\mathbf{do}\, J \to E\, \mathbf{do}$ is a while-loop with the operational guard J and the body E that consists of all the machine events except initialisation. For example, the behaviour of CS can be described simply as $(Initialisation; \mathbf{do}\, J \to iter\, \mathbf{do})$.

Each iteration of the loop maps the current operational system state into a subset of S. The resulting set of states represents all possible states that can be reached due to system nondeterministic behaviour. Therefore, an iteration of a cyclic system M can be defined as a partial function $\mathscr{I}_M$ of the type $S_{op} \to \mathscr{P}(S)$.[1] The concrete definition of $\mathscr{I}_M$ can be derived from the composition of before-after predicates of the involved events. Moreover, we can also consider the behaviour of the overall system and observe that the final state of every iteration defines the initial state of the next iteration provided the system has not failed.

The specification pattern for modelling cyclic systems defined above restricts the shape of Event-B models. This restriction allow us to propose a scalable

[1] Equivalently, we can define an iteration as a relation between S_{op} and S.

approach to integrating probabilistic analysis of dependability into Event-B. This approach we present next.

4 Stochastic Modelling in Event-B

4.1 Introducing Probabilistic Choice

Hallerstede and Hoang [7] have extended the Event-B framework with a new operator – *qualitative probabilistic choice*, denoted $\oplus|$. This operator assigns new values to variables with some positive but generally unknown probability. The extension aimed at introducing into Event-B the concept of "almost-certain convergence"– probabilistically certain termination of new event operations introduced by model refinement. The new operator can replace a nondeterministic choice (assignment) statement in the event actions. It has been shown that any probabilistic choice statement always refines its demonic nondeterministic counterpart [13]. Hence such an extension is not interfering with traditional refinement process.

In this paper we aim at introducing *quantitative* probabilistic choice, i.e., the operator $\oplus|$ with precise probabilistic information about how likely a particular choice should be made. In other words, it behaves according to some known probabilistic distribution. The quantitative probabilistic assignment

$$x \oplus| \; x_1 \; @ \; p_1; \ldots; x_n \; @ \; p_n,$$

where $\sum_{i=1}^{n} p_i = 1$, assigns to the variable x a new value x_i with the corresponding non-zero probability p_i. Similarly to Hallerstede and Hoang, we can introduce probabilistic choice only to replace the existing demonic one.

To illustrate the proposed extension, in Fig.5 we present a small example of a probabilistic communication protocol implementing transmission over unreliable channel. Since the channel is unreliable, sent messages may be lost. In the model AM shown on the left-hand side, the occurrence of faults is modelled nondeterministically. Specifically, the variable msg_a is nondeterministically assigned *delivered* or *lost*. In the model AM', the nondeterministic choice is replaced by the probabilistic one, where the non-zero constant probabilities p and $1-p$ express how likely a message is getting delivered or lost. According to the theory of probabilistic refinement [13], the machine AM' is a refinement of the machine AM. The model refinement relation is denoted $\sqsubseteq$.

Next we show how to define refinement between probabilistic systems modelled in (extended) Event-B. In particular, our notion of model refinement can be specialized to quantitatively demonstrate that the refined system is at least as reliable as its more abstract counterpart.

4.2 Fully Probabilistic Systems

Let us first consider fully probabilistic systems, i.e., systems containing only probabilistic nondeterminism. The quantitative information present in a

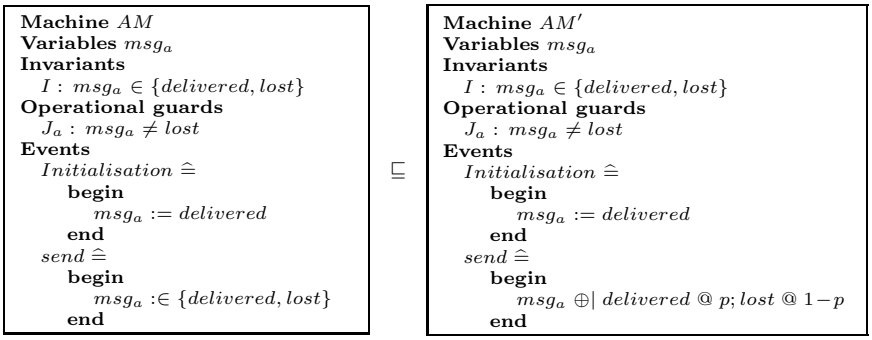


Fig. 5. A simple communication protocol: introducing probabilities

probabilistic Event-B model requires lifting the notion of the system state to a probabilistic distribution over it:

Definition 1 (Probabilistic distribution). *For a system state space S, the set of distributions over S is*

$$\bar{S} \quad \widehat{=} \quad \{\Delta : S \to [0,1] \mid \sum_{s \in S} \Delta.s \; = \; 1\},$$

where $\Delta.s$ is the probability of reaching the state s.

Each iteration of a fully probabilistic system then maps some initial operational state to a subset of S according to some probabilistic distribution, i.e., we can define a single iteration $\mathscr{PI}_M$ of a probabilistic cyclic system M as a partial function of the type $S_{op} \to \bar{S}$.

There is a simple connection between iteration $\mathscr{I}_M$ of a cyclic system M and and its probabilistic counterpart $\mathscr{PI}_M$ – some state is reachable by $\mathscr{I}_M$ if and only it is reachable by $\mathscr{PI}_M$ with a non-zero probability:

$$\forall s,\, s' \cdot \; s \in \mathbf{dom}.\mathscr{I}_M \; \wedge \; s' \in \mathscr{I}_M.s \; \Leftrightarrow \; s \in \mathbf{dom}.\mathscr{PI}_M \; \wedge \; \mathscr{PI}_M.s.s' > 0,$$

where **dom** is the function domain operator.

For example, it is straightforward to see that for our model AM of the communication channel, the iteration function $\mathscr{I}_{AM}$ is

$$\mathscr{I}_{AM} = \{delivered \mapsto \{delivered, lost\}\},$$

while the probabilistic iteration function $\mathscr{PI}_{AM'}$ for the model AM' is

$$\mathscr{PI}_{AM'} = \{delivered \mapsto \{delivered \mapsto p, lost \mapsto (1-p)\}\}.$$

As it was mentioned before, all elements of system state are partitioned into two disjoint classes of operational and non-operational states. For any state $s \in S_{op}$, its distribution Δ is defined by probabilistic choice statements (assignments)

presented in an Event-B machine. However, once the system fails, it stays in the failed (non-operational) state. This means that, for any state $s \in S_{nop}$, its distribution Δ is such that $\Delta.s = 1$ and $\Delta.s' = 0$, if $s' \neq s$.

Once we know the probabilistic state distribution Δ, we can quantitatively assess the probability that the operational guard J is preserved by a single iteration. However, our goal is to evaluate system reliability. In engineering, reliability [24,17] is generally measured by the probability that an entity $\mathscr{E}$ can perform a required function under given conditions for the time interval $[0, t]$:

$$R(t) = \mathbf{P}\{\mathscr{E} \textit{ not failed over time } [0, t]\}.$$

Hence reliability can be expressed as the probability that J remains *true* during a certain number of iterations, i.e., the probability of system staying operational for k iterations:

$$R(t) = \mathbf{P}\{\Box^{\leq k} J\}.$$

Here we use the modal operator $\Box$ borrowed from temporal logic (LTL or (P)CTL, for instance). The formula $(\Box^{\leq k} J)$ means that J holds *globally* for the first k iterations. It is straightforward to see that this property corresponds to the standard definition of reliability given above.

Let M and M' be probabilistic Event-B models of cyclic systems. We strengthen the notion of Event-B refinement by additionally requiring that the refined model will execute more iterations before shutdown with a higher probability:

Definition 2 (Refinement for probabilistic cyclic systems)
For two probabilistic Event-B models M and M' of cyclic systems such that $M \mathrel{\widehat{=}} (Initialisation; \mathbf{do}\, J \to E\, \mathbf{od})$ and $M' \mathrel{\widehat{=}} (Initialisation'; \mathbf{do}\, J' \to E'\, \mathbf{od})$, we say that M' is a refinement of M, if and only if

1. *M' is an Event-B refinement of M ($M \sqsubseteq M'$), and*
2. $\forall k \in \mathbb{N}_1 \cdot \mathbf{P}\{\Box^{\leq k} J\} \leq \mathbf{P}\{\Box^{\leq k} J'\}$.

Remark 1. If the second condition of Definition 2 holds not for all k, but for some interval $k \in 1..K$, $K \in \mathbb{N}_1$, we say that M' is a *partial* refinement of M for $k \leq K$.

From the reliability point of view, a comparison of probabilistic distributions corresponds to a comparison of how likely the system would fail in its next iteration. This consideration allows us to define an order over the set $\bar{S}$ of system distributions:

Definition 3 (Ordering over distributions). *For two distributions $\Delta, \Delta' \in \bar{S}$ we define the ordering relation $\preceq$ as follows*

$$\Delta \preceq \Delta' \quad \Leftrightarrow \quad \sum_{s \in S_{op}} \Delta.s \leq \sum_{s \in S_{op}} \Delta'.s.$$

It is easy to see that the ordering relation $\preceq$ defined in this way is reflexive and transitive and hence is a total preorder on S. Let us note that the defined order is not based on pointwise comparison between the corresponding single state probabilities. Instead, we rely on the accumulated likelihood that the system stays operational.

McIver and Morgan [13] have considered deterministic probabilistic programs with possible nontermination. They have defined the set of (sub-)distributions for terminating programs, with the order over distributions introduced as $\Delta \preceq \Delta' \Leftrightarrow (\forall s \in S \cdot \Delta.s \leq \Delta'.s)$. Such a pointwise definition of an order is too strong for our purposes. We focus on quantitative evaluation of system reliability, treating all the operational states in system distributions as one class, i.e., we do not distinguish between single operational states or their groups. In our future work it would be interesting to consider a more fine-grained classification of operational states, e.g., taking into account different classes of degraded states of the system.

The order over final state distributions can be in turn used to define the order over the associated initial states:

Definition 4 (Ordering over states). *Let M be a probabilistic cyclic system. Then, for its iteration $\mathscr{PI}_M$, any initial states $s_i, s_j \in S_{op}$ and distributions $\Delta_i, \Delta_j \in \bar{S}$ such that $\Delta_i = \mathscr{PI}_M.s_i$ and $\Delta_j = \mathscr{PI}_M.s_j$, we define the ordering relation $\preceq_M$ as*

$$s_i \preceq_M s_j \quad \Leftrightarrow \quad \Delta_i \preceq \Delta_j$$

We can use this state ordering to represent the system state space S as an ordered set $\{s_1, \ldots, s_n\}$, where $n \in \mathbb{N}_{\geq 2}$ and $(\forall i \in 1..(n-1) \cdot \Delta_{i+1} \preceq \Delta_i)$.

Generally, all the non-operational states S_{nop} can be treated as a singleton set, since we do not usually care at which particular state the operational guard has been violated. Therefore, by assuming that $S = \{s_1, \ldots, s_n\}$ and $S_{nop} = \{s_n\}$, it can be easily shown that s_n is the least element (bottom) of S:

$$\Delta_n.s_n = 1 \Rightarrow \forall i \in 1..n \cdot s_n \preceq_M s_i$$

Now let us consider the behaviour of some cyclic system M in detail. We can assume that the initial system state s_1 belongs to the ordered set $\{s_1, \ldots, s_n\}$. This is a state where the system works "perfectly". After its first iteration, the system goes to some state s_i with the probability $\Delta_1.s_i$ and s_i becomes the current system state. At this point, if $i = n$, system shutdown is initiated. Otherwise, the system starts a new iteration and, as a result, goes to some state s_j with the probability $\Delta_i.s_j$ and so on. It is easy to see that this process is completely defined by the following state transition matrix

$$P_M = \begin{pmatrix} \Delta_1.s_1 & \Delta_1.s_2 & \ldots & \Delta_1.s_n \\ \Delta_2.s_1 & \Delta_2.s_2 & \ldots & \Delta_2.s_n \\ \vdots & \vdots & \ddots & \vdots \\ \Delta_n.s_1 & \Delta_n.s_2 & \ldots & \Delta_n.s_n \end{pmatrix},$$

which in turn unambiguously defines the underlying Markov process (absorbing discrete time Markov chain, to be precise).

Let us note that the state transition matrix of a Markov chain and its initial state allow us to calculate the probability that the defined Markov process (after k steps) will be in a state s_i (see [9] for example). Let assume that the operational states of the system are ordered according to Definition 4 and initially a system is in the state s_1. Then we can rewrite the second condition of Definition 2 in the following way:

Proposition 1. *For two probabilistic Event-B models M and M' such that $M \mathrel{\widehat{=}} (Initialisation; \mathbf{do}\, J \to E\, \mathbf{do})$ and $M' \mathrel{\widehat{=}} (Initialisation'; \mathbf{do}\, J' \to E'\, \mathbf{do})$, the inequality*

$$\forall k \in \mathbb{N}_1 \cdot \mathbf{P}\{\Box^{\leq k} J\} \leq \mathbf{P}\{\Box^{\leq k} J'\}$$

is equivalent to

$$\forall k \in \mathbb{N}_1 \cdot ((P_{M'})^k)_{1n'} \leq ((P_M)^k)_{1n}, \tag{1}$$

where $S = \{s_1, \ldots, s_n\}$ and $S' = \{s_1, \ldots, s_{n'}\}$ are the ordered system state spaces of M and M' accordingly, and $(\ldots)_{1n}$ is a $(1n)$-th element of a matrix.

Proof. Directly follows from our definition of the order on state distributions and fundamental theorems of the Markov chains theory. ∎

In general, the initial system state is not necessarily the given state s_1 but can be defined by some initial state distribution Δ_0. In this case the inequality (1) should be replaced with

$$([\Delta_0'] \cdot P_{M'}^k)(n') \leq ([\Delta_0] \cdot P_M^k)(n),$$

where $[\Delta_0] = \begin{pmatrix} \Delta_0.s_1 \\ \vdots \\ \Delta_0.s_n \end{pmatrix}$, $[\Delta_0'] = \begin{pmatrix} \Delta_0'.s_1 \\ \vdots \\ \Delta_0'.s_{n'} \end{pmatrix}$ and $([\Delta_0] \cdot P_M^k)(n)$ is the n-th component of the column vector $([\Delta_0] \cdot P_M^k)$.

To illustrate our approach to refining fully probabilistic systems, let us revisit our transmission protocol example. To increase reliability of transmission, we refine the protocol to allow the sender to repeat message sending in case of delivery failure. The maximal number of such attempts is given as the predefined positive constant N. The resulting Event-B model CM is presented in Fig.6. Here the variable *att* represents the current sending attempt. Moreover, the event *send* is split to model the situations when the threshold N has been accordingly reached and not reached.

The Event-B machine CM can be proved to be a probabilistic refinement of its abstract probabilistic model (the machine AM' in Fig.5) according to Definition 2.

In this section we focused on fully probabilistic systems. In the next section we generalize our approach to the systems that also contain nondeterminism.

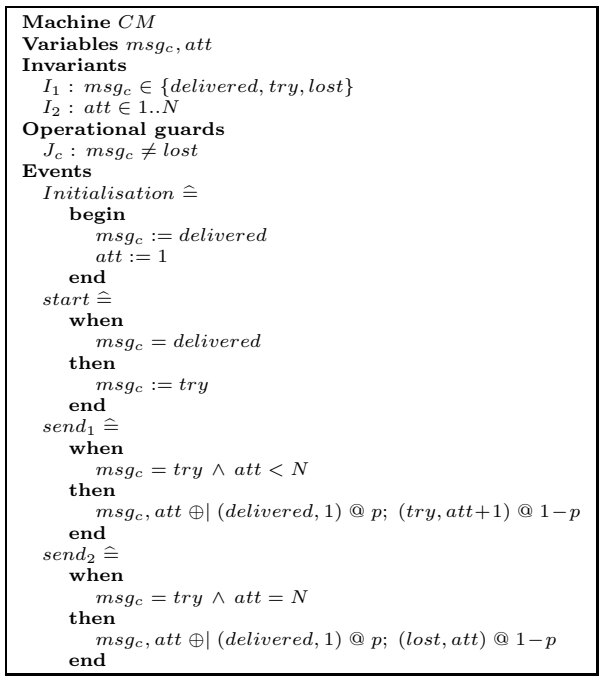


Fig. 6. A simple communication protocol: probabilistic refinement

4.3 Probabilistic Systems with Nondeterminism

For a cyclic system M containing both probabilistic and demonic nondeterminism we define a single iteration as the partial function $\mathscr{PI}_M$ of the type $S_{op} \to \mathscr{P}(\bar{S})$, i.e., as a mapping of the operational state into a *set* of distributions over S.

Nondeterminism has a demonic nature in Event-B. Hence such a model represent a worst case scenario, i.e., choosing the "worst" of operative sub-distributions – the distributions with a domain restriction on S_{op}. From reliability perspective, it means that while assessing reliability of such a system we obtain the lowest bound estimate of reliability. In this case the notions of probabilistic system refinement and the state ordering are defined as follows:

Definition 5 (Refinement for nondeterministic-probabilistic systems). *For two nondeterministic-probabilistic Event-B models M and M' of cyclic systems such that $M \mathrel{\widehat{=}} (Initialisation; \mathbf{do}\, J \to E\, \mathbf{do})$ and $M' \mathrel{\widehat{=}} (Initialisation'; \mathbf{do}\, J' \to E'\, \mathbf{do})$, we say that M' is a refinement of M, if and only if*

1. *M' is an Event-B refinement of M ($M \sqsubseteq M'$);*
2. *$\forall k \in \mathbb{N}_1 \cdot \mathbf{P}_{min}\{\Box^{\leq k} J\} \leq \mathbf{P}_{min}\{\Box^{\leq k} J'\}$,*

where $\mathbf{P}_{min}\{\Box^{\leq k} J\}$ is the minimum probability that J remains true during the first k iterations.

Remark 2. If the second refinement condition of the Definition 5 holds not for all k, but for some interval $k \in 1..K$, $K \in \mathbb{N}_1$, we say that M' is a *partial* refinement of M for $k \leq K$.

Definition 6 (Ordering over distributions)
For two sets of distributions $\{\Delta_{i_l} \mid l \in 1..L\}$ *and* $\{\Delta_{j_k} \mid k \in 1..K\} \in \mathscr{P}(\bar{S})$*, we define the ordering relation* $\preceq$ *as*

$$\{\Delta_{i_l} \mid l \in 1..L\} \preceq \{\Delta_{j_k} \mid k \in 1..K\} \Leftrightarrow \min_l(\sum_{s\in S} \Delta_{i_l}.s) \leq \min_k(\sum_{s\in S} \Delta_{j_k}.s).$$

As in the previous section, the order over final state distributions can be in turn used to define the order over the associated initial states:

Definition 7 (Ordering over states). *Let* M *be a nondeterministic-probabilistic system. Then, for its iteration* $\mathscr{PI}_M$*, any initial states* $s_i, s_j \in S_{op}$ *and sets of distributions* $\{\Delta_{i_l} \mid l \in 1..L\}, \{\Delta_{j_k} \mid k \in 1..K\} \in \mathscr{P}(\bar{S})$ *such that* $\{\Delta_{i_l} \mid l \in 1..L\} = \mathscr{PI}_M.s_i$ *and* $\{\Delta_{j_k} \mid k \in 1..K\} = \mathscr{PI}_M.s_j$*, we define the ordering relation* $\preceq_M$ *as*

$$s_i \preceq_M s_j \quad \Leftrightarrow \quad \{\Delta_{i_l} \mid l \in 1..L\} \preceq \{\Delta_{j_k} \mid k \in 1..K\}$$

The underlying Markov process representing the behaviour of a nondeterministic-probabilistic cyclic system is a simple form of a Markov decision process. For every $i \in 1, \ldots, (n-1)$, let us define $\underline{\Delta}_i = \min_l(\sum_{s\in S} \Delta_{i_l}.s)$ and $\underline{\Delta}_n = \Delta_n$. Then, the state transition matrix that represents the worst-case scenario system behaviour is defined in the following way:

$$\underline{P}_M = \begin{pmatrix} \underline{\Delta}_1.s_1 & \underline{\Delta}_1.s_2 & \ldots & \underline{\Delta}_1.s_n \\ \underline{\Delta}_2.s_1 & \underline{\Delta}_2.s_2 & \ldots & \underline{\Delta}_2.s_n \\ \vdots & \vdots & \ddots & \vdots \\ \underline{\Delta}_n.s_1 & \underline{\Delta}_n.s_2 & \ldots & \underline{\Delta}_n.s_n \end{pmatrix},$$

and the second refinement condition of Definition 5 can be rewritten as follows:

Proposition 2. *For two nondeterministic-probabilistic Event-B models* M *and* M' *such that* $M \mathrel{\widehat{=}} (Initialisation; \mathbf{do}\, J \rightarrow E\, \mathbf{do})$ *and* $M' \mathrel{\widehat{=}} (Initialisation'; \mathbf{do}\, J' \rightarrow E'\, \mathbf{do})$*, the inequality*

$$\forall k \in \mathbb{N} \cdot \mathbf{P}\{\Box^{\leq k} J\} \leq \mathbf{P}\{\Box^{\leq k} J'\}$$

is equivalent to

$$\forall k \in \mathbb{N}_1 \cdot ((\underline{P}_{M'})^k)_{1n'} \leq ((\underline{P}_M)^k)_{1n}, \tag{2}$$

where $S = \{s_1, \ldots, s_n\}$ *and* $S' = \{s_1, \ldots, s_{n'}\}$ *are the ordered system state spaces of* M *and* M' *accordingly.*

Proof. This proof is the same as the proof for Proposition 1. ∎

Similarly as for fully-probabilistic systems, if the initial system state is not a single state s_1, but instead it is defined by some initial state distribution Δ_0, then the inequality (2) is replaced by

$$([\Delta'_0] \cdot \underline{P}^k_{M'})(n') \leq ([\Delta_0] \cdot \underline{P}^k_M)(n).$$

4.4 Discussion

For fully probabilistic systems, we can often reduce the state space size using the lumping technique [9] or equally probabilistic bisimulation [12]. For nondeterministic probabilistic systems, a number of bisimulation techniques [8,22] have been also developed.

For simple system models, deriving the set of state distributions $\bar{S}$ and calculating reliability probabilities P^k_M for each refinement step can be done manually. However, for complex real-size systems this process can be extremely time and effort consuming. Therefore, it is beneficial to have an automatic tool support for routine calculations. Development and verification of Event-B models is supported by the Rodin Platform [19] – integrated extensible development environment for Event-B. However, at the moment the support for quantitative verification is sorely missing. To prove probabilistic refinement of Event-B models according to Definition 2 and Definition 5, we need to extend the Rodin platform with a dedicated plug-in or integrate some external tool.

One of the available automated techniques widely used for analysing systems that exhibit probabilistic behaviour is probabilistic model checking [4,10]. In particular, the probabilistic model checking frameworks like PRISM or MRMC [18,16] provide good tool support for formal modelling and verification of discrete- and continuous-time Markov processes. To enable the quantitative reliability analysis of Event-B models, it would be advantageous to develop a Rodin plug-in enabling automatic translation of Event-B models to existing probabilistic model checking frameworks.

5 Related Work and Conclusions

5.1 Related Work

The Event-B framework has been extended by Hallerstede and Hoang [7] to take into account model probabilistic behaviour. They introduce qualitative probabilistic choice operator to reason about almost certain termination. This operator attempts to bound demonic nondeterminism that, for instance, allows us to demonstrate convergence of certain protocols. However, this approach is not suitable for reliability assessment since explicit quantitative representation of reliability is not supported.

Several researches have already used quantitative model checking for dependability evaluation. For instance, Kwiatkowska et al. [11] have proposed an

approach to assessing dependability of control systems using continuous time Markov chains. The general idea is similar to ours – to formulate reliability as a system property to be verified. This approach differs from ours because it is aims at assessing reliability of already developed systems. However, dependability evaluation late at the development cycle can be perilous and, in case of poor results, may lead to major system redevelopment causing significant financial and time losses. In our approach reliability assessment proceeds hand-in-hand with the system development by refinement. It allows us to assess dependability of designed system on the early stages of development, for instance, every time when we need to estimate impact of unreliable component on the system reliability level. This allows a developer to make an informed decision about how to guarantee a desired system reliability.

A similar topic in the context of refinement calculus has been explored by Morgan et al. [14,13]. In this approach the probabilistic refinement has been used to assess system dependability. Such an approach is much stronger than the approach described in this paper. Probabilistic refinement allows the developers to obtain algebraic solutions even without pruning the system state space. Meanwhile, probabilistic verification gives us only numeric solutions for restricted system models. In a certain sense, our approach can be seen as a property-wise refinement evaluation. Indeed, while evaluating dependability, we essentially check that, for the same samples of system parameters, the probability of system to hold a certain property is not decreased by refinement.

5.2 Conclusions

In this paper we proposed an approach to integrating probabilistic assessment of reliability into Event-B modelling. We defined reliability of a cyclic system as the probability of the system to stay in its operational state for a given number of iterations. Our approach to augmenting Event B models with probabilities allows us to give the semantic of a Markov process (or, in special cases, a Markov chain) to augmented models. In turn, this allow us to algebraically compute reliability by using any of numerous automated tools for reliability estimation.

In general, continuous-time Markov processes are more often used for dependability evaluation. However, the theory of refinement of systems with continuous behaviour has not reached maturity yet [3,15]. In this paper we showed that, by restricting the shape of Event-B models and augmenting them with probabilities, we can make a smooth transition to representing a cyclic system as a Markov process. This allow us to rely on standard techniques for assessing reliability.

In our future work it would be interesting to explore continuous-time reasoning as well as generalise the notion of refinement to take into account several dependability attributes.

Acknowledgments

This work is supported by IST FP7 DEPLOY Project. We also wish to thank the anonymous reviewers for their helpful comments.

References

1. Abrial, J.R.: Extending B without Changing it (for Developing Distributed Systems). In: Habiras, H. (ed.) First Conference on the B method, pp. 169–190. IRIN Institut de recherche en informatique de Nantes (1996)
2. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (2005)
3. Back, R.J.R., Petre, L., Porres, I.: Generalizing Action Systems to Hybrid Systems. In: Joseph, M. (ed.) FTRTFT 2000. LNCS, vol. 1926, pp. 202–213. Springer, Heidelberg (2000)
4. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
5. Craigen, D., Gerhart, S., Ralson, T.: Case study: Paris metro signaling system. IEEE Software, 32–35 (1994)
6. EU-project DEPLOY, `http://www.deploy-project.eu/`
7. Hallerstede, S., Hoang, T.S.: Qualitative probabilistic modelling in Event-B. In: Davies, J., Gibbons, J. (eds.) IFM 2007. LNCS, vol. 4591, pp. 293–312. Springer, Heidelberg (2007)
8. Hansson, H.: Time and Probability in Formal Design of Distributed Systems. Elsevier, Amsterdam (1995)
9. Kemeny, J.G., Snell, J.L.: Finite Markov Chains. D. Van Nostrand Company (1960)
10. Kwiatkowska, M.: Quantitative verification: models techniques and tools. In: ESEC/FSE 2007, pp. 449–458. ACM, New York (2007)
11. Kwiatkowska, M., Norman, G., Parker, D.: Controller dependability analysis by probabilistic model checking. In: Control Engineering Practice, pp. 1427–1434 (2007)
12. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Information and Computation 94, 1–28 (1991)
13. McIver, A.K., Morgan, C.C.: Abstraction, Refinement and Proof for Probabilistic Systems. Springer, Heidelberg (2005)
14. McIver, A.K., Morgan, C.C., Troubitsyna, E.: The probabilistic steam boiler: a case study in probabilistic data refinement. In: Proc. International Refinement Workshop, ANU, Canberra. Springer, Heidelberg (1998)
15. Meinicke, L., Smith, G.: A Stepwise Development Process for Reasoning about the Reliability of Real-Time Systems. In: Davies, J., Gibbons, J. (eds.) IFM 2007. LNCS, vol. 4591, pp. 439–458. Springer, Heidelberg (2007)
16. MRMC – Markov Reward Model Checker, `http://www.mrmc-tool.org/`
17. O'Connor, P.D.T.: Practical Reliability Engineering, 3rd edn. John Wiley & Sons, Chichester (1995)
18. PRISM – Probabilistic Symbolic Model Checker, `http://www.prismmodelchecker.org/`
19. Rigorous Open Development Environment for Complex Systems (RODIN): IST FP6 STREP project, `http://rodin.cs.ncl.ac.uk/`
20. Rigorous Open Development Environment for Complex Systems (RODIN): Deliverable D7, Event-B Language, `http://rodin.cs.ncl.ac.uk/`
21. RODIN. Event-B Platform, `http://www.event-b.org/`
22. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing 2(2), 250–273 (1995)
23. Storey, N.: Safety-Critical Computer Systems. Addison-Wesley, Reading (1996)
24. Villemeur, A.: Reliability, Availability, Maintainability and Safety Assessment. John Wiley & Sons, Chichester (1995)

Paper III

Formal Modelling and Verification of Service-Oriented Systems in Probabilistic Event-B

Anton Tarasyuk, Elena Troubitsyna and Linas Laibinis

Originally published in: J. Derrick et al. (Eds.), *Proceedings of 9th International Conference on Integrated Formal Methods (iFM 2012)*, LNCS 7321, 237–252, Springer, 2012

Formal Modelling and Verification of Service-Oriented Systems in Probabilistic Event-B

Anton Tarasyuk[1,2], Elena Troubitsyna[2], and Linas Laibinis[2]

[1] Turku Centre for Computer Science, Turku, Finland
[2] Åbo Akademi University, Turku, Finland
{anton.tarasyuk,elena.troubitsyna,linas.laibinis}@abo.fi

Abstract. Modelling and refinement in Event-B provides a scalable support for systematic development of complex service-oriented systems. This is achieved by a gradual transformation of an abstract service specification into its detailed architecture. In this paper we aim at integrating quantitative assessment of essential quality of service attributes into the formal modelling process. We propose an approach to creating and verifying a dynamic service architecture in Event-B. Such an architecture can be augmented with stochastic information and transformed into the corresponding continuous-time Markov chain representation. By relying on probabilistic model-checking techniques, we allow for quantitative evaluation of quality of service at early development stages.

1 Introduction

The main goal of service-oriented computing is to enable rapid building of complex software by assembling readily-available services. While promising productivity gain in the development, such an approach also poses a significant verification challenge – how to guarantee correctness of complex composite services?

In our previous work we have demonstrated how to build complex service-oriented systems (SOSs) by refinement in Event-B [12,11]. We have not only formalised Lyra – an industrial model-driven approach – but also augmented it with a systematic modelling of fault tolerance. However, within this approach we could not evaluate whether the designed fault tolerant mechanisms are appropriate, i.e., they suffice to meet the desired quality of service (QoS) attributes.

To address this issue, in this paper we propose an approach to building a dynamic service architecture – an Event-B model that formally represents the service orchestration. In particular, we define the set of requirements – the formal verification conditions – that allow us to ensure that the modelled service architecture faithfully represents the dynamic service behaviour. Such an Event-B model can be then augmented with stochastic information about system failures and duration of the orchestrated services. Essentially, this results in creating a continuous-time Markov chain (CTMC) model representation and hence enables the use of existing probabilistic model checking techniques to verify the desired

J. Derrick et al. (Eds.): IFM 2012, LNCS 7321, pp. 237–252, 2012.

QoS attributes. We demonstrate how to formulate a number of widely used QoS attributes as temporal logic formulae to be verified by PRISM [14]. Overall, our approach enables an early quantitative evaluation of essential QoS attributes and rigorous verification of the dynamic aspects of the system behaviour.

The paper is organised as follows. In Section 2 we briefly describe our formal modelling framework, Event-B, and also define its underlying transition system. Section 3 discusses SOSs and the associated dynamic service architectures. In Section 4 we propose a set of necessary requirements for SOSs as well as their formalisation in Event-B. Section 5 presents a small case study that illustrates building a dynamic service architecture. In Section 6 we explain how to convert Event-B models into CTMCs and also demonstrate the use of probabilistic model checking for analysis of QoS attributes. Finally, Section 7 gives some concluding remarks as well as overviews the related work in the field.

2 Modelling in Event-B

Event-B is a formal framework derived from the (classical) B method [1] to model parallel, distributed and reactive systems [2]. The Rodin platform provides tool support for modelling and formal verification (by theorem proving) in Event-B [17]. Currently, Event-B is used in the EU project Deploy to model dependable systems from from automotive, railway, space and business domains [9].

In Event-B, a system specification is defined using the notion of an *abstract state machine*. An abstract state machine encapsulates the model state, represented as a collection of model variables, and defines operations on this state via machine *events*. The occurrence of events represents the system behaviour. In a most general form, an Event-B model can be defined as follows.

Definition 1. *An Event-B model is a tuple* $(\mathcal{C}, \mathcal{S}, \mathcal{A}, \upsilon, \mathcal{I}, \Sigma, \mathcal{E}, Init)$ *where:*

- $\mathcal{C}$ *is a set of model constants;*
- $\mathcal{S}$ *is a set of model sets;*
- $\mathcal{A}$ *is a set of axioms over* $\mathcal{C}$ *and* $\mathcal{S}$*;*
- υ *is a set of system variables;*
- $\mathcal{I}$ *is a set of invariant properties over* υ*,* $\mathcal{C}$ *and* $\mathcal{S}$*;*
- Σ *is a model state space defined by all possible values of the vector* υ*;*
- $\mathcal{E} \subseteq \mathcal{P}(\Sigma \times \Sigma)$ *is a non-empty set of model events;*
- $Init$ *is a predicate defining an non-empty set of model initial states.*

The model variables υ are strongly typed by the constraining predicates specified in $\mathcal{I}$ and initialised by the predicate $Init$. Furthermore, $\mathcal{I}$ defines important properties that must be preserved by the system during its execution.

Generally, an event has the following form:

$$e \mathrel{\widehat{=}} \textbf{any } a \textbf{ where } G_e \textbf{ then } R_e \textbf{ end},$$

where e is the event's name, a is the list of local variables, and the *guard* G_e is a predicate over the model variables. The R_e is a next-state relation called *generalised substitution*. The guard defines the conditions under which the substitution

can be performed, i.e., when the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution nondeterministically.

If an event does not have local variables, it can be described simply as

$$e \mathrel{\widehat{=}} \textbf{when } G_e \textbf{ then } R_e \textbf{ end}.$$

Throughout the paper we will consider only such simple events. It does not make any impact on the generality of our approach because any event specified by using local variables can be always rewritten in the simple form.

Essentially, such an event definition is just a syntactic sugar for the underlying relation $e(\sigma, \sigma') = G_e(\sigma) \wedge R_e(\sigma, \sigma')$. Generally, a substitution R_e is defined by a *multiple* (possibly nondeterministic) assignment over a vector of system variables $u \subseteq \upsilon$, i.e., $u := X$, for some vector of values X. Hence the state transformation (via R_e) can be intuitively defined as $R_e(\sigma, \sigma') \Rightarrow \sigma' = \sigma[X/u]$, where $\sigma[X/u]$ is a substitution of values of the variables u in σ by the vector X. Obviously, due to presence of nondeterminism the successor state σ' is not necessarily unique.

For our purposes, it is convenient to define an Event-B model as a transition system. To describe a state transition for an Event-B model, we define two functions before and after from $\mathcal{E}$ to $\mathcal{P}(\Sigma)$ in a way similar to [8]:

$$\mathsf{before}(e) = \{\sigma \in \Sigma \mid \mathcal{I}(\sigma) \wedge G_e(\sigma)\} \quad \text{and}$$
$$\mathsf{after}(e) = \{\sigma' \in \Sigma \mid \mathcal{I}(\sigma') \wedge \big(\exists \sigma \in \Sigma \cdot \mathcal{I}(\sigma) \wedge G_e(\sigma) \wedge R_e(\sigma, \sigma')\big)\}.$$

These functions essentially return the domain and the range of an event e constrained by the model invariants $\mathcal{I}$. It is easy to see that e is enabled in σ if $\sigma \in \mathsf{before}(e)$. At any state σ, the behaviour of an Event-B machine is defined by all the enabled in σ events.

Definition 2. *The behaviour of any Event-B machine is defined by a transition relation* $\rightarrow$*:*

$$\frac{\sigma, \sigma' \in \Sigma \ \wedge \ \sigma' \in \bigcup\limits_{e \in \mathcal{E}_\sigma} \mathsf{after}(e)}{\sigma \rightarrow \sigma'},$$

where $\mathcal{E}_\sigma = \{e \in \mathcal{E} \mid \sigma \in \mathsf{before}(e)\}$ *is a subset of events enabled in* σ*.*

Remark 1. The soundness of Definition 2 is guaranteed by the *feasibility* property of Event-B events. According to this property, such σ' should always exist for any $\sigma \in \mathsf{before}(e)$, where $e \in \mathcal{E}_\sigma$ [2].

Together Definitions 1 and 2 allow us to describe any Event-B model as a transition system with state space Σ, transition relation $\rightarrow$ and a set of initial states defined by *Init*. Next we describe the essential structure of SOSs and reflect on our experience in modelling SOSs in Event-B.

3 Service-Oriented Systems

3.1 Service Orchestration

Service-oriented computing is a popular software development paradigm that facilitates building complex distributed services by coordinated aggregation of

lower-level services (called subservices). Coordination of a service execution is typically performed by a *service director* (or *service composer*). It is a dedicated software component that on the one hand, communicates with a service requesting party and on the other hand, orchestrates the service execution flow.

To coordinate service execution, the service director keeps information about subservices and their execution order. It requests the corresponding components to provide the required subservices and monitors the results of their execution. Let us note that any subservice might also be composed of several subservices, i.e., in its turn, the subservice execution might be orchestrated by its (sub)service director. Hence, in general, a SOS might have several layers of hierarchy.

Often, a service director not only ensures the predefined control and data flow between the involved subservices but also implements fault-tolerance mechanisms. Indeed, an execution of any subservice might fail. Then the service director should analyse the failed response and decide on the course of error recovery actions. For instance, if an error deemed to be recoverable, it might repeat the request to execute the failed subservice. However, it might also stop execution of a particular subservice to implement coordinated error recovery or abort the whole service execution due to some unrecoverable error.

Let us consider a simple case when the involved subservices S_1, S_2, ..., S_n should be executed in a fixed sequential order:

$$S_1 \longrightarrow S_2 \longrightarrow S_3 \ldots. \longrightarrow S_n$$

According the discussion above, the overall control flow can be graphically represented as shown in Fig. 1.

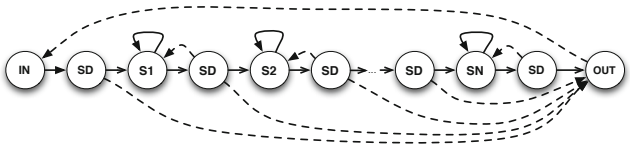


Fig. 1. Service flow in a service director

Here IN and OUT depict receiving new requests and sending service responses, while the service director SD monitors execution of the subservices and performs the required controlling or error recovery actions. These actions may involve requesting a particular subservice to repeat its execution (a dashed arrow from SD to S_i), aborting the whole service (a dashed arrow from SD to OUT), or allowing a subservice to continue its execution (a looping arrow for S_i).

Though we have considered a sequential service execution flow, the service execution per se might have any degree of parallelism. Indeed, any subservice might consist of a number of independent subservices $S_{i1}, ..., S_{ik}$ that can be executed in parallel. Such a service architecture allows the designer to improve performance or increase reliability, e.g., if parallel subservices replicate each other.

3.2 Towards Formalisation of Service Orchestration

In our previous work [12,11], we have proposed a formalisation of the service-oriented development method Lyra in the B and Event-B frameworks. In our approach, refinement formalises unfolding of architectural layers and consequently introduces explicit representation of subservices at the corresponding architectural layer. Reliance of refinement, decomposition and proof-based verification offers a scalable support for development of complex services and verification of their functional correctness. The result of refinement process is a detailed system specification that can be implemented on a targeted platform. However, before such an implementation is undertaken, it is desirable to evaluate whether the designed service meets its QoS requirements.

To enable such an evaluation, we propose to build a formal model that *explicitly* represents service orchestration, i.e., defines the *dynamic service architecture*, while suppressing unnecessary modelling details. Such a model can be augmented with probabilistic information and serve as an input for the evaluation of the desirable QoS attributes, as we will describe in Section 6.

To achieve this goal, we should strengthen our previous approach by formalising service orchestration requirements. Indeed, in [12,11] the service execution flow and possible parallelism were modelled via an abstract function *Next*. Essentially, this function served as an abstract scheduler of subservices. However, such a representation does not allow for a verification of service orchestration that is essential for building an adequate model of the dynamic service architecture.

We start our formalisation of service orchestration requirements by assuming that a service S is composed of a finite set of subservices $\{S_1, S_2, \ldots, S_n\}$ that are orchestrated by a service director. The behaviour of the service director consists of a number of activities $\{IN, OUT, SD\}$, where IN and OUT are modelling the start (i.e., receiving a service request) and the end (i.e., sending a service response) of the service execution flow. SD represents the decision making procedure performed by the service director after execution of any subservice, i.e., it computes whether to restart the execution of the current subservice, call the next scheduled subservice, or abort the service execution.

In Event-B, the subservices $S_1, ..., S_n$ as well as the service director activities IN, OUT, SD can be represented as groups of mutually exclusive events. Without losing generality, we will treat all these activities as single events.

Let us also tailor our generic definitions of before and after functions to modelling service-oriented systems. For a composite subservice S_i, i.e., a subservice that is a parallel composition of sub-subservices $S_{i1}, ..., S_{ik}$, we define

$$\mathsf{before}(S_i) = \bigcup_{j \in 1..k} \mathsf{before}(S_{ij}) \quad \text{and} \quad \mathsf{after}(S_i) = \bigcup_{j \in 1..k} \mathsf{after}(S_{ij}).$$

Moreover, we introduce a version of the function after that is "narrowed" with respect to a particular fixed pre-state σ:

$$\mathsf{after}_\sigma(e) = \{\sigma' \in \Sigma \mid \mathcal{I}(\sigma') \wedge \mathcal{I}(\sigma) \wedge G_e(\sigma) \wedge R_e(\sigma, \sigma')\}.$$

Essentially, this function gives the relational image of the next-state relation R_e for the given singleton set $\{\sigma\}$. We will rely on these definitions while postulating the service orchestration conditions that we present next.

4 Modelling the Dynamic Service Architecture

In this paper, we focus on modelling of SOSs that can provide service to only one request at any time instance. In other words, it means that once a SOS starts to serve a request, it becomes unavailable for the environment until the current service is finished or aborted. Such a discipline of service imposes the following requirements for receiving a service request and sending a service response:

(REQ1) *After receiving a service request, the service director is activated to handle it;*
(REQ2) *Once the service execution is finished, the service director is ready to receive a new service request;*
(REQ3) *Receiving a service request and sending a service response is not possible when orchestration of the service execution is still in process.*

Formally, the first two requirements can be formulated as follows:

$$\mathsf{after}(IN) \subseteq \mathsf{before}(SD) \quad \text{and} \quad \mathsf{after}(OUT) \subseteq \mathsf{before}(IN),$$

while the formalisation of (REQ3) can be defined by the following two predicates:

$$\forall e_1, e_2 \in \{IN, OUT, SD\} \cdot \mathsf{before}(e_1) \cap \mathsf{before}(e_2) = \varnothing,$$
$$\big(\mathsf{before}(IN) \cup \mathsf{before}(OUT)\big) \cap \big(\bigcup_{i \in 1..n} \mathsf{before}(S_i)\big) = \varnothing.$$

Essentially, these predicates state that IN and OUT cannot be enabled at the same time as any of subservices S_i or the service director event SD.

Moreover, the service director should follow the predefined order of the service execution. This, however, should not prevent the service director from interfering:

(REQ4) *At any moment only one of sequential subservices (i.e., only one of $S_1, ..., S_n$) can be active;*
(REQ5) *The service director has an ability to always provide a required control action upon execution of the active subservice (or any of parallel subservices);*

Formally, (REQ4) is ensured by requiring that the sets of states, where any two different subservices are enabled, are disjoint:

$$\forall i, j \in 1..n \cdot i \neq j \Rightarrow \mathsf{before}(S_i) \cap \mathsf{before}(S_j) = \varnothing,$$

while (REQ5) implies that SD can be enabled by execution of any subservice:

$$\forall i \in 1..n \cdot \mathsf{after}(S_i) \subseteq \mathsf{before}(S_i) \cup \mathsf{before}(SD).$$

Let us note that we delegate a part of the service director activity to the guards of the events modelling the subservices. Specifically, we allow a subservice to be executed in a cyclic manner without any interference from the service director, i.e., it may either block itself or continue its activity if it remains enabled.

However, if an active subservice blocks itself, it must enable the service director:

$$\forall i \in 1..n, \sigma \in \Sigma \cdot \sigma \in \mathsf{after}(S_i) \wedge \sigma \notin \mathsf{before}(S_i) \Rightarrow \sigma \in \mathsf{before}(SD)$$

that directly follows from the formalisation of (REQ5).

The next requirement concerns handling performed by the service director.

(REQ6) *The reaction of the service director depends on the result returned by the supervised subservice (or several parallel subservices) and can be one of the following:*

- *upon successful termination of the subservice, the service director calls the next scheduled subservice;*
- *in case of a recoverable failure of the subservice, the service director restarts it;*
- *in case of an unrecoverable failure (with respect to the main service) of the subservice, the service director aborts the execution of the whole service.*

Formally, it can be specified in the following way:

$$\forall i \in 1..n-1; \sigma \in \Sigma \cdot \sigma \in \mathsf{after}(S_i) \cap \mathsf{before}(SD) \Rightarrow \mathsf{after}_\sigma(SD) \subseteq \mathsf{before}(S_i) \cup \mathsf{before}(S_{i+1}) \cup \mathsf{before}(OUT)$$

and, for a special case when a subservice is the last one in the execution flow,

$$\forall \sigma \in \Sigma \cdot \sigma \in \mathsf{after}(S_n) \cap \mathsf{before}(SD) \Rightarrow \mathsf{after}_\sigma(SD) \subseteq \mathsf{before}(S_n) \cup \mathsf{before}(OUT).$$

In addition, it is important to validate that execution of any subservice cannot disable execution of the service director:

(REQ7) *None of active subservices can block execution of the service director.*

It means that, whenever SD is enabled, it stays enabled after execution of any subservice:

$$\forall i \in 1..n; \sigma \in \Sigma \cdot \sigma \in \mathsf{before}(SD) \cup \mathsf{before}(S_i) \Rightarrow \mathsf{after}_\sigma(S_i) \subseteq \mathsf{before}(SD).$$

In the general case, when the execution flow of a SOS contains parallel compositions of subservices, a couple of additional requirements are needed:

(REQ8) *All the subservices of a parallel composition must be independent of each other, i.e., their execution order does not affect the execution of the overall service;*

(REQ9) *Execution of any subservice of a parallel composition cannot block execution of any other active parallel subservice;*

In terms of model events, the independence requirement means that forward relational composition of two different events running in any order have the same range:

$$\forall i \in 1..n; j, l \in 1..k \cdot \mathsf{after}(S_{ij}; S_{il}) = \mathsf{after}(S_{il}; S_{ij}),$$

where

$$\mathsf{after}(e_i; e_j) = \big\{\sigma'' \in \Sigma \mid \sigma'' \in \mathsf{after}(e_j) \wedge \big(\exists \sigma, \sigma' \in \Sigma \cdot \sigma \in \mathsf{before}(e_i) \wedge \sigma' \in \mathsf{after}(e_i) \cap \mathsf{before}(e_j) \wedge R_{e_i}(\sigma, \sigma') \wedge R_{e_j}(\sigma', \sigma'')\big)\big\}.$$

Such a definition of independence is imposed by the interleaving semantics of Event-B and the fact that the framework does not support events composition directly. Finally, we formulate the requirement (REQ9) as follows:

$$\forall i \in 1..n; j, l \in 1..k \cdot j \neq l \wedge \\ \sigma \in \mathsf{before}(S_{ij}) \cup \mathsf{before}(S_{il}) \Rightarrow \mathsf{after}_{\sigma}(S_{ij}) \subseteq \mathsf{before}(S_{il})$$

$$\text{and} \quad \forall i \in 1..n; j \in 1..k \cdot \mathsf{after}(S_{ij}) \cap \mathsf{before}(S_{ij}) \neq \varnothing,$$

where the second formula states that any subservice can continue its execution without interference from the service director.

To verify that an Event-B model of a SOS satisfies the requirements (REQ1)–(REQ9), their formalisation (based on concrete model elements) could be generated and added as a collection of model theorems. A similar approach has been applied in [8]. The generation and proof of additional model theorems can be partially automated, provided that the mapping between the model events and the subservices as well as the activities IN, OUT, and SD is supplied.

In the next section we consider a small example of a SOS. To demonstrate our approach, we formally model the system dynamic architecture in Event-B and then show that the model satisfies the formulated flow conditions.

5 Case Study

We model a simple dynamic service architecture that consists of a service director and five subservices. The latter can be structured into two composite subservices, S_1 and S_2, where S_1 is a parallel composition of the subservices S_{11}, S_{12} and S_{13}, while S_2 consists of the parallel subservices S_{21} and S_{22}. Next we define the fault assumptions and decision rules used by the service director. Besides (fully) *successful* termination of subservices, the following alternatives are possible:

- the subservice S_{11} can terminate with a *transient failure*, in which case its execution should be restarted. The total number of retries cannot exceed the predefined upperbound number MAX;
- the subservices S_{12} and S_{13} can terminate with a *permanent failure*. Moreover, each of them may return a *partially successful* result, complementing the result of the other subservice;
- the subservices S_{21} and S_{22} can also terminate with a *permanent failure*. These subservices are functionally identical, thus successful termination of one of them is sufficient for the overall success of S_2.

Fig. 2 shows the events that abstractly model the behaviour of subservices. The variables srv_{ij}, where $i \in 1..2$ and $j \in 1..3$, represent statuses of the corresponding subservices. Here, the value *nd* (meaning "not defined") is used to distinguish between the subservices that are currently inactive, and those that are active but have not yet returned any result. The value *nok* stands for a permanent failure of a subservice, while the values *ok* and *pok* represent respectively successful and partially successful subservice execution.

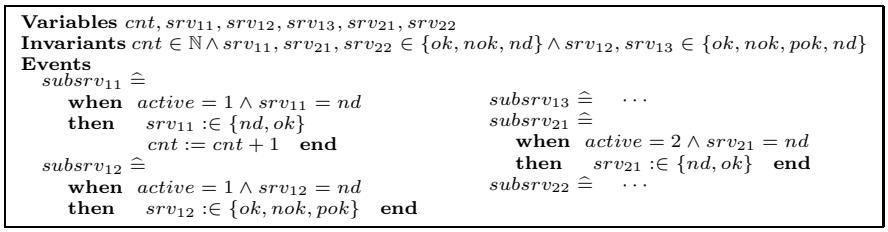


Fig. 2. Case study: modelling subservices in Event-B

Initially all the subservices have the status *nd*. Note that, in case of a transient failure of S_{11}, the value of srv_{11} remains *nd* and, as a result, the subservice can be restarted. The counter variable *cnt* stores the number of retries of S_{11}.

To provide the overall service, the following necessary conditions must be satisfied:

- S_{11} returns a successful result within MAX retries;
- both S_{12} and S_{13} do not fail and at least one of them returns a (fully) successful result;
- at least one of S_{21} and S_{22} does not fail.

The service director controls execution of subservices and checks preservation of these conditions. The events modelling behaviour of the service director are shown in Fig. 3. Here, the boolean variable *idle* stores the status of the overall service, i.e., whether the service director is waiting for a new service request or the service execution is already in progress. The variable *active* indicates which group of subservices or which activity of the service director is currently enabled. The value of the boolean variable *abort* shows whether one of the conditions necessary for successful completion of the service has been violated and thus the service execution should be interrupted. Finally, the variable $failed$ counts the number of such interrupted service requests. Please note that the service director activities $sd\_success_1$ and sd_fail_1 may run in parallel with $subsrv_{11}$. On the other hand, $sd\_success_2$ can be activated even if only one of $subsrv_{21}$ and $subsrv_{22}$ has been successfully executed and the other one is still running.

For our case study, it is easy to prove that the presented model satisfies the requirements (REQ1)–(REQ9). For instance, a proof of (REQ7) preservation can be split into two distinct cases, when $active = 1$ and $active = 2$. Let us consider the first case. When $\sigma \in \mathsf{before}(sd\_success_1)$, then all the subservices' events are disabled and the requirement is obviously satisfied. When $\sigma \in \mathsf{before}(sd\_fail_1)$, at least one of the disjuncts in the guard of sd_fail_1 is satisfied. In the case when $cnt > MAX$ is true, $subsrv_{11}$ can only increase the value of *cnt*, and neither $subsrv_{12}$ nor $subsrv_{13}$ can modify it. In the case when either ($srv_{12} = nok \vee srv_{13} = nok$) or ($srv_{12} = pok \wedge srv_{13} = pok$) is true, both $subsrv_{12}$ and $subsrv_{13}$ are disabled, and since $subsrv_{11}$ cannot affect any of these two guards the requirement is satisfied. Overall, the formal proofs for this model are simple though quite tedious.

Variables $idle, active, abort, failed$
Invariants $idle \in BOOL \wedge active \in 0..3 \wedge abort \in BOOL \wedge failed \in \mathbb{N}$
Events

$in \mathrel{\widehat{=}}$
when $active = 0 \wedge idle = TRUE$
then $idle := FALSE$ **end**

$sd_{in} \mathrel{\widehat{=}}$
when $active = 0 \wedge idle = FALSE$
then $active := 1$ **end**

$out_{success} \mathrel{\widehat{=}}$
when $active = 3 \wedge abort = FALSE$
then $active, cnt := 1, 0$
$idle := TRUE$
$srv_{11} := nd$
$srv_{12} := nd \ \ldots$ **end**

$out_{fail} \mathrel{\widehat{=}}$
when $active = 3 \wedge abort = TRUE$
then $\ldots$
$failed := failed + 1$
$abort := FALSE$ **end**

$sd\_success_1 \mathrel{\widehat{=}}$
when $active = 1 \wedge$
$srv_{11} = ok \wedge cnt \leq MAX \wedge$
$srv_{12} \neq nok \wedge srv_{13} \neq nok \wedge$
$(srv_{12} = ok \vee srv_{13} = ok)$
then $active := 2$ **end**

$sd\_fail_1 \mathrel{\widehat{=}}$
when $active = 1 \wedge (cnt > MAX \vee$
$srv_{12} = nok \vee srv_{13} = nok \vee$
$(srv_{12} = pok \wedge srv_{13} = pok))$
then $active, abort := 3, TRUE$ **end**

$sd\_success_2 \mathrel{\widehat{=}}$
when $active = 2 \wedge$
$(srv_{21} = ok \vee srv_{22} = ok)$
then $active := 3$ **end**

$sd\_fail_2 \mathrel{\widehat{=}}$
when $active = 2 \wedge$
$srv_{21} = nok \wedge srv_{22} = nok$
then $active, abort := 3, TRUE$ **end**

Fig. 3. Case study: modelling the service director in Event-B

As we have mentioned in the previous section, the verification of an Event-B model against the formulated requirements (`REQ1`)–(`REQ9`) is based on generation and proof of a number of Event-B theorems in the Rodin platform. However, for more complex, industrial-size systems, it can be quite difficult to prove such theorems in Rodin. To tackle this problem, some external mechanised proving systems, such as HOL or Isabelle, can be used. Bridging the Rodin platform with such external provers is currently under development.

The goal of building a model of dynamic service architecture is to enable quantitative evaluation of QoS attributes. In the next section we show how an Event-B machine can be represented by a CTMC and probabilistic model checking used to achieve the desired goal.

6 Probabilistic Verification in Event-B

6.1 Probabilistic Event-B

In this paper, we aim at quantitative verification of QoS of SOSs modelled in Event-B. To perform such a verification, we will transform Event-B models defining dynamic service architecture into CTMCs. The properties that we are interested to verify are the time-bounded reachability and reward properties related to a possible abort of service execution. For continuous-time models, such probabilistic properties can be specified as CSL (Continuous Stochastic Logic) formulae [3,4]. A detailed survey and specification patterns for probabilistic properties can be found in [7]. There are several examples of properties of SOSs that can be interesting for verification:

- what is the probability that at least one service execution will be aborted during a certain time interval?
- what is the probability that a number of aborted services during a certain time interval will not exceed some threshold?
- what is the mean number of served requests during a certain time interval?
- what is the mean number of failures of some particular subservice during a certain time interval?

To transform an Event-B machine into a CTMC, we augment all the events with information about probability and duration of all the actions that may occur during its execution. More specifically, we refine all the events by their probabilistic counterparts.

Let us consider a system state $\sigma \in \Sigma$ and an event $e \in \mathcal{E}$ such that $\sigma \in \mathsf{before}(e)$. Assume that R_e can transform σ to a set of states $\{\sigma'_1, \ldots, \sigma'_m\}$, where $m \geq 1$. Please recall that in Event-B, if $m > 1$ then the choice between the successor states is nondeterministic. We augment every such state transformation with a constant *rate* $\lambda_i \in \mathbb{R}^+$, where λ_i is a parameter of *the exponentially distributed sojourn time* that the system will spend in the state σ before it goes to the new state σ'_i. In such a way, we can replace a nondeterministic choice between the possible successor states by the probabilistic choice associated with the (exponential) race condition.

It is easy to show that such a replacement is a valid refinement step. Indeed, let p_i be a probability to choose a state transformation $\sigma \to \sigma'_i$, $\sigma \notin \{\sigma'_1, \ldots, \sigma'_m\}$. For $i \in 1..m$, it is convenient to define p_i as:

$$p_i = \frac{\lambda_i}{\sum\limits_{j=1}^{m} \lambda_j}.$$

The probabilities p_i define a next-state distribution for the current state σ. Refinement of the nondeterministic branching by the (discrete) probabilistic one is a well-known fact (see [15] for instance), which directly implies the validity of the refinement.

We adopt the notation $\lambda_e(\sigma, \sigma')$ to denote the transition rate from σ to σ' via the event e, where $\sigma \in \mathsf{before}(e)$ and $R_e(\sigma, \sigma')$. Augmenting all the event actions with transition rates, we can respectively modify Definition 2 as follows.

Definition 3. *The behaviour of any probabilistically augmented Event-B machine is defined by a transition relation* $\xrightarrow{\Lambda}$*:*

$$\frac{\sigma, \sigma' \in \Sigma \ \wedge \ \sigma' \in \bigcup\limits_{e \in \mathcal{E}_\sigma} \mathsf{after}(e)}{\sigma \xrightarrow{\Lambda} \sigma'},$$

where $\Lambda = \sum\limits_{e \in \mathcal{E}_\sigma} \lambda_e(\sigma, \sigma')$.

With such a probabilistic transition relation, an Event-B machine becomes a CTMC, whereas p_i are the one-step transition probabilities of the embedded (discrete-time) Markov chain. Such an elimination of nondeterminism between

enabled events is not always suitable for modelling. However, for SOSs this assumption seems quite plausible. Indeed, the fact that execution of two or more simultaneously enabled services may "lead" to the same state usually means that all these services share the same functionality. In this situation it is natural to expect that the overall transition rate will increase and thus summing of the corresponding subservice rates looks absolutely essential. Moreover, for parallel composition of subservices, the interleaving semantics of Event-B perfectly coheres with the fact that the probability that two or more exponentially distributed transition delays elapse at the same time moment is zero.

Generally, we can assume that $\sigma \in \{\sigma'_1, \ldots, \sigma'_m\}$ and attach a rate for this *skip* transformation as well. While participating in the race, such a transition does not affect it (because it does not change the system state and due to the memoryless property of the exponential distribution). However, the skip transition can be useful for verification of specific reward properties, e.g., the number of restarts for a particular subservice, the number of lost customers in the case of buffer overflow, etc. Obviously, such a transition is excluded from the calculation of p_i.

6.2 Case Study: Quantitative Modelling and Verification

Now let us perform quantitative verification of QoS attributes of the SOS presented in our case study using the probabilistic symbolic model checker PRISM. We start by creating a PRISM specification corresponding to our Event-B model. Short guidelines for Event-B to PRISM model transformation can be found in our previous work [19]. Fig. 4 and Fig. 5 show the resulting PRISM model as well as the rates we attached to all the model transitions. The behaviour of subservices is modelled by two modules S_1 and S_2. Note that the rate of successful execution of S_{11} is decreasing with the number of retries.

In Fig. 5, the modules SD and IN_OUT model behaviour of the service director. Since the model checker cannot work with infinite sets we have bounded from above the number of interrupted service requests by the predefined constant value MAX_failed. Such a restriction is reasonable because when the number of interrupted service requests exceeds some acceptable threshold, the system is usually treated as unreliable and must be redesigned.

Various properties that can be probabilistically verified for such a system were presented in the beginning of this section. In particular, the following CSL property is used to analyse the likelihood that a service request is interrupted as time progresses:

$$\mathbf{P}_{=?}[\mathbf{F} \leq T\ abort].$$

Usually the probability to "lose" at least one request is quite high (for instance, it equals 0.99993 for 10^4 time units and rates presented in Fig. 4–5). Therefore, it is interesting to assess the probability that the number of interrupted (failed) service request will exceed some threshold or reach the predefined acceptable threshold:

$$\mathbf{P}_{=?}[\mathbf{F} \leq T\ (failed > 10)] \quad \text{and} \quad \mathbf{P}_{=?}[\mathbf{F} \leq T\ (failed = MAX\_failed)].$$

Fig. 6(a) demonstrates how these probabilities change over a period of $T = 10^4$ time units.

// successful service rates of subservices
const double $\alpha_{11} = 0.9$; **const double** $\alpha_{12} = 0.1$; **const double** $\alpha_{13} = 0.12$;
const double $\alpha_2 = 0.085$;

const double $\gamma = 0.001$; *// transient failure rate of* S_{11}

// partially successful service rates of S_{12} *and* S_{13}
const double $\beta_{12} = 0.025$; **const double** $\beta_{13} = 0.03$;

// permanent failure rates of $S_{12}, S_{13},$ *and* $S_{21}(S_{22})$
const double $\delta_{12} = 0.001$; **const double** $\delta_{13} = 0.002$; **const double** $\delta_2 = 0.003$;

const int $MAX = 5$; *// upperbound for retries of* S_{11}

// subservice states: $0 = nd, 1 = ok, 2 = nok, 3 = pok$
global srv_{11} : [0..1] **init** 0; **global** srv_{12} : [0..1] **init** 0; ... **global** cnt : [0..100] **init** 0;

module S_1

$[]\ (active = 1)\&(srv_{11} = 0) \rightarrow \alpha_{11}/(cnt+1) : (srv'_{11} = 1) + \gamma : (cnt' = cnt + 1)$;
$[]\ (active = 1)\&(srv_{12} = 0) \rightarrow \alpha_{12} : (srv'_{12} = 1) + \delta_{12} : (srv'_{12} = 2) + \beta_{12} : (srv'_{12} = 3)$;
$[]\ (active = 1)\&(srv_{13} = 0) \rightarrow \alpha_{13} : (srv'_{13} = 1) + \delta_{13} : (srv'_{13} = 2) + \beta_{13} : (srv'_{13} = 3)$;

endmodule

module S_2

$[]\ (active = 2)\&(srv_{21} = 0) \rightarrow \alpha_2 : (srv'_{21} = 1) + \delta_2 : (srv'_{21} = 2)$;
$[]\ (active = 2)\&(srv_{22} = 0) \rightarrow \alpha_2 : (srv'_{22} = 1) + \delta_2 : (srv'_{22} = 2)$;

endmodule

Fig. 4. Case study: modelling subservices in PRISM

const double $\lambda = 0.2$ *// service request arrival rate*
const double $\mu = 0.6$; *// service director's output rate*
const double $\eta = 1$; *// service director's handling rate*

const int $MAX\_failed = 40$; *// max acceptable threshold for the failed requests*

global $abort$: **bool init** $false$; *// 0 =* $nd, 1 = ok, 2 = nok, 3 = pok$
global $active$: [0..3] **init** 1; *// 1 =* $IN, 2 = S_1..S_3, 3 = S_4..S_5, 4 = OUT$

module SD

$[]\ (active = 0)\&(!idle) \rightarrow \eta : (active' = 1)$;
$[]\ (active = 1)\&(srv_{11} = 1)\&(cnt \leq MAX)\&(srv_{12} \neq 2)\&(srv_{13} \neq 2)\&$
$(srv_{12} = 1 \,|\, srv_{13} = 1) \rightarrow \eta : (active' = 2)$;
$[]\ (active = 1)\&(srv_{12} = 2 \,|\, srv_{13} = 2 \,|\, cnt > MAX \,|\, (srv_{12} = 3 \,\&\, srv_{13} = 3)) \rightarrow$
$\eta : (active' = 3)\&(abort' = true)$;
$[]\ (active = 2)\&(srv_{21} = 1 \,|\, srv_{22} = 1) \rightarrow \eta : (active' = 3)$;
$[]\ (active = 2)\&(srv_{21} = 2)\&(srv_{22} = 2) \rightarrow \eta : (active' = 3)\&(abort' = true)$;

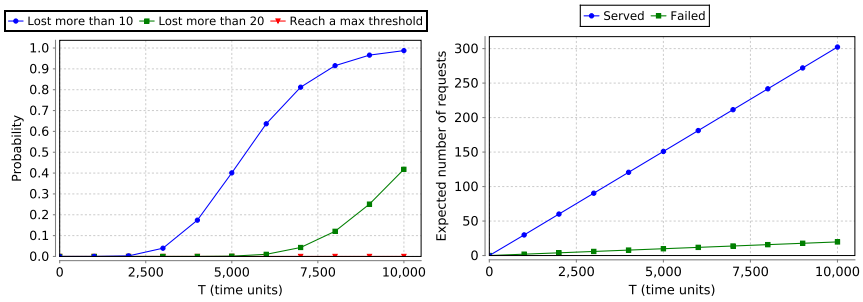
endmodule

module IN_OUT

$idle$: **bool init** $true$;
$failed$: $[0..MAX\_failed + 1]$ **init** 0;
$[]\ (active = 0)\&(idle) \rightarrow \lambda : (idle' = false)$;
$[]\ (active = 3)\&(!abort) \rightarrow \mu : (active' = 0)\&(cnt' = 0)\&(idle' = true)\&$
$(srv'_{11} = 0)\&(srv'_{12} = 0)\&(srv'_{13} = 0)\&(srv'_{21} = 0)\&(srv'_{22} = 0)$;
$[]\ (active = 3)\&(abort)\&(failed \leq MAX\_failed) \rightarrow \mu : (active' = 0)\&(cnt' = 0)\&$
$(idle' = true)\&(abort' = false)\&(failed' = failed + 1)\&(srv'_{11} = 0)\& \ldots$;
$[]\ (active = 3)\&(abort)\&(failed > MAX\_failed) \rightarrow \mu : (active' = 0)\&(cnt' = 0)\&$
$(idle' = true)\&(abort' = false)\&(srv'_{11} = 0)\& \ldots$;

endmodule

Fig. 5. Case study: modelling the service director in PRISM



(a) Probabilistic properties (b) Reward-based properties

Fig. 6. Case study: results of probabilistic analysis by PRISM

The next part of the analysis is related to estimation of the failed/served requests over a period of T time units. This analysis was accomplished in PRISM using its support for reward-based properties. For each class of states corresponding to the OUT activity of the service director, a cost structure which assigns a cost of 1 is used. The properties

$$\mathbf{R}\{\text{`}num\_failed\text{'}\}_{=?}[\mathbf{C} \leq T] \quad \text{and} \quad \mathbf{R}\{\text{`}num\_served\text{'}\}_{=?}[\mathbf{C} \leq T]$$

are then used to compute the expected number of failed and served service requests cumulated by the system over T time units (see Fig. 6(b)).

7 Related Work and Conclusions

Modelling of SOSs is a topic of active ongoing research. Here we only overview two research strands closely related to our approach: 1) formal approaches to modelling SOSs and quantitative assessment of QoS, and 2) the approaches that facilitate explicit reasoning about the dynamic system behaviour in Event-B.

Significant research efforts have been put into developing dedicated languages for modelling SOSs and their dynamic behaviour. For instance, Orc [10] is a language specifically designed to provide a formal basis for conceptual programming of web-services, while COWS (Calculus for Orchestration of Web Services) is a process calculus for specifying and combining services [13]. Similarly to our approach, the stochastic extension of COWS relies on CTMCs and the PRISM model checker to enable quantitative assessment of QoS parameters [16]. A fundamental approach to stochastic modelling of SOSs is proposed by De Nicola et al. [6]. The authors define a structural operational semantics of MarCaSPiS – a Markovian extension of CaSPiS (Calculus of Sessions and Pipelines). The proposed semantics is based on a stochastic version of two-party (CCS-like) synchronisation, typical for service-oriented approaches, while guaranteeing associativity and commutativity of parallel composition of services.

In contrast, in our approach we rely on a formal framework that enables unified modelling of functional requirements and orchestration aspects of SOSs. We have extended our previous work on formalisation of Lyra, an UML-based approach

for development of SOSs [12,11], in two ways. First, we defined a number of formal verification requirements for service orchestration. Second, we proposed a probabilistic extension of Event-B that, in combination with the probabilistic model checker PRISM, enables stochastic assessment of QoS attributes.

There is also an extensive body of research on applying of model checking techniques for quantitative evaluation of QoS (see, e.g., [5]). We however focus on combining formal refinement techniques with quantitative assessment of QoS.

Several approaches have been recently proposed to enable explicit reasoning about the dynamic system behaviour in Event-B. Iliasov [8] has proposed to augment Event-B models with additional proof obligations derived from the provided use case scenarios and control flow diagrams. An integration of CSP and Event-B to facilitate reasoning about the dynamic system behaviour has been proposed by Schneider et al. [18]. In the latter work, CSP is used to provide an explicit control flow for an Event-B model as well as to separate the requirements dependent on the control flow information. The approach we have taken is inspired by these works. We however rely solely on Event-B to built a dynamic service architecture and verify the required service orchestration.

We can summarise our technical contribution as being two-fold. First, we have put forward an approach to defining a dynamic service architecture in Event-B. Such an Event-B model represents service orchestration explicitly, i.e., it depicts interactions of a service director with the controlled services, the order of service execution as well as fault tolerance mechanisms. Moreover, we have formally defined the conditions required for verification of a dynamic service architecture modelled in Event-B. Second, we have demonstrated how to augment such a model with stochastic information and transform it into a CTMC. It allows us to rely on probabilistic model checking techniques to quantitatively assess the desired quality of essential service attributes. In our future work, it would be interesting to extend the proposed approach to deal with dynamic reconfiguration as well as unreliable communication channels.

References

1. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press (2005)
2. Abrial, J.R.: Modeling in Event-B. Cambridge University Press (2010)
3. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Verifying Continuous Time Markov Chains. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 269–276. Springer, Heidelberg (1996)
4. Baier, C., Katoen, J.-P., Hermanns, H.: Approximate Symbolic Model Checking of Continuous-Time Markov Chains (Extended Abstract). In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 146–161. Springer, Heidelberg (1999)
5. Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic QoS Management and Optimization in Service-Based Systems. IEEE Trans. Softw. Eng. 37, 387–409 (2011)
6. De Nicola, R., Latella, D., Loreti, M., Massink, M.: MarCaSPiS: a Markovian Extension of a Calculus for Services. Electronic Notes in Theoretical Computer Science 229(4), 11–26 (2009)

7. Grunske, L.: Specification patterns for probabilistic quality properties. In: International Conference on Software Engineering, ICSE 2008, pp. 31–40. ACM (2008)
8. Iliasov, A.: Use Case Scenarios as Verification Conditions: Event-B/Flow Approach. In: Troubitsyna, E.A. (ed.) SERENE 2011. LNCS, vol. 6968, pp. 9–23. Springer, Heidelberg (2011)
9. Industrial Deployment of System Engineering Methods Providing High Dependability and Productivity (DEPLOY): IST FP7 IP Project, http://www.deploy-project.eu/
10. Kitchin, D., Quark, A., Cook, W., Misra, J.: The Orc Programming Language. In: Lee, D., Lopes, A., Poetzsch-Heffter, A. (eds.) FMOODS/FORTE 2009. LNCS, vol. 5522, pp. 1–25. Springer, Heidelberg (2009)
11. Laibinis, L., Troubitsyna, E., Leppänen, S.: Formal Reasoning about Fault Tolerance and Parallelism in Communicating Systems. In: Butler, M., Jones, C., Romanovsky, A., Troubitsyna, E. (eds.) Methods, Models and Tools for Fault Tolerance. LNCS, vol. 5454, pp. 130–151. Springer, Heidelberg (2009)
12. Laibinis, L., Troubitsyna, E., Leppänen, S., Lilius, J., Malik, Q.A.: Formal Model-Driven Development of Communicating Systems. In: Lau, K.-K., Banach, R. (eds.) ICFEM 2005. LNCS, vol. 3785, pp. 188–203. Springer, Heidelberg (2005)
13. Lapadula, A., Pugliese, R., Tiezzi, F.: A Calculus for Orchestration of Web Services. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 33–47. Springer, Heidelberg (2007)
14. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of Probabilistic Real-Time Systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
15. McIver, A.K., Morgan, C.C.: Abstraction, Refinement and Proof for Probabilistic Systems. Springer (2005)
16. Prandi, D., Quaglia, P.: Stochastic COWS. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 245–256. Springer, Heidelberg (2007)
17. Rodin: Event-B Platform, http://www.event-b.org/
18. Schneider, S., Treharne, H., Wehrheim, H.: A CSP Approach to Control in Event-B. In: Méry, D., Merz, S. (eds.) IFM 2010. LNCS, vol. 6396, pp. 260–274. Springer, Heidelberg (2010)
19. Tarasyuk, A., Troubitsyna, E., Laibinis, L.: Quantitative Reasoning about Dependability in Event-B: Probabilistic Model Checking Approach. In: Dependability and Computer Engineering: Concepts for Software-Intensive Systems, pp. 459–472. IGI Global (2011)

Paper IV

Formal Development and Assessment of a Reconfigurable On-board Satellite System

Anton Tarasyuk, Inna Pereverzeva, Elena Troubitsyna, Timo Latvala and Laura Nummila

Originally published in: Frank Ortmeier, Peter Daniel (Eds.), *Proceedings of 31st International Conference on Computer Safety, Reliability and Security (SAFECOMP 2012)*, LNCS 7612, 210–222, Springer, 2012

Formal Development and Assessment of a Reconfigurable On-board Satellite System

Anton Tarasyuk[1,2], Inna Pereverzeva[1,2], Elena Troubitsyna[1], Timo Latvala[3], and Laura Nummila[3]

[1] Åbo Akademi University, Turku, Finland
[2] Turku Centre for Computer Science, Turku, Finland
[3] Space Systems Finland, Espoo, Finland
{inna.pereverzeva,anton.tarasyuk,elena.troubitsyna}@abo.fi,
{timo.latvala,laura.nummila}@ssf.fi

Abstract. Ensuring fault tolerance of satellite systems is critical for achieving goals of the space mission. Since the use of redundancy is restricted by the size and the weight of the on-board equipments, the designers need to rely on dynamic reconfiguration in case of failures of some components. In this paper we propose a formal approach to development of dynamically reconfigurable systems in Event-B. Our approach allows us to build the system that can discover possible reconfiguration strategy and continue to provide its services despite failures of its vital components. We integrate probabilistic verification to evaluate reconfiguration alternatives. Our approach is illustrated by a case study from aerospace domain.

Keywords: Formal modelling, fault tolerance, Event-B, refinement, probabilistic verification.

1 Introduction

Fault tolerance is an important characteristics of on-board satellite systems. One of the essential means to achieve it is redundancy. However, the use of (hardware) component redundancy in spacecraft is restricted by the weight and volume constraints. Thus, the system developers need to perform a careful cost-benefit analysis to minimise the use of spare modules yet achieve the required level of reliability.

Despite such an analysis, Space System Finland has recently experienced a double-failure problem with a system that samples and packages scientific data in one of the operating satellites. The system consists of two identical modules. When one of the first module subcomponents failed, the system switched to the use of the second module. However, after a while a subcomponent of the spare has also failed, so it became impossible to produce scientific data. To not lose the entire mission, the company has invented a solution that relied on healthy subcomponents of both modules and a complex communication mechanism to restore system functioning. Obviously, a certain amount of data has been lost before a repair was deployed. This motivated our work on exploring proactive

F. Ortmeier and P. Daniel (Eds.): SAFECOMP 2012, LNCS 7612, pp. 210–222, 2012.

solutions for fault tolerance, i.e., planning and evaluating of scenarios implementing a seamless reconfiguration using a fine-grained redundancy.

In this paper we propose a formal approach to modelling and assessment of on-board reconfigurable systems. We generalise the ad-hoc solution created by Space Systems Finland and propose an approach to formal development and assessment of fault tolerant satellite systems. The essence of our modelling approach is to start from abstract modelling functional goals that the system should achieve to remain operational, and to derive reconfigurable architecture by refinement in the Event-B formalism [1]. The rigorous refinement process allows us to establish the precise relationships between component failures and goal reachability. The derived system architecture should not only satisfy functional requirements but also achieve its reliability objective. Moreover, since the reconfiguration procedure requires additional inter-component communication, the developers should also verify that system performance remains acceptable. Quantitative evaluation of reliability and performance of probabilistically augmented Event-B models is performed using the PRISM model checker [8].

The main novelty of our work is in proposing an integrated approach to formal derivation of reconfigurable system architectures and probabilistic assessment of their reliability and performance. We believe that the proposed approach facilitates early exploration of the design space and helps to build redundancy-frugal systems that meet the desired reliability and performance requirements.

2 Reconfigurable Fault Tolerant Systems

2.1 Case Study: Data Processing Unit

As mentioned in the previous section, our work is inspired by a solution proposed to circumvent the double failure occurred in a currently operational on-board satellite system. The architecture of that system is similar to Data Processing Unit (DPU) – a subsystem of the European Space Agency (ESA) mission BepiColombo [2]. Space Systems Finland is one of the providers for BepiColombo. The main goal of the mission is to carry out various scientific measures to explore the planet Mercury. DPU is an important part of the Mercury Planetary Orbiter. It consists of four independent components (computers) responsible for receiving and processing data from four sensor units: SIXS-X (X-ray spectrometer), SIXS-P (particle spectrometer), MIXS-T (telescope) and MIXS-C (collimator).

The behaviour of DPU is managed by telecommands (TCs) received from the spacecraft and stored in a circular buffer (TC pool). With a predefined rate, DPU periodically polls the buffer, decodes a TC and performs the required actions. Processing of each TC results in producing telemetry (TM). Both TC and TM packages follow the syntax defined by the ESA Packet Utilisation Standard [12]. As a result of TC decoding, DPU might produce a housekeeping report, switch to some mode or initiate/continue production of *scientific data*. The main purpose of DPU is to ensure a required rate of producing TM containing scientific data. In this paper we focus on analysing this particular aspect of the system

behaviour. Hence, in the rest of the paper, TC will correspond to the telecommands requiring production of scientific data, while TM will designate packages containing scientific data.

2.2 Goal-Oriented Reasoning about Fault Tolerance

We use the notion of a goal as a basis for reasoning about fault tolerance. Goals – the functional and non-functional objectives that the system should achieve – are often used to structure the requirements of dependable systems [7,9].

Let $\mathcal{G}$ be a predicate that defines a desired goal and $\mathcal{M}$ be a system model. Ideally, the system design should ensure that the goal can be reached "infinitely often". Hence, while verifying the system, we should establish that

$$\mathcal{M} \models \Box\Diamond\,\mathcal{G}.$$

The main idea of a goal-oriented development is to decompose the high-level system goals into a set of subgoals. Essentially, subgoals define the intermediate stages of achieving a high-level goal. In the process of goal decomposition we associate system components with tasks – the lowest-level subgoals. A component is associated with a task if its functionality enables establishing the goal defined by the corresponding task.

For instance, in this paper we consider *"produce scientific TM"* as a goal of DPU. DPU sequentially enquires each of its four components to produce its part of scientific data. Each component acquires fresh scientific data from the corresponding sensor unit (SIXS-X, SIXS-P, MIXS-T or MIXS-C), preprocesses it and makes available to DPU that eventually forms the entire TM package. Thus, the goal can be decomposed into four similar tasks *"sensor data production"*.

Generally, the goal $\mathcal{G}$ can be decomposed into a finite set of tasks:

$$\mathcal{T} = \{task_j \mid j \in 1..n \wedge n \in \mathbb{N}_1\},$$

Let also $\mathcal{C}$ be a finite set of components capable of performing tasks from $\mathcal{T}$:

$$\mathcal{C} = \{comp_j \mid j \in 1..m \wedge m \in \mathbb{N}_1\},$$

where $\mathbb{N}_1$ is the set of positive integers. Then the relation Φ defined below associates components with the tasks:

$$\Phi \in \mathcal{T} \leftrightarrow \mathcal{C}, \;\; \text{such that} \;\; \forall t \in \mathcal{T} \cdot \exists c \in \mathcal{C} \cdot \Phi(t, c),$$

where $\leftrightarrow$ designates a binary relation.

To reason about fault tolerance, we should take into account component unreliability. A failure of a component means that it cannot perform its associated task. Fault tolerance mechanisms employed to mitigate results of component failures rely on various forms of component redundancy. Spacecraft have stringent limitations on the size and weight of the on-board equipment, hence high degree of redundancy is rarely present. Typically, components are either duplicated or triplicated. Let us consider a duplicated system that consists of two identical DPUs – DPU_A and DPU_B. As it was explained above, each DPU contains four components responsible for controlling the corresponding sensor.

Traditionally, satellite systems are designed to implement the following simple redundancy scheme. Initially DPU_A is active, while DPU_B is a cold spare. DPU_A allocates tasks on its components to achieve the system goal $\mathcal{G}$ – processing of a TC and producing the TM. When some component of DPU_A fails, DPU_B is activated to achieve the goal $\mathcal{G}$. Failure of DPU_B results in failure of the overall system. However, even though none of the DPUs can accomplish $\mathcal{G}$ on its own, it might be the case that the operational components of both DPUs can together perform the entire set of tasks required to reach $\mathcal{G}$. This observation allows us to define the following dynamic reconfiguration strategy.

Initially DPU_A is active and assigned to reach the goal $\mathcal{G}$. If some of its components fails, resulting in a failure to execute one of four scientific tasks (let it be $task_j$), the spare DPU_B is activated and DPU_A is deactivated. DPU_B performs the $task_j$ and the consecutive tasks required to reach $\mathcal{G}$. It becomes fully responsible for achieving the goal $\mathcal{G}$ until some of its component fails. In this case, to remain operational, the system performs *dynamic reconfiguration*. Specifically, it reactivates DPU_A and tries to assign the failed task to its corresponding component. If such a component is operational then DPU_A continues to execute the subsequent tasks until it encounters a failed component. Then the control is passed to DPU_B again. Obviously, the overall system stays operational until two identical components of both DPUs have failed.

We generalise the architecture of DPU by stating that essentially a system consists of a number of modules and each module consists of n components:

$$\mathcal{C} = \mathcal{C}_a \cup \mathcal{C}_b, \quad \text{where} \quad \mathcal{C}_a = \{a\_comp_j \mid j \in 1..n \wedge n \in \mathbb{N}_1\} \quad \text{etc.}$$

Each module relies on its components to achieve the tasks required to accomplish $\mathcal{G}$. An introduction of redundancy allows us to associate not a single but several components with each task. We reformulate the goal reachability property as follows: a goal remains reachable while there exists at least one *operational* component associated with each task. Formally, it can be specified as:

$$\mathcal{M} \models \Box \mathcal{O}_s, \quad \text{where} \quad \mathcal{O}_s \equiv \forall t \in \mathcal{T} \cdot (\exists c \in \mathcal{C} \cdot \Phi(t, c) \wedge \mathcal{O}(c))$$

and $\mathcal{O}$ is a predicate over the set of components $\mathcal{C}$ such that $\mathcal{O}(c)$ evaluates to $TRUE$ if and only if the component c is operational.

2.3 Probabilistic Assessment

If a duplicated system with the dynamic reconfiguration achieves the desired reliability level, it might allow the designers to avoid module triplication. However, it also increases the amount of intercomponent communication that leads to decreasing the system performance. Hence, while deciding on a fault tolerance strategy, it is important to consider not only reachability of functional goals but also their performance and reliability aspects.

In engineering, reliability is usually measured by the probability that the system remains operational under given conditions for a certain time interval. In terms of goal reachability, the system remains operational until it is capable of

reaching targeted goals. Hence, to guarantee that system is capable of performing a required functions within a time interval t, it is enough to verify that

$$\mathcal{M} \models \Box^{\leq t}\, \mathcal{O}_s. \tag{1}$$

However, due to possible component failures we usually cannot guarantee the absolute preservation of (1). Instead, to assess the reliability of a system, we need to show that the probability of preserving the property (1) is sufficiently high. On the other hand, the system performance is a reward-based property that can be measured by the number of successfully achieved goals within a certain time period.

To quantitatively verify these quality attributes we formulate the following CSL (Continuous Stochastic Logic) formulas [6]:

$$\mathbf{P}_{=?}\{\mathbf{G} \leq t\, \mathcal{O}_s\} \quad \text{and} \quad \mathbf{R}(|goals|)_{=?}\{\mathbf{C} \leq t\,\}.$$

The formulas above are specified using PRISM notation. The operator **P** is used to refer to the probability of an event occurrence, **G** is an analogue of $\Box$, **R** is used to analyse the *expected values* of rewards specified in a model, while **C** specifies that the reward should be cumulated only up to a given time bound. Thus, the first formula is used to analyse how likely the system remains operational as time passes, while the second one is used to compute the expected number of achieved goals cumulated by the system over t time units.

In this paper we rely on modelling in Event-B to formally define the architecture of a dynamically reconfigurable system, and on the probabilistic extension of Event-B to create models for assessing system reliability and performance. The next section briefly describes Event-B and its probabilistic extension.

3 Modelling in Event-B and Probabilistic Analysis

3.1 Modelling and Refinement in Event-B

Event-B is a state-based formal approach that promotes the correct-by-construction development paradigm and formal verification by theorem proving. In Event-B, a system model is specified using the notion of an *abstract state machine* [1], which encapsulates the model state, represented as a collection of variables, and defines operations on the state, i.e., it describes the behaviour of a modelled system. Usually, a machine has an accompanying component, called *context*, which includes user-defined sets, constants and their properties given as a list of model axioms. The model variables are strongly typed by the constraining predicates. These predicates and the other important properties that must be preserved by the model constitute model *invariants*.

The dynamic behaviour of the system is defined by a set of atomic *events*. Generally, an event has the following form:

$$e \mathrel{\widehat{=}} \textbf{any}\ a\ \textbf{where}\ G_e\ \textbf{then}\ R_e\ \textbf{end},$$

where e is the event's name, a is the list of local variables, the *guard* G_e is a predicate over the local variables of the event and the state variables of the

system. The body of the event is defined by the next-state relation R_e. In Event-B, R_e is defined by a *multiple* (possibly nondeterministic) assignment over the system variables. The guard defines the conditions under which the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution nondeterministically.

Event-B employs a top-down refinement-based approach to system development. Development starts from an abstract specification that nondeterministically models the most essential functional requirements. In a sequence of refinement steps we gradually reduce nondeterminism and introduce detailed design decisions. In particular, we can add new events, split events as well as replace abstract variables by their concrete counterparts, i.e., perform *data refinement*. When data refinement is performed, we should define *gluing invariants* as a part of the invariants of the refined machine. They define the relationship between the abstract and concrete variables. The proof of data refinement is often supported by supplying *witnesses* – the concrete values for the replaced abstract variables and parameters. Witnesses are specified in the event clause **with**.

The consistency of Event-B models, i.e., verification of well-formedness and invariant preservation as well as correctness of refinement steps, is demonstrated by discharging the relevant proof obligations generated by the Rodin platform [11]. The platform provides an automated tool support for proving.

3.2 Augmenting Event-B Models with Probabilities

Next we briefly describe the idea behind translating of an Event-B machine into continuous time Markov chain – CTMC (the details can be found in [15]). To achieve this, we augment all events of the machine with information about the probability and duration of all the actions that may occur during their execution, and refine them by their probabilistic counterparts.

Let Σ be a state space of an Event-B model defined by all possible values of the system variables. Let also $\mathcal{I}$ be the model invariant. We consider an event e as a binary relation on Σ, i.e., for any two states $\sigma, \sigma' \in \Sigma$:

$$e(\sigma, \sigma') \stackrel{def}{=} G_e(\sigma) \wedge R_e(\sigma, \sigma').$$

Definition 1. *The behaviour of an Event-B machine is fully defined by a transition relation* $\rightarrow$*:*

$$\frac{\sigma, \sigma' \in \Sigma \ \wedge \ \sigma' \in \bigcup\limits_{e \in \mathcal{E}_\sigma} \mathsf{after}(e)}{\sigma \rightarrow \sigma'},$$

where $\mathsf{before}(e) = \{\sigma \in \Sigma \mid \mathcal{I}(\sigma) \wedge G_e(\sigma)\}$, $\mathcal{E}_\sigma = \{e \in \mathcal{E} \mid \sigma \in \mathsf{before}(e)\}$ *and* $\mathsf{after}(e) = \{\sigma' \in \Sigma \mid \mathcal{I}(\sigma') \wedge (\exists \sigma \in \Sigma \cdot \mathcal{I}(\sigma) \wedge G_e(\sigma) \wedge R_e(\sigma, \sigma'))\}$.

Furthermore, let us denote by $\lambda_e(\sigma, \sigma')$ the (exponential) transition rate from σ to σ' via the event e, where $\sigma \in \mathsf{before}(e)$ and $R_e(\sigma, \sigma')$. By augmenting all the event actions with transition rates, we can modify Definition 1 as follows.

Definition 2. *The behaviour of a probabilistically augmented Event-B machine is defined by a transition relation* $\xrightarrow{\Lambda}$*:*

$$\frac{\sigma, \sigma' \in \Sigma \ \wedge \ \sigma' \in \bigcup\limits_{e \in \mathcal{E}_\sigma} \mathsf{after}(e)}{\sigma \xrightarrow{\Lambda} \sigma'}, \quad \textit{where } \Lambda = \sum_{e \in \mathcal{E}_\sigma} \lambda_e(\sigma, \sigma').$$

Definition 2 allows us to define the semantics of a probabilistically augmented Event-B model as a probabilistic transition system with the state space Σ, transition relation $\xrightarrow{\Lambda}$ and the initial state defined by model initialisation (for probabilistic models we require the initialisation to be deterministic). Clearly, such a transition system corresponds to a CTMC.

In the next section we demonstrate how to formally derive an Event-B model of the architecture of a reconfigurable system.

4 Deriving Fault Tolerant Architectures by Refinement in Event-B

The general idea behind our formal development is to start from an abstract goal modelling, decompose it into tasks and introduce an abstract representation of the goal execution flow. Such a model can be refined into different fault tolerant architectures. Subsequently, these models are augmented with probabilistic data and used for the quantitative assessment.

4.1 Modelling Goal Reaching

Goal Modelling. Our initial specification abstractly models the process of reaching the goal. The progress of achieving the goal is modelled by the variable *goal* that obtains values from the enumerated set $STATUS = \{not\_reached, reached, failed\}$. Initially, the system is not assigned any goals to accomplish, i.e., the variable *idle* is equal to $TRUE$. When the system becomes engaged in establishing the goal, *idle* obtains value $FALSE$ as modelled by the event *Activation*. In the process of accomplishing the goal, the variable *goal* might eventually change its value from *not_reached* to *reached* or $failed$, as modelled by the event *Body*. After the goal is reached the system becomes idle, i.e., a new goal can be assigned. The event $Finish$ defines such a behaviour. We treat the failure to achieve the goal as a permanent system failure. It is represented by the infinite stuttering defined in the event *Abort*.

Activation $\widehat{=}$
when $idle = TRUE$
then $idle := FALSE$
end

Body $\widehat{=}$
when $idle = FALSE \wedge goal = not\_reached$
then $goal :\in STATUS$
end

Finish $\widehat{=}$
when $idle = FALSE \wedge goal = reached$
then $goal, idle := not\_reached, TRUE$
end

Abort $\widehat{=}$
when $goal = failed$
then $skip$
end

Goal Decomposition. The aim of our first refinement step is to define the goal execution flow. We assume that the goal is decomposed into n tasks, and can be achieved by a sequential execution of one task after another. We also assume that the id of each task is defined by its execution order. Initially, when the goal is assigned, none of the tasks is executed, i.e., the state of each task is "not defined" (designated by the constant value ND). After the execution, the state of a task might be changed to success or failure, represented by the constants OK and NOK correspondingly. Our refinement step is essentially data refinement that replaces the abstract variable *goal* with the new variable *task* that maps the id of a task to its state, i.e., $task \in 1..n \to \{OK, NOK, ND\}$.

We omit showing the events of the refined model (the complete development can be found in [13]). They represent the process of sequential selection of one task after another until either all tasks are executed, i.e., the goal is reached, or execution of some task fails, i.e., goal is not achieved. Correspondingly, the guards ensure that either the goal reaching has not commenced yet or the execution of all previous task has been successful. The body of the events nondeterministically changes the state of the chosen task to OK or NOK. The following invariants define the properties of the task execution flow:

$$\forall l \cdot l \in 2 \mathinner{..} n \wedge task(l) \neq ND \Rightarrow (\forall i \cdot i \in 1 \mathinner{..} l-1 \Rightarrow task(i) = OK),$$
$$\forall l \cdot l \in 1 \mathinner{..} n-1 \wedge task(l) \neq OK \Rightarrow (\forall i \cdot i \in l+1 \mathinner{..} n \Rightarrow task(i) = ND).$$

They state that the goal execution can progress, i.e., a next task can be chosen for execution, only if none of the previously executed tasks failed and the subsequent tasks have not been executed yet.

From the requirements perspective, the refined model should guarantee that the system level goal remains achievable. This is ensured by the gluing invariants that establish the relationship between the abstract goal and the tasks:

$$task[1 \mathinner{..} n] = \{OK\} \Rightarrow goal = reached,$$
$$(task[1 \mathinner{..} n] = \{OK, ND\} \vee task[1 \mathinner{..} n] = \{ND\}) \Rightarrow goal = not\_reached,$$
$$(\exists i \cdot i \in 1 \mathinner{..} n \wedge task(i) = NOK) \Rightarrow goal = failed.$$

Introducing Abstract Communication. In the second refinement step we introduce an abstract model of communication. We define a new variable ct that stores the id of the last achieved task. The value of ct is checked every time when a new task is to be chosen for execution. If task execution succeeds then ct is incremented. Failure to execute the task leaves ct unchanged and results only in the change of the failed task status to NOK. Essentially, the refined model introduces an abstract communication via shared memory. The following gluing invariants allow us to prove the refinement:

$$ct > 0 \Rightarrow (\forall i \cdot i \in 1 \mathinner{..} ct \Rightarrow task(i) = OK), \quad ct < n \Rightarrow task(ct+1) \in \{ND, NOK\},$$
$$ct < n-1 \Rightarrow (\forall i \cdot i \in ct+2 \mathinner{..} n \Rightarrow task(i) = ND).$$

As discussed in Section 2, each task is independently executed by a separate component of a high-level module. Hence, by substituting the id of a task with the id of the corresponding component, i.e., performing a data refinement with the gluing invariant

$$\forall i \in 1..n \cdot task(i) = comp(i),$$

we specify a *non-redundant* system architecture. This invariant trivially defines the relation Φ. Next we demonstrate how to introduce either a triplicated architecture or duplicated architecture with a dynamic reconfiguration by refinement.

4.2 Reconfiguration Strategies

To define triplicated architecture with static reconfiguration, we define three identical modules A, B and C. Each module consists of n components executing corresponding tasks. We refine the abstract variable $task$ by the three new variables a_comp, b_comp and c_comp:

$$a\_comp \in 1..n \to STATE,\ b\_comp \in 1..n \to STATE,\ c\_comp \in 1..n \to STATE.$$

To associate the tasks with the components of each module, we formulate a number of gluing invariants that essentially specify the relation Φ. Some of these invariants are shown below:

$$\begin{gathered}
\forall i \cdot i \in 1 \mathinner{..} n \wedge module = A \wedge a_comp(i) = OK \Rightarrow task(i) = OK,\\
module = A \Rightarrow (\forall i \cdot i \in 1 \mathinner{..} n \Rightarrow b_comp(i) = ND \wedge c_comp(i) = ND),\\
\forall i \cdot i \in 1 \mathinner{..} n \wedge module = A \wedge a_comp(i) \neq OK \Rightarrow task(i) = ND,\\
\forall i \cdot i \in 1 \mathinner{..} n \wedge module = B \wedge b_comp(i) \neq OK \Rightarrow task(i) = ND,\\
\forall i \cdot i \in 1 \mathinner{..} n \wedge module = C \Rightarrow c_comp(i) = task(i),\\
module = B \Rightarrow (\forall i \cdot i \in 1 \mathinner{..} n \Rightarrow c_comp(i) = ND).
\end{gathered}$$

Here, a new variable $module \in \{A, B, C\}$ stores the id of the currently active module. The complete list of invariants can be found in [13]. Please note, that these invariants allows us to mathematically prove that the Event-B model preserves the desired system architecture.

An alternative way to perform this refinement step is to introduce a duplicated architecture with dynamic reconfiguration. In this case, we assume that our system consists of two modules, A and B, defined in the same way as discussed above. We replace the abstract variable $task$ with two new variables a_comp and b_comp. Below we give an excerpt from the definition of the gluing invariants:

$$\begin{gathered}
module = A \wedge ct > 0 \wedge a_comp(ct) = OK \Rightarrow task(ct) = OK,\\
module = B \wedge ct > 0 \wedge b_comp(ct) = OK \Rightarrow task(ct) = OK,\\
\forall i \cdot i \in 1 \mathinner{..} n \wedge a_comp(i) = NOK \wedge b_comp(i) = NOK \Rightarrow task(i) = NOK,\\
\forall i \cdot i \in 1 \mathinner{..} n \wedge a_comp(i) = NOK \wedge b_comp(i) = ND \Rightarrow task(i) = ND,\\
\forall i \cdot i \in 1 \mathinner{..} n \wedge b_comp(i) = NOK \wedge a_comp(i) = ND \Rightarrow task(i) = ND.
\end{gathered}$$

Essentially, the invariants define the behavioural patterns for executing the tasks according to dynamic reconfiguration scenario described in Section 2.

Since our goal is to study the fault tolerance aspect of the system architecture, in our Event-B model we have deliberately abstracted away from the representation of the details of the system behaviour. A significant number of functional requirements is formulated as gluing invariants. As a result, to verify correctness of the models we discharged more than 500 proof obligations. Around 90% of them have been proved automatically by the Rodin platform and the rest have been proved manually in the Rodin interactive proving environment.

Note that the described development for a generic system can be easily instantiated to formally derive fault tolerant architectures of DPU. The goal of DPU – handling the scientific TC by producing TM – is decomposed into four tasks that define the production of data by the satellite's sensor units – SIXS-X, SIXS-P, MIXS-T and MIXS-C. Thus, for such a model we have four tasks ($n=4$) and each task is handled by the corresponding computing component of DPU. The high-level modules A, B and C correspond to three identical DPUs that control handling of scientific TC – DPU_A, DPU_B and DPU_C, while functions *a_comp*, *b_comp* and *c_comp* represent statuses of their internal components.

From the functional point of view, both alternatives of the last refinement step are equivalent. Indeed, each of them models the process of reaching the goal by a fault tolerant system architecture. In the next section we will present a quantitative assessment of their reliability and performance aspects.

5 Quantitative Assessment of Reconfiguration Strategies

The scientific mission of BepiColombo on the orbit of the Mercury will last for one year with possibility to extend this period for another year. Therefore, we should assess the reliability of both architectural alternatives for this period of time. Clearly, the triplicated DPU is able to tolerate up to three DPU failures within the two-year period, while the use of a duplicated DPU with a dynamic reconfiguration allows the satellite to tolerate from one (in the worst case) to four (in the best case) failures of the components.

Obviously, the duplicated architecture with a dynamic configuration minimises volume and the weight of the on-board equipment. However, the dynamic reconfiguration requires additional inter-component communication that slows down the process of producing TM. Therefore, we need to carefully analyse the performance aspect as well. Essentially, we need to show that the duplicated system with the dynamic reconfiguration can also provide a sufficient amount of scientific TM within the two-year period.

To perform the probabilistic assessment of reliability and performance, we rely on two types of data:

- probabilistic data about lengths of time delays required by DPU components and sensor units to produce the corresponding parts of scientific data
- data about occurrence rates of possible failures of these components

It is assumed that all time delays are exponentially distributed. We refine the Event-B specifications obtained at the final refinement step by their probabilistic counterparts. This is achieved via introducing probabilistic information into events and replacing all the local nondeterminism with the (exponential) race conditions. Such a refinement relies on the model transformation presented in Section 3. As a result, we represent the behaviour of Event-B machines by CTMCs. This allows us to use the probabilistic symbolic model checker PRISM to evaluate reliability and performance of the proposed models.

Due to the space constraints, we omit showing the PRISM specifications in the paper, they can be found in [13]. The guidelines for Event-B to PRISM model transformation can be found in our previous work [14].

The results of quantitative verification performed by PRISM show that with probabilistic characteristics of DPU presented, in Table 1[1], both reconfiguration strategies lead to a similar level of system reliability and performance with insignificant advantage of the triplicated DPU. Thus, the reliability levels of both systems within the two-year period are approximately the same with the difference of just 0.003 at the end of this period (0.999 against 0.996). Furthermore, the use of two DPUs under dynamic reconfiguration allows the satellite to handle only 2 TCs less after two years of work – 1104 against 1106 returned TM packets in the case of the triplicated DPU. Clearly, the use of the duplicated architecture with dynamic reconfiguration to achieve the desired levels of relia bility and performance is optimal for the considered system.

Table 1. Rates (time is measured by minutes)

TC access rate when the system is idle	λ	$\frac{1}{12 \cdot 60}$	SIXS-P work rate	α_2	$\frac{1}{30}$
TM output rate when a TC is handled	μ	$\frac{1}{20}$	SIXS-P failure rate	β_2	$\frac{1}{10^6}$
Spare DPU activation rate (power on)	δ	$\frac{1}{10}$	MIXS-T work rate	α_3	$\frac{1}{30}$
DPUs "communication" rate	τ	$\frac{1}{5}$	MIXS-T failure rate	β_3	$\frac{1}{9 \cdot 10^7}$
SIXS-X work rate	α_1	$\frac{1}{60}$	MIXS-C work rate	α_4	$\frac{1}{90}$
SIXS-X failure rate	β_1	$\frac{1}{8 \cdot 10^7}$	MIXS-C failure rate	β_4	$\frac{1}{6 \cdot 10^7}$

Finally, let us remark that the goal-oriented style of the reliability and performance analysis has significantly simplified the assessment of the architectural alternatives of DPU. Indeed, it allowed us to abstract away from the configuration of input and output buffers, i.e., to avoid modelling of the circular buffer as a part of the analysis.

6 Conclusions and Related Work

In this paper we proposed a formal approach to development and assessment of fault tolerant satellite systems. We made two main technical contributions. On the one hand, we defined the guidelines for development of the dynamically reconfigurable systems. On the other hand, we demonstrated how to formally assess reconfiguration strategy and evaluate whether the chosen fault tolerance mechanism fulfils reliability and performance objectives. The proposed approach was illustrated by a case study – development and assessment of the reconfigurable DPU. We believe that our approach not only guarantees correct design of complex fault tolerance mechanisms but also facilitates finding suitable trade-offs between reliability and performance.

[1] Provided information may differ form the characteristics of the real components. It is used merely to demonstrate how the required comparison of reliability/performance can be achieved.

A large variety of aspects of the dynamic reconfiguration has been studied in the last decade. For instance, Wermelinger et al. [17] proposed a high-level language for specifying the dynamically reconfigurable architectures. They focus on modifications of the architectural components and model reconfiguration by the algebraic graph rewriting. In contrast, we focused on the functional rather than structural aspect of reasoning about reconfiguration.

Significant research efforts are invested in finding suitable models of triggers for run-time adaptation. Such triggers monitor performance [3] or integrity [16] of the application and initiate reconfiguration when the desired characteristics are not achieved. In our work we perform the assessment of reconfiguration strategy at the development phase that allows us to rely on existing error detection mechanisms to trigger dynamic reconfiguration.

A number of researchers investigate self* techniques for designing adaptive systems that autonomously achieve fault tolerance, e.g., see [4,10]. However, these approaches are characterised by a high degree of uncertainty in achieving fault tolerance that is unsuitable for the satellite systems. The work [5] proposes an interesting conceptual framework for establishing a link between changing environmental conditions, requirements and system-level goals. In our approach we were more interested in studying a formal aspect of dynamic reconfiguration.

In our future work we are planning to further study the properties of dynamic reconfiguration. It particular, it would be interesting to investigate reconfiguration in the presence of parallelism and complex component interdependencies.

References

1. Abrial, J.-R.: Modeling in Event-B. Cambridge University Press (2010)
2. BepiColombo: ESA Media Center, Space Science, http://www.esa.int/esaSC/SEMNEM3MDAF_0_spk.html
3. Caporuscio, M., Di Marco, A., Inverardi, P.: Model-Based System Reconfiguration for Dynamic Performance Management. J. Syst. Softw. 80, 455–473 (2007)
4. de Castro Guerra, P.A., Rubira, C.M.F., de Lemos, R.: A Fault-Tolerant Software Architecture for Component-Based Systems. In: Architecting Dependable Systems, pp. 129–143. Springer (2003)
5. Goldsby, H.J., Sawyer, P., Bencomo, N., Cheng, B., Hughes, D.: Goal-Based Modeling of Dynamically Adaptive System Requirements. In: ECBS 2008, pp. 36–45. IEEE Computer Society (2008)
6. Grunske, L.: Specification Patterns for Probabilistic Quality Properties. In: ICSE 2008, pp. 31–40. ACM (2008)
7. Kelly, T.P., Weaver, R.A.: The Goal Structuring Notation – A Safety Argument Notation. In: DSN 2004, Workshop on Assurance Cases (2004)
8. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of Probabilistic Real-Time Systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
9. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: RE 2001, pp. 249–263. IEEE Computer Society (2001)
10. de Lemos, R., de Castro Guerra, P.A., Rubira, C.M.F.: A Fault-Tolerant Architectural Approach for Dependable Systems. IEEE Software 23, 80–87 (2006)
11. Rodin: Event-B Platform, http://www.event-b.org/

12. Space Engineering: Ground Systems and Operations – Telemetry and Telecommand Packet Utilization: ECSS-E-70-41A. ECSS Secretariat (January 30, 2003), http://www.ecss.nl/
13. Tarasyuk, A., Pereverzeva, I., Troubitsyna, E., Latvala, T., Nummila, L.: Formal Development and Assessment of a Reconfigurable On-board Satellite System. Tech. Rep. 1038, Turku Centre for Computer Science (2012)
14. Tarasyuk, A., Troubitsyna, E., Laibinis, L.: Quantitative Reasoning about Dependability in Event-B: Probabilistic Model Checking Approach. In: Dependability and Computer Engineering: Concepts for Software-Intensive Systems, pp. 459–472. IGI Global (2011)
15. Tarasyuk, A., Troubitsyna, E., Laibinis, L.: Formal Modelling and Verification of Service-Oriented Systems in Probabilistic Event-B. In: Derrick, J., Gnesi, S., Latella, D., Treharne, H. (eds.) IFM 2012. LNCS, vol. 7321, pp. 237–252. Springer, Heidelberg (2012)
16. Warren, I., Sun, J., Krishnamohan, S., Weerasinghe, T.: An Automated Formal Approach to Managing Dynamic Reconfiguration. In: ASE 2006, pp. 18–22. Springer (2006)
17. Wermelinger, M., Lopes, A., Fiadeiro, J.: A Graph Based Architectural Reconfiguration Language. SIGSOFT Softw. Eng. Notes 26, 21–32 (2001)

Paper V

Quantitative Verification of System Safety in Event-B

Anton Tarasyuk, Elena Troubitsyna and Linas Laibinis

Originally published in: Elena Troubitsyna (Ed.), *Proceedings of 3rd International Workshop on Software Engineering for Resilient Systems (SERENE 2011)*, LNCS 6968, 24–39, Springer, 2011

Quantitative erification of System Safety in Event-B

Anton Tarasyuk[1,2], Elena Troubitsyna[2], and Linas Laibinis[2]

[1] Turku Centre for Computer Science
[2] Åbo Akademi University
Joukahaisenkatu 3-5, 20520 Turku, Finland
{anton.tarasyuk,elena.troubitsyna,linas.laibinis}@abo.fi

Abstract. Certification of safety-critical systems requires formal verification of system properties and behaviour as well as quantitative demonstration of safety. Usually, formal modelling frameworks do not include quantitative assessment of safety. This has a negative impact on productivity and predictability of system development. In this paper we present an approach to integrating quantitative safety analysis into formal system modelling and verification in Event-B. The proposed approach is based on an e tension of Event-B, which allows us to perform quantitative assessment of safety within proof-based verification of system behaviour. This enables development of systems that are not only correct but also safe by construction. The approach is demonstrated by a case study – an automatic railway crossing system.

1 Introduction

Safety is a property of a system to not endanger human life or environment [4]. To guarantee safety, designers employ various rigorous techniques for formal modelling and verification. Such techniques facilitate formal reasoning about system correctness. In particular, they allow us to guarantee that a safety invariant – a logical representation of safety – is always preserved during system execution. However, real safety-critical systems, i.e., the systems whose components are susceptible to various kinds of faults, are not "absolutely" safe. In other words, certain combinations of failures may lead to an occurrence of a hazard – a potentially dangerous situation breaching safety requirements. While designing and certifying safety-critical systems, we should demonstrate that the probability of a hazard occurrence is acceptably low. In this paper we propose an approach to combining formal system modelling and quantitative safety analysis.

Our approach is based on a probabilistic extension of Event-B [22]. Event-B is a formal modelling framework for developing systems correct-by-construction [3,1]. It is actively used in the EU project Deploy [6] for modelling and verifying of complex systems from various domains including railways. The Rodin platform [20] provides the designers with an automated tool support that facilitates formal verification and makes Event-B relevant in an industrial setting.

E.A. r bits na (Ed.): SE ENE 2011, LNCS 6 6 , pp. 2 –3 , 2011.

The main development technique of Event-B is refinement – a top-down process of gradual unfolding of the system structure and elaborating on its functionality. In this paper we propose design strategies that allow the developers to structure safety requirements according to the system abstraction layers. Essentially, such an approach can be seen as a process of extracting a fault tree – a logical representation of a hazardous situation in terms of the primitives used at different abstraction layers. Eventually, we arrive at the representation of a hazard in terms of failures of basic system components. Since our model explicitly contains probabilities of component failures, standard calculations allow us to obtain a probabilistic evaluation of a hazard occurrence. As a result, we obtain an algebraic representation of the probability of safety violation. This probability is defined using the probabilities of system component failures. To illustrate our approach, we present a formal development and safety analysis of a radio-based railway crossing. We believe the proposed approach can potentially facilitate development, verification and assessment of safety-critical systems.

The rest of the paper is organised as follows. In Section 2 we describe our formal modelling framework – Event-B, and briefly introduce its probabilistic extension. In Section 3 we discuss a general design strategy for specifying Event-B models amenable for probabilistic analysis of system safety. In Section 4 we demonstrate the presented approach by a case study. Finally, Section 5 presents an overview of the related work and some concluding remarks.

2 Modelling in Event-B

The B Method [2] is an approach for the industrial development of highly dependable software. The method has been successfully used in the development of several complex real-life applications [19,5]. Event-B is a formal framework derived from the B Method to model parallel, distributed and reactive systems. The Rodin platform provides automated tool support for modelling and verification in Event-B. Currently Event-B is used in the EU project Deploy to model several industrial systems from automotive, railway, space and business domains.

Event-B anguage and Semantics. In Event-B, a system model is defined using the notion of an *abstract state machine* [18]. An abstract state machine encapsulates the model state, represented as a collection of model variables, and defines operations on this state. Therefore, it describes the dynamic part of the modelled system. A machine may also have an accompanying component, called *context*, which contains the static part of the system. In particular, It can include user-defined carrier sets, constants and their properties given as a list of model axioms. A general form of Event-B models is given in Fig. 1.

The machine is uniquely identified by its name M. The state variables, v, are declared in the **a riables** clause and initialised in the $Init$ event. The variables are strongly typed by the constraining predicates I given in the **Invariants** clause. The invariant clause also contains other predicates defining properties that must be preserved during system execution.

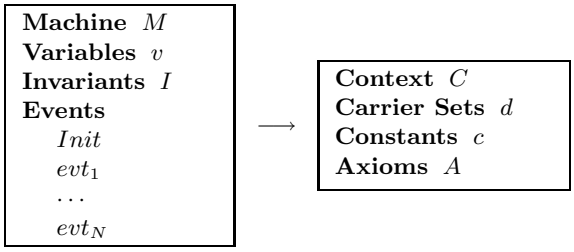


Fig. 1. Event-B machine and conte t

Action (S)	$BA(S)$
$x := E(x,y)$	$x' = E(x,y) \ \wedge \ y' = y$
$x :\in Set$	$\exists z \cdot (z \in Set \wedge x' = z) \ \wedge \ y' = y$
$x :\mid Q(x,y,x')$	$\exists z \cdot (Q(x,z,y) \wedge x' = z) \ \wedge \ y' = y$

Fig. 2. Before-after predicates

The dynamic behaviour of the system is defined by the set of atomic events specified in the **Events** clause. Generally, an event can be defined as follows:

$$evt \mathrel{\widehat{=}} \textbf{any } a \textbf{ where } g \textbf{ then } S \textbf{ end},$$

where a is the list of local variables, the guard g is a conjunction of predicates over the local variables a and state variables v, while the action S is a state assignment. If the list a is empty, an event can be described simply as

$$evt \mathrel{\widehat{=}} \textbf{ when } g \textbf{ then } S \textbf{ end}.$$

The occurrence of events represents the observable behaviour of the system. The guard defines the conditions under which the action can be executed, i.e., when the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution nondeterministically. If none of the events is enabled then the system deadlocks.

In general, the action of an event is a parallel composition of variable assignments. The assignments can be either deterministic or non-deterministic. A deterministic assignment, $x := E(x,y)$, has the standard syntax and meaning. A nondeterministic assignment is denoted either as $x :\in Set$, where Set is a set of values, or $x :\mid Q(x,y,x')$, where Q is a predicate relating initial values of x, y to some final value of x'. As a result of such a non-deterministic assignment, x can get any value belonging to Set or satisfying Q.

The semantics of Event-B actions is defined using so-called before-after (BA) predicates [3,18]. A BA predicate describes a relationship between the system states before and after execution of an event, as shown in Fig. 2. Here x and y are disjoint lists of state variables, and x', y' represent their values in the after-state. The semantics of a whole Event-B model is formulated as a number of *proof obligations*, expressed in the form of logical sequents. The full list of proof obligations can be found in [3].

Probabilistic Event-B. In our previous work [22] we have have extended the Event-B modelling language with a new operator – *quantitative probabilistic choice*, denoted $\oplus|$. It has the following syntax

$$x \oplus| \; x_1 \quad p_1 \; \ldots \; x_n \quad p_n,$$

where $\sum_{i\ 1}^{n} p_i = 1$. It assigns to the variable x a new value x_i with the corresponding non-zero probability p_i. The quantitative probabilistic choice (assignment) allows us to precisely represent the probabilistic information about how likely a particular choice is made. In other words, it behaves according to some known probabilistic distribution.

We have restricted the use of the new probabilistic choice operator by introducing it only to replace the existing demonic one. This approach has also been adopted by Hallerstede and Hoang, who have proposed extending the Event-B framework with *qualitative probabilistic choice* [10]. It has been shown that any probabilistic choice statement always refines its demonic nondeterministic counterpart [13]. Hence, such an extension is not interfering with the established refinement process. Therefore, we can rely on the Event-B proof obligations to guarantee functional correctness of a refinement step. Moreover, the probabilistic information introduced in new quantitative probabilistic choices can be used to stochastically evaluate certain non-functional system properties.

For instance, in [22] we have shown how the notion of Event-B refinement can be strengthened to quantitatively demonstrate that the refined system is more reliable than its abstract counterpart. In this paper we aim at enabling quantitative safety analysis within Event-B development.

3 Safety Analysis in Event-B

In this paper we focus on modelling of highly dynamic reactive control systems. Such systems provide instant control actions as a result of receiving stimuli from the controlled environment. Such a restriction prevents the system from executing automated error recovery, i.e. once a component fails, its failure is considered to be permanent and the system ceases its automatic functioning.

Generally, control systems are built in a layered fashion and reasoning about their behaviour is conducted by unfolding layers of abstraction. Deductive safety analysis is performed in a similar way. We start by identifying a hazard – a dangerous situation associated with the system. By unfolding the layers of abstraction we formulate the hazard in terms of component states of different layers.

In an Event-B model, a hazard can be naturally defined as a predicate over the system variables. Sometimes, it is more convenient to reformulate a hazard as a dual safety requirement (property) that specifies a proper behaviour of a system in a hazardous situation. The general form of such a safety property is:

$$SAF \mathrel{\widehat{=}} H(v) \Rightarrow K(v),$$

where the predicate $H(v)$ specifies a hazardous situation and the predicate $K(v)$ defines the safety requirements in the terms of the system variables and states.

The essential properties of an Event-B model are usually formulated as invariants. However, to represent system behaviour realistically, our specification should include modelling of not only normal behaviour but also component failure occurrence. Since certain combinations of failures will lead to hazardous situations, we cannot guarantee "absolute" preservation of safety invariants. Indeed, the goal of development of safety-critical systems is to guarantee that the probability of violation of safety requirements is sufficiently small.

To assess the preservation of a desired safety property, we will unfold it (in the refinement process) until it refers only to concrete system components that have direct impact on the system safety. To quantitatively evaluate this impact, we require that these components are probabilistically modelled in Event-B using the available information about their reliability. Next we demonstrate how the process of unfolding the safety property from the abstract to the required concrete representation can be integrated into our formal system development.

Often, functioning of a system can be structured according to a number of *execution stages*. There is a specific component functionality associated with each stage. Since there is no possibility to replace or repair failed system components, we can divide the process of quantitative safety assessment into several consecutive steps, where each step corresponds to a particular stage of the system functioning. Moreover, a relationship between different failures of components and the system behaviour at a certain execution stage is preserved during all the subsequent stages. On the other hand, different subsystems can communicate with each other, which leads to possible additional dependencies between system failures (not necessarily within the same execution stage). This fact significantly complicates quantitative evaluation of the system safety.

We can unfold system safety properties either in a *backward* or in a *forward* way. In the backward unfolding we start from the last execution stage preceding the stage associated with the potentially hazardous situation. In the forward one we start from the first execution stage of the system and continue until the last stage just before the hazardous situation occurs. In this paper we follow the former approach. The main idea is to perform a stepwise analysis of any possible behaviour of all the subsystems at every execution stage preceding the hazardous situation, while gradually unfolding the abstract safety property in terms of new (concrete) variables representing faulty components of the system.

Specifically, in each refinement step, we have to establish the relationship between the newly introduced variables and the abstract variables present in the safety property. A standard way to achieve this is to formulate the required relationship as a number of safety invariants in Event-B. According to our development strategy, each such invariant establishes a connection between abstract and more concrete variables that have an impact on system safety. Moreover, the preservation of a safety invariant is usually verified for a particular subsystem at a specific stage. Therefore, we can define a general form of such an invariant:

$$I_s(v,u) \mathrel{\widehat{=}} F(v) \Rightarrow (K(v) \Leftrightarrow L(u)),$$

where the predicate F restricts the execution stage and the subsystems involved, while the predicate $K \Leftrightarrow L$ relates the values of the newly introduced variables

u with the values the abstract variables v present in the initially defined safety property or/and in the safety invariants defined in the previous refinement steps.

To calculate the probability of preservation of the safety property, the refinement process should be continued until all the abstract variables, used in the definition of the system safety property, are related to the concrete, probabilistically updated variables, representing various system failures or malfunctioning. The process of probability evaluation is rather straightforward and based on basic definitions and rules for calculating probabilities (see [7] for instance).

Let us consider a small yet generic example illustrating the calculation of probability using Event-B safety invariants. We assume that the safety property SAF is defined as above. In addition, let us define two safety invariants – I_s and J_s – introduced in two subsequent refinement steps. More specifically,

$$I_s \mathrel{\widehat{=}} F \Rightarrow (K(v) \Leftrightarrow L_1(u_1) \vee L_2(u_2)) \text{ and } J_s \mathrel{\widehat{=}} F \Rightarrow (L_2(u_2) \Leftrightarrow N(w)),$$

where $u_1 \subset u, u_1 \neq \emptyset$ are updated probabilistically in the first refinement, while $u_2 = u \setminus u_1$ are still abstract in the first refinement machine and related by J_s to the probabilistically updated variables w in the following one. Let us note that the predicate F must define the earlier stage of the system than the predicate F does. Then the probability that the safety property SAF is preserved is

$$P_{SAF} = P\{K(v)\} = P\{L_1(u_1) \vee L_2(u_2)\} = P\{L_1(u_1) \vee N(w)\} = \\ P\{L_1(u_1)\} + P\{N(w)\} - P\{L_1(u_1) \wedge N(w)\},$$

where

$$P\{L_1(u_1) \wedge N(w)\} = P\{L_1(u_1)\} \cdot P\{N(w)\}$$

in the case of independent L_1 and N, and

$$P\{L_1(u_1) \wedge N(w)\} = P\{L_1(u_1)\} \cdot P\{N(w) \mid L_1(u_1)\}$$

otherwise. Note that the predicate $H(v)$ is not participating in the calculation of P_{SAF} directly. Instead, it defines "the time and the place" when and where the values of the variables u and v should be considered, and, as long as it specifies the hazardous situation following the stages defined by F and F, it can be understood as the *post-state* for all the probabilistic events.

In the next section we will demonstrate the approach presented above by a case study – an automatic railway crossing system.

4 Case Study

To illustrate safety analysis in the probabilistically enriched Event-B method, in this section we present a quantitative safety analysis of a radio-based railway crossing. This case study is included into priority program 1064 of the German Research Council (DFG) prepared in cooperation with Deutsche Bahn AG. The main difference between the proposed technology and traditional control systems of railway crossings is that signals and sensors on the route are replaced by radio communication and software computations performed at the train and railway crossings. Formal system modelling of such a system has been undertaken previously [16,15]. However, the presented methodology is focused on logical (qualitative) reasoning about safety and does not include quantitative

safety analysis. Below we demonstrate how to integrate formal modelling and probabilistic safety analysis.

Let us now briefly describe the functioning of a radio-based railway crossing system. The train on the route continuously computes its position. When it approaches a crossing, it broadcasts a *close* request to the crossing. When the railway crossing receives the command *close*, it performs some routine control to ensure safe train passage. It includes switching on the traffic lights, that is followed by an attempt to close the barriers. Shortly before the train reaches the *latest braking point*, i.e., the latest point where it is still possible for the train to stop safely, it requests the *status* of the railway crossing. If the crossing is secured, it responds with a *release* signal, which indicates that the train may pass the crossing. Otherwise, the train has to brake and stop before the crossing. More detailed requirements can be found in [16] for instance.

In our development we abstract away from modelling train movement, calculating train positions and routine control by the railway crossing. Let us note that, any time when the train approaches to the railway crossing, it sequentially performs a number of predefined operations:

- it sends the *close* request to the crossing controller
- after a delay it sends the *status* request
- it awaits for an answer from the crossing controller.

The crossing controller, upon receiving the close request, tries to close the barriers and, if successful, sends the *release* signal to the train. Otherwise, it does not send any signal and in this case the train activates the emergency brakes. Our safety analysis focused on defining the hazardous events that may happen in such a railway crossing system due to different hardware and/or communication failures, and assess the probability of the hazard occurrences. We make the following fault assumptions:

- the radio communication is unreliable and can cause messages to be lost
- the crossing barrier motors may fail to start
- the positioning sensors that are used by the crossing controller to determine a physical position of the barriers are unreliable
- the train emergency brakes may fail.

The abstract model. We start our development with identification of all the high-level subsystems we have to model. Essentially, our system consists of two main components – the train and the crossing controller. The system environment is represented by the physical position of the train. Therefore, each control cycle consists of three main phases – Env, $Train$ and $Crossing$. To indicate the current *phase*, the eponymous variable is used.

The type modelling abstract train positions is defined as the enumerated set of nonnegative integers $POS\_SET = \{0, CRP, SRP, SRS, DS\}$, where $0 < CRP < SRP < SRS < DS$. Each value of POS_SET represents a specific position of the train. Here 0 stands for some initial train position outside the communication area, CRP and SRP stand for the close and status request points, and SRS and DS represent the safe reaction and danger spots respectively. The actual train position is modelled by the variable $train\_pos \in POS\_SET$. In

Machine $RailwayCrossing$
Variables $train\_pos, phase, emrg\_brakes, bar_1, bar_2$
Invariants $\cdots$
Events $\cdots$
$UpdatePosition_1 \mathrel{\widehat{=}}$
when $phase = Env \wedge train\_pos < DS \wedge emrg\_brakes = FALSE$
then
$train\_pos := \min(\{p \mid p \in POS\_SET \wedge p > train\_pos\}) \,||\, phase := Train$
end
$UpdatePosition_2 \mathrel{\widehat{=}}$
when $phase = Env \wedge$
$((train\_pos = DS \wedge emrg\_brakes = FALSE) \vee emrg\_brakes = TRUE)$
then
$skip$
end
$TrainIdle \mathrel{\widehat{=}}$
when $phase = Train \wedge train\_pos \neq SRS$
then
$phase := Crossing$
end
$TrainReact \mathrel{\widehat{=}}$
when $phase = Train \wedge train\_pos = SRS$
then
$emrg\_brakes :\in BOOL \,||\, phase := Crossing$
end
$CrossingBars \mathrel{\widehat{=}}$
when $phase = Crossing \wedge train\_pos = CRP$
then
$bar_1, bar_2 :\in BAR\_POS \,||\, phase := Env$
end
$CrossingIdle \mathrel{\widehat{=}}$
when $phase = Crossing \wedge train\_pos \neq CRP$
then
$phase := Env$
end

Fig. 3. Railway crossing: the abstract machine

addition, we use the boolean variable *emrg_brakes* to model the status of the train emergency brakes. We assume that initially they are not triggered, i.e., $emrg\_brakes = FALSE$.

The crossing has two barriers – one at each side of the crossing. The status of the barriers is modelled by the variables bar_1 and bar_2 that can take values *Opened* and *Closed*. We assume that both barriers are initially open.

The initial abstract machine *RailwayCrossing* is presented in Fig. 3. We omit showing here the *Initialisation* event and the **Invariants** clause (it merely defines the types of variables). Due to lack of space, in the rest of the section we will also present only some selected excerpts of the model. The full Event-B specifications of the *Railway crossing system* can be found in [21].

In *RailwayCrossing* we consider only the basic functionality of the system. Two events $UpdatePosition_1$ and $UpdatePosition_2$ are used to abstractly model train movement. The first event models the train movement outside the danger spot by updating the train abstract position according to the next value of the POS_SET. $UpdatePosition_2$ models the train behaviour after it has passed the last braking point or when it has stopped in the safe reaction spot. Essentially, this event represents the system termination (both safe and unsafe cases), which is modelled as infinite stuttering. Such an approach for modelling of the train movement is sufficient since we only analyse system behaviour within the

train-crossing communication area, i.e., the area that consists of the close and status request points, and the safe reaction spot. A more realistic approach for modelling of the train movement is out of the scope of our safety analysis.

For the crossing controller, we abstractly model closing of the barriers by the event $CrossingBar$, which non-deterministically assigns the variables bar_1 and bar_2 from the set BAR_POS. Let us note that in the abstract machine the crossing controller immediately knows when the train enters the close request area and makes an attempt to close the barriers. In further refinement steps we eliminate this unrealistic abstraction by introducing communication between the train and the crossing controller. In addition, in the $Train$ phase the event $TrainReact$ models triggering of the train brakes in the safe reaction spot.

The hazard present in the system is the situation when the train passes the crossing while at least one barrier is not closed. In terms of the introduced system variables and their states it can defined as follows:

$$train\_pos = DS \wedge (bar_1 = Opened \vee bar_2 = Opened).$$

In a more traditional (for Event-B invariants) form, this hazard can be dually reformulated as the following safety property:

$$train\_pos = SRS \wedge phase = Crossing \Rightarrow (bar_1 = Closed \wedge bar_2 = Closed) \vee emrg\_brakes = TRUE. \quad (1)$$

This safety requirement can be interpreted as follows: after the train, being in the safe reaction spot, reacts on signals from the crossing controller, the system is in the safe state only when both barriers are closed or the emergency brakes are activated. Obviously, this property cannot be formulated as an Event-B invariant – it might be violated due to possible communication and/or hardware failures. Our goal is to assess the probability of violation of the safety property (1). To achieve this, during the refinement process, we have to unfold (1) by introducing representation of all the system components that have impact on safety. Moreover, we should establish a relationship between the variables representing these components and the abstract variables present in (1).

The first refinement. In the first refinement step we examine in detail the system behaviour at the safe reaction spot – the last train position preceding the danger spot where the hazard may occur. As a result, the abstract event $TrainReact$ is refined by three events $TrainRelease_1$, $TrainRelease_2$ and $TrainStop$ that represent reaction of the train on the presence or absence of the release signal from the crossing controller. The first two events are used to model the situations when the release signal has been successfully delivered or lost respectively. The last one models the situation when the release signal has not been sent due to some problems at the crossing controller side. Please note that since the events $TrainRelease_2$ and $TrainStop$ perform the same actions, i.e., trigger the emergency brakes, they differ only in their guards.

The event $CrossingStatusReq$ that "decides" whether to send or not to send the release signal is very abstract at this stage – it does not have any specific guards except those that define the system phase and train position. Moreover, the variable $release\_snd$ is updated in the event body non-deterministically. To

Machine $RailwayCrossing\_R1$
Variables $\ldots, release\_snd, release\_rcv, emrg\_brakes\_failure, release\_com\_failure, \ldots$
Invariants $\cdots$
Events $\cdots$
$TrainRelease_1 \mathrel{\widehat{=}}$
when $phase = Train \wedge train\_pos = SRS \wedge release\_snd = TRUE$
$release\_comm\_failure = FALSE \wedge deceleration = FALSE \wedge comm\_ct = FALSE$
then
$emrg\_brakes := FALSE \,||\, release\_rcv := TRUE \,||\, phase := Crossing$
end
$TrainRelease_2 \mathrel{\widehat{=}}$
when $phase = Train \wedge train\_pos = SRS \wedge release\_snd = TRUE$
$release\_comm\_failure = TRUE \wedge deceleration = FALSE \wedge comm\_ct = FALSE$
then
$emrg\_brakes :| emrg\_brakes' \in BOOL \wedge (emrg\_brakes' = TRUE \Leftrightarrow emrg\_brakes\_failure = FALSE)$
$release\_rcv := TRUE \,||\, phase := Crossing$
end
$TrainStop \mathrel{\widehat{=}}$
when $phase = Train \wedge train\_pos = SRS \wedge release\_snd = FALSE \wedge deceleration = FALSE$
then
$\cdots$
end
$CrossingStatusReq \mathrel{\widehat{=}}$
when $phase = Crossing \wedge train\_pos = SRP$
then
$release\_snd :\in BOOL \,||\, phase := Env$
end
$ReleaseComm \mathrel{\widehat{=}}$
when $phase = Train \wedge train\_pos = SRS \wedge release\_snd = TRUE \wedge comm\_ct = TRUE$
then
$release\_comm\_failure \oplus| \; TRUE \quad p_1 \;\; FALSE \quad 1 - p_1 \,||\, comm\_ct := FALSE$
end
$TrainDec \mathrel{\widehat{=}}$
when $phase = Train \wedge train\_pos = SRS \wedge deceleration = TRUE$
then
$emrg\_brakes\_failure \oplus| \; TRUE \quad p_4 \;\; FALSE \quad 1 - p_4 \,||\, deceleration := FALSE$
end

Fig. 4. Railway crossing: first refinement

model the failures of communication and emergency brakes, we introduce two new events with *probabilistic* bodies – the events $ReleaseComm$ and $TrainDec$ correspondingly. For convenience, we consider communication as a part of the receiving side behaviour. Thus the release communication failure occurrence is modelled in the $Train$ phase while the train being in the SRS position. Some key details of the Event-B machine $RailwayCrossing\_R1$ that refines the abstract machine $RailwayCrossing$ are shown in Fig. 4.

The presence of concrete variables representing unreliable system components in $RailwayCrossing\_R1$ allows us to formulate two safety invariants (saf_inv_1 and saf_inv_2) that glue the abstract variable $emrg\_brakes$ participating in the safety requirement (1) with the (more) concrete variables $release\_rcv$, $emrg\_brakes\_failure$, $release\_snd$ and $release\_com\_failure$.

$$\textbf{saf\_inv}_\textbf{1} : train\_pos = SRS \wedge phase = Crossing \Rightarrow (emrg\_brakes = TRUE \Leftrightarrow release\_rcv = FALSE \wedge emrg\_brakes\_failure = FALSE)$$

$$\textbf{saf\_inv}_\textbf{2} : train\_pos = SRS \wedge phase = Crossing \Rightarrow (release\_rcv = FALSE \Leftrightarrow release\_snd = FALSE \vee release\_comm\_failure = TRUE)$$

We split the relationship between the variables into two invariant properties just to improve the readability and make the invariants easier to understand. Obviously, since the antecedents of both invariants coincide, one can easily merge them together by replacing the variable $release\_rcv$ in saf_inv_1 with the right hand side of the equivalence in the consequent of saf_inv_1. Please note that the variable $release\_snd$ corresponds to a certain combination of system actions and hence should be further unfolded during the refinement process.

The second refinement. In the second refinement step we further elaborate on the system functionality. In particular, we model the request messages that the train sends to the crossing controller, as well as sensors that read the position of the barriers. Selected excerpts from the second refinement machine $RailwayCrossing\_R2$ are shown in Fig. 5. To model sending of the close and status requests by the train, we refine the event $TrainIdle$ by two simple events $TrainCloseReq$ and $TrainStatusReq$ that activate sending of the close and status requests at the corresponding stages. In the crossing controller part, we refine the event $CrossingBars$ by the event $CrossingCloseReq$ that sets the actuators closing the barriers in response to the close request from the train. Clearly, in the case of communication failure occurrence during the close request transmission, both barriers remain open.

Moreover, the abstract event $CrossingStatusReq$ is refined by two events $CrossingStatusReq_1$ and $CrossingStatusReq_2$ to model a reaction of the crossing controller on the status request. The former event is used to model the situation when the close request has been successfully received (at the previous stage) and the latter one models the opposite situation. Notice that in the refined event $CrossingStatusReq_1$ the controller sends the release signal only when it has received both request signals and identified that both barriers are closed. This interconnection is reflected in the safety invariant saf_inv_3.

$$\textbf{saf\_inv}_\textbf{3} : train\_pos = SRP \wedge phase = Env \Rightarrow$$
$$(release\_snd = TRUE \Leftrightarrow close\_req\_rcv = TRUE \wedge$$
$$status\_req\_rcv = TRUE \wedge sensor_1 = Closed \wedge sensor_2 = Closed)$$

Here the variables $sensor_1$ and $sensor_2$ represent values of the barrier positioning sensors. Let us remind that the sensors are unreliable and can return the actual position of the barriers incorrectly. Specifically, the sensors can get stuck at their previous values or spontaneously change the values to the opposite ones. In addition, to model the communication failures, we add two new events $CloseComm$ and $StatusComm$. These events are similar to the $ReleaseComm$ event of the $RailwayCrossing\_R1$ machine. Rather intuitive dependencies between the train requests delivery and communication failure occurrences are defined by a pair of safety invariants saf_inv and saf_inv .

$$\textbf{saf\_inv}_\textbf{4} : train\_pos = SRP \wedge phase = Env \Rightarrow$$
$$(status\_req\_rcv = TRUE \Leftrightarrow status\_com\_failure = FALSE)$$

$$\textbf{saf\_inv}_\textbf{5} : train\_pos = CRP \wedge phase = Env \Rightarrow$$
$$(close\_req\_rcv = TRUE \Leftrightarrow close\_com\_failure = FALSE)$$

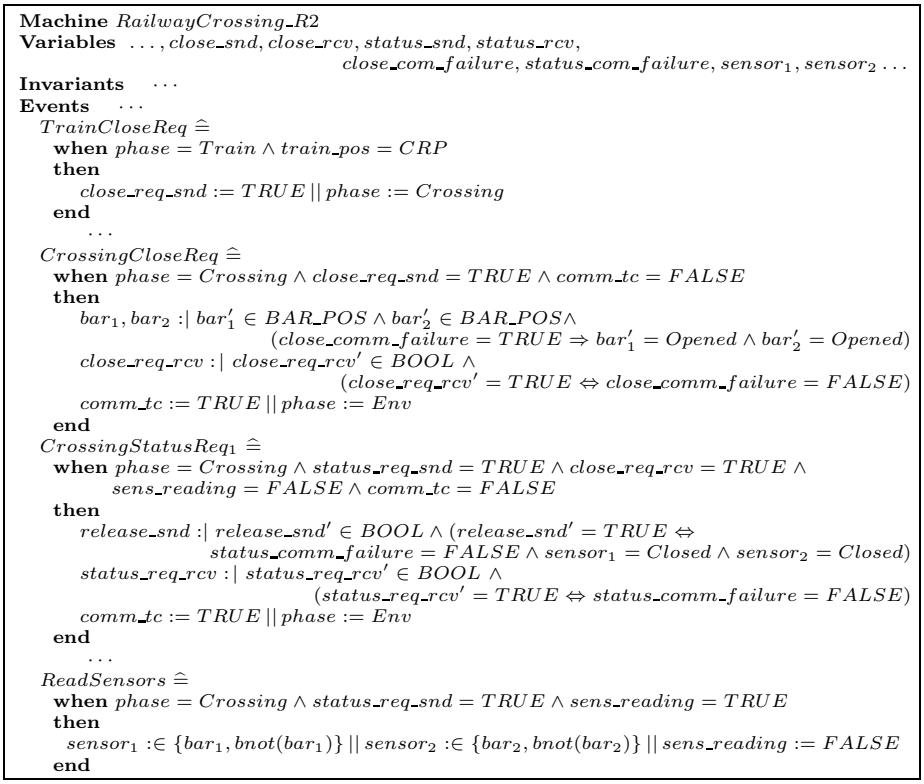


Fig. 5. Railway crossing: second refinement

The third refinement. In the third Event-B machine $RailwayCrossing\_R3$, we refine the remaining abstract representation of components mentioned in the safety requirement (1), i.e., modelling of the barrier motors and positioning sensors. We introduce the new variables $bar\_failure_1$, $bar\_failure_2$, $sensor\_failure_1$ and $sensor\_failure_2$ to model the hardware failures. These variables are assigned probabilistically in the newly introduced events $BarStatus$ and $SensorStatus$ in the same way as it was done for the communication and emergency brakes failures in the first refinement. We refine $CrossingCloseReq$ and $ReadSensors$ events accordingly. Finally, we formulate four safety invariants saf_inv_6, saf_inv_7, saf_inv and saf_inv to specify the correlation between the physical position of the barriers, the sensor readings, and the hardware failures.

$$\textbf{saf\_inv}_\textbf{6} : train\_pos = CRP \wedge phase = Env \Rightarrow (bar_1 = Closed \Leftrightarrow bar\_failure_1 = FALSE \wedge close\_comm\_failure = FALSE)$$

$$\textbf{saf\_inv}_\textbf{7} : train\_pos = CRP \wedge phase = Env \Rightarrow (bar_2 = Closed \Leftrightarrow bar\_failure_2 = FALSE \wedge close\_comm\_failure = FALSE)$$

$$\textbf{saf\_inv}_{\textbf{8}} : train\_pos = SRP \wedge phase = Env \Rightarrow (sensor_1 = Closed \Leftrightarrow ((bar_1 = Closed \wedge sensor\_failure_1 = FALSE) \vee (bar_1 = Opened \wedge sensor\_failure_1 = TRUE)))$$

$$\textbf{saf\_inv}_{\textbf{9}} : train\_pos = SRP \wedge phase = Env \Rightarrow (sensor_2 = Closed \Leftrightarrow ((bar_2 = Closed \wedge sensor\_failure_2 = FALSE) \vee (bar_2 = Opened \wedge sensor\_failure_2 = TRUE)))$$

The first two invariants state that the crossing barrier can be closed (in the post-state) only when the controller has received the close request and the barrier motor has not failed to start. The second pair of invariants postulates that the positioning sensor may return the value *Closed* in two cases – when the barrier is closed and the sensor works properly, or when the barrier has got stuck while opened and the sensor misreads its position.

Once we have formulated the last four safety invariants, there is no longer any variable, in the safety property (1), that cannot be expressed via some probabilistically updated variables introduced during the refinement process. This allows us to calculate the probability P_{SAF} that (1) is preserved:

$$P_{SAF} = P\{(bar_1 = Closed \wedge bar_2 = Closed) \vee emrg\_brakes = TRUE\} = P\{bar_1 = Closed \wedge bar_2 = Closed\} + P\{emrg\_brakes = TRUE\} - P\{bar_1 = Closed \wedge bar_2 = Closed\} \cdot P\{emrg\_brakes = TRUE \mid bar_1 = Closed \wedge bar_2 = Closed\}.$$

Let us recall that we have idenified four different types of failures in our system – the communication failure, the failure of the barrier motor, the sensor failure and emergency brakes failure. We suppose that the probabilities of all these failures are constant and equal to p_1, p_2, p_3 and p correspondingly. The first probability presented in the sum above can be trivially calculated based on the safety invariants saf_inv_7 and saf_inv :

$$P\{bar_1 = Closed \wedge bar_2 = Closed\} = P\{bar\_failure_1 = FALSE \wedge bar\_failure_2 = FALSE \wedge close\_comm\_failure = FALSE\} = (1 - p_1) \cdot (1 - p_2)^2.$$

Indeed, both barriers are closed only when the crossing controller received the close request and none of the barrier motors has failed. The calculation of the other two probabilities is slightly more complicated. Nevertheless, they can be straightforwardly obtained using the model safety invariants and basic rules for calculating probability. We omit the computation details due to a lack of space. The resulting probability of preservation of the safety property (1) is:

$$P_{SAF} = (1 - p_1) \cdot (1 - p_2)^2 + (1 - p_4) \cdot \left(1 - (1 - p_1)^3 \cdot (p_2 \cdot p_3 + (1 - p_2) \cdot (1 - p_3))^2\right) - (1 - p_1) \cdot (1 - p_2)^2 \cdot (1 - p_4) \cdot \left(1 - (1 - p_1)^2 \cdot (1 - p_3)^2\right).$$

Please note that P_{SAF} is defined as a function of probabilities of component failures, i.e., probabilities $p_1, \ldots, p$. Provided the numerical values of them are given, we can use the obtained formula to verify whether the system achieves the desired safety threshold.

5 Discussion

.1 Related or

Formal methods are extensively used for the development and verification of safety-critical systems. In particular, the B Method and Event-B are successfully being applied for formal development of railway systems [12,5]. A safety analysis of the formal model of a radio-based railway crossing controller has also been performed with the I theorem prover [16,15]. However, the approaches for integrating formal verification and quantitative assessment are still scarce.

Usually, quantitative analysis of safety relies on probabilistic model checking techniques. For instance, in [11], the authors demonstrate how the quantitative model checker PRISM [17] can be used to evaluate system dependability attributes. The work reported in [8] presents model-based probabilistic safety assessment based on generating PRISM specifications from Simulink diagrams annotated with failure logic. A method pFMEA (probabilistic Failure Modes and Effect Analysis) also relies on the PRISM model checker to conduct quantitative analysis of safety [9]. The approach integrates the failure behaviour into the system model described in continuous time Markov chains via failure injection. In [14] the authors propose a method for probabilistic model-based safety analysis for synchronous parallel systems. It has been shown that different types of failures, in particular per-time and per-demand, can be modelled and analysed using probabilistic model checking.

However, in general the methods based on model checking aim at safety evaluation of already developed systems. They extract a model eligible for probabilistic analysis and evaluate impact of various system parameters on its safety. In our approach, we aim at providing the designers with a safety-explicit development method. Indeed, safety analysis is essentially integrated into system development by refinement. It allows us to perform quantitative assessment of safety within proof-based verification of the system behaviour.

.2 Conclusions

In this paper we have proposed an approach to integrating quantitative safety assessment into formal system development in Event-B. The main merit of our approach is that of merging logical (qualitative) reasoning about correctness of system behaviour with probabilistic (quantitative) analysis of its safety. An application of our approach allows the designers to obtain a probability of hazard occurrence as a function over probabilities of component failures.

Essentially, our approach sets the guidelines for safety-explicit development in Event-B. We have shown how to explicitly define safety properties at different levels of refinement. The refinement process has facilitated not only correctness-preserving model transformations but also establishes a logical link between safety conditions at different levels of abstraction. It leads to deriving a logical representation of hazardous conditions. An explicit modelling of probabilities of component failures has allowed us to calculate the likelihood of hazard occurrence. The B Method and Event-B are successfully and intensively used in

the development of safety-critical systems, particularly in the railway domain. We believe that our approach provides the developers with a promising solution unifying formal verification and quantitative reasoning.

In our future work we are planning to further extend the proposed approach to enable probabilistic safety assessment at the architectural level.

Ac nowledgments. This work is supported by IST FP7 DEPLO Project. We also wish to thank the anonymous reviewers for their helpful comments.

References

1. Abrial, J.R.: E tending B without Changing it (for Developing Distributed Systems). In: Habiras, H. (ed.) First Conference on the B Method, pp. 169–190 (1996)
2. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (2005)
3. Abrial, J.R.: Modeling in Event-B. Cambridge University Press, Cambridge (2010)
4. Avizienis, A., Laprie, J.C., Randell, B.: Fundamental Concepts of Dependability, Research Report No 1145, LAAS-CNRS (2001)
5. Craigen, D., Gerhart, S., Ralson, T.: Case Study: Paris Metro Signaling System. IEEE Software, 32–35 (1994)
6. EU-pro ect DEPLO , http://www. eploy-pro ect.eu/
7. Feller, W.: An Introduction to Probability Theory and its Applications, vol. 1. John Wiley Sons, Chichester (1967)
8. Gomes, A., Mota, A., Sampaio, A., Ferri, F., Buzzi, J.: Systematic Model-Based Safety Assessment Via Probabilistic Model Checking. In: Margaria, T., Steffen, B. (eds.) ISoLA 2010. LNCS, vol. 6415, pp. 625–639. Springer, Heidelberg (2010)
9. Grunske, L., Colvin, R., Winter, K.: Probabilistic Model-Checking Support for FMEA. In: QEST 2007, International Conference on Quantitative Evaluation of Systems (2007)
10. Hallerstede, S., Hoang, T.S.: Qualitative Probabilistic Modelling in Event-B. In: Davies, J., Gibbons, J. (eds.) IFM 2007. LNCS, vol. 4591, pp. 293–312. Springer, Heidelberg (2007)
11. Kwiatkowska, M., Norman, G., Parker, D.: Controller Dependability Analysis by Probabilistic Model Checking. In: Control Engineering Practice, pp. 1427–1434 (2007)
12. Lecomte, T., Servat, T., Pouzancre, G.: Formal Methods in Safety-Critical Railway Systems. In: Brasilian Symposium on Formal Methods (2007)
13. McIver, A.K., Morgan, C.C.: Abstraction, Refinement and Proof for Probabilistic Systems. Springer, Heidelberg (2005)
14. Ortmeier, F., Gudemann, M.: Probabilistic Model-Based Safety Analysis. In: Workshop on Quantitative Aspects of Programming Languages, QAPL 2010. EPTCS, pp. 114–128 (2010)
15. Ortmeier, F., Reif, W., Schellhorn, G.: Formal Safety Analysis of a Radio-Based Railroad Crossing Using Deductive Cause-Consequence Analysis (DCCA). In: Dal Cin, M., Kaaniche, M., Pataricza, A. (eds.) EDCC 2005. LNCS, vol. 3463, pp. 210–224. Springer, Heidelberg (2005)
16. Ortmeier, F., Schellhorn, G.: Formal Fault Tree Analysis: Practical E periences. In: International Workshop on Automated Verification of Critical Systems, AVoCS 2006. ENTCS, vol. 185, pp. 139–151. Elsevier, Amsterdam (2007)

17. PRISM – Probabilistic Symbolic Model Checker, http://www.prismmo elchecker.org/
18. Rigorous Open Development Environment for Comple Systems (RODIN): Deliverable D7, Event-B Language, http://ro in.cs.ncl.ac.uk/
19. Rigorous Open Development Environment for Comple Systems (RODIN): IST FP6 STREP pro ect, http://ro in.cs.ncl.ac.uk/
20. RODIN. Event-B Platform, http://www.event-b.org/
21. Tarasyuk, A., Troubitsyna, E., Laibinis, L.: Quantitative Verification of System Safety in Event-B. Tech. Rep. 1010, Turku Centre for Computer Science (2011)
22. Tarasyuk, A., Troubitsyna, E., Laibinis, L.: Towards Probabilistic Modelling in Event-B. In: Mery, D., Merz, S. (eds.) IFM 2010. LNCS, vol. 6396, pp. 275–289. Springer, Heidelberg (2010)

Paper VI

Augmenting Formal Development of Control Systems with Quantitative Reliability Assessment

Anton Tarasyuk, Elena Troubitsyna and Linas Laibinis

Originally published in: *Proceedings of 2nd International Workshop on Software Engineering for Resilient Systems (SERENE 2010)*, ACM, 2010

Augmenting Formal Development of Control Systems with Quantitative Reliability Assessment

Anton Tarasyuk
Åbo Akademi University
Joukahaisenkatu 3-5A
20520 Turku, Finland
anton.tarasyuk@abo.fi

Elena Troubitsyna
Åbo Akademi University
Joukahaisenkatu 3-5A
20520 Turku, Finland
elena.troubitsyna@abo.fi

Linas Laibinis
Åbo Akademi University
Joukahaisenkatu 3-5A
20520 Turku, Finland
linas.laibinis@abo.fi

ABSTRACT

Formal methods, in particular the B Method and its extension Event-B, have demonstrated their value in the development of complex control systems. However, while providing us with a powerful development platform, these frameworks poorly support quantitative assessment of dependability attributes. Yet, by assessing dependability at the early design phase we would facilitate development of systems that are not only correct-by-construction but also achieve the desired dependability level. In this paper we demonstrate how to integrate reliability assessment performed via Markov analysis into refinement in Event-B. Such an integration allows us to combine logical reasoning about functional correctness with probabilistic reasoning about reliability. Hence we obtain a method that enables building the systems that are not only provably correct but also have a required level of reliability.

Keywords

Reliability assessment, formal modelling, Event-B, Markov processes, refinement, probabilistic model checking

1. INTRODUCTION

Formal approaches provide us with rigorous methods for establishing correctness of complex systems. The advances in expressiveness, usability and automation offered by these approaches enable their use in the design of wide range of complex dependable systems. For instance, Event-B [1] provides us with a powerful framework for developing systems correct-by-construction. The top-down development paradigm based on stepwise refinement adopted by Event-B has proved its worth in several industrial projects [19, 7].

While developing system by refinement, we start from a specification that abstracts away from low-level design decisions yet defines the most essential behaviour and properties of the system. While refining the abstract specification, we gradually introduce the desired implementation decisions that initially were modelled non-deterministically.

SERENE '10, 13-16 April 2010, London, United Kingdom

In general, there may be several ways to resolve non-determinism, i.e., there are might be several possible implementation decisions that adhere to the abstract specification. These alternatives are equivalent from the correctness point of view, i.e., they faithfully implement functional requirements. Yet they might be different from the point of view of non-functional requirements, e.g., reliability, performance etc. Early quantitative assessment of various design alternatives is certainly useful and desirable. However, within the current refinement frameworks there are no scalable solutions for that [15]. In this paper we propose an approach to overcome this problem.

Reliability is a probability of system to function correctly over a given period of time under a given set of operating conditions [22, 24, 17]. Obviously, to assess reliability of various design alternatives, we need to model their behaviour stochastically. In this paper, we show how Event-B models can be augmented with probabilistic information required to perform the quantitative dependability analysis. We also demonstrate how probabilistic model verification [18] of such augmented models can be used to assess system reliability. Such an approach allows us to use our formal models not only for reasoning about correctness but also for quantitative analysis of dependability. We exemplify our approach by refinement and reliability evaluation of a simple control system.

In this paper we focus on reliability assessment of control systems. Control systems are usually cyclic, i.e., at periodic intervals they get inputs from the sensors, process them, and then output new values to the actuators. We show how to model such systems in Event-B as well as formally reason about the behaviour of control systems in terms of observable event traces. Moreover, once we augment Event-B models with probabilities, we extend the notion of event traces by incorporating the probabilistic information into them. As a result, the formal semantics given in terms of probabilistic events traces allows us to establish a clear connection to the classical probabilistic reliability definitions used in engineering [24, 17].

The remainder of the paper is structured as follows. In Section 2 we give a brief overview of our modelling formalism the Event-B framework. Section 3 presents abstract specification of a control system and also shows how we can reason about the behaviour of a system in terms of event traces. In Section 4 we explain how we can do stochastic modelling in Event-B. We also define here the notion of probabilistic event traces. Section 5 explains how probabilistic model verification can done using the PRISM models checker. In Sec-

tion 6 we exemplify our approach by presenting a case study of the simple heater control system. Finally, in Sections 7 and 8 we overview the related work, discuss the obtained results and propose some directions for the future work.

2. INTRODUCTION TO EVENT-B

The B Method [2] is an approach for the industrial development of highly dependable software. The method has been successfully used in the development of several complex real-life applications [19, 7]. Event-B is an extension of the B Method to model parallel, distributed and reactive systems. The Rodin platform [21] provides automated tool support for modelling and verification (by theorem proving) in Event-B. Currently Event-B is used in the EU project Deploy [8] to model several industrial systems from automotive, railway, space and business domains.

In Event-B a system specification is defined using an abstract (state) machine notion [20]. An abstract machine encapsulates the state (the variables) of a model and defines operations on its state. It has the following general form:

Machine M
Variables v
Invariants I
Events
$init$
evt_1
$\dots$
evt_N

The machine is uniquely identified by its name M. The state variables, v, are declared in the **Variables** clause and initialised in the *init* event. The variables are strongly typed by the constraining predicates I given in the **Invariants** clause. The invariant clause might also contain other predicates defining properties that should be preserved during system execution.

The dynamic behaviour of the system is defined by the set of atomic events specified in the **Events** clause. Generally, an event can be defined as follows:

$$evt \mathrel{\widehat{=}} \textbf{when } g \textbf{ then } S \textbf{ end},$$

where the guard g is a conjunction of predicates over the state variables v and the action S is an assignment to the state variables.

The guard defines the conditions under which the action can be executed, i.e., when the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution non-deterministically. If none of the events is enabled then the system deadlocks.

In general, the action of an event is a composition of assignments executed simultaneously. In this paper the simultaneous execution is denoted as $\|$. The assignments can be either deterministic or non-deterministic. A deterministic assignment, $x := E(v)$, has the standard syntax and meaning. A non-deterministic assignment is denoted as $x :\in S$, where S is a set of values. As a result of such a non-deterministic assignment, x can get any value belonging to S.

Event-B employs top-down refinement-based approach to system development. Development starts from an abstract system specification that models the most essential functional requirements. While capturing more detailed requirements, each refinement step typically introduces new events and variables into the abstract specification. These new events correspond to stuttering steps that are not visible at the abstract level. By verifying correctness of refinement, we ensure that all invariant properties of (more) abstract machines are preserved. A detailed description of formal semantics of Event-B and foundations of the verification process can be found in [20].

Event-B adopts the interleaving semantics, i.e., if several events are enabled simultaneously only one of them is (non-deterministically) chosen for execution. This allows us to implicitly model parallelism. Specifically, if several enabled events are defined on disjoint sets of variables, they can be eventually implemented as parallel activities. However, sometimes it is convenient to explicitly model parallel execution of several events already at the abstract system stage. To achieve this, we rely on the Butler's idea of event fusion [5]. For two events evt_1 and evt_2,

$$evt_1 \mathrel{\widehat{=}} \textbf{when } g_1 \textbf{ then } S_1 \textbf{ end}$$

$$evt_2 \mathrel{\widehat{=}} \textbf{when } g_2 \textbf{ then } S_2 \textbf{ end}$$

their parallel composition $evt_1 \parallel evt_2$ is defined as

$$evt_1 \parallel evt_2 \mathrel{\widehat{=}} \textbf{when } g_1 \wedge g_2 \textbf{ then } S_1 \parallel S_2 \textbf{ end}.$$

In such a way we explicitly model that an execution of events evt_1 and evt_2 is synchronised. The event fusion technique can be also applied in reverse, i.e., by splitting an event into a group of events that should be always executed simultaneously. This can be useful in the refinement process, e.g., when decomposing system into several components that synchronise their activities via such parallel events.

In general, while refining a system in Event-B, we gradually introduce certain design decisions into the system specification. Sometimes there are several refinement alternatives that can adequately implement a certain functional requirement. These alternatives can have different impact on non-functional system requirements, e.g., such as dependability, performance etc. Obviously, it would be advantageous to evaluate this impact already at the development stage to ensure that the most optimal solutions are chosen. To achieve this, we would need to perform quantitative analysis of system dependability. For instance, to assess system reliability, it is necessary to evaluate the probability of system functioning correctly over the given period of time. To assess safety of a system that terminates when its safety is breached, it is imperative to evaluate the probability of termination within a certain period of time. These kinds of evaluations are rather easily performed via probabilistic model checking.

3. MODELLING A CONTROL SYSTEM IN EVENT-B

3.1 Abstract Specification of a Control System

Usually a control system have cyclic behaviour – a system goes through a number of predefined execution phases. A system phase usually includes reading the sensors that monitor the controlled physical processes, processing the obtained sensor values, and, finally, setting actuators accordingly to a predefined algorithm. Ideally, the system can

function in this way infinitely. However, different failures may force the system to shutdown at any moment. In the most abstract way, we can specify such a control system as follows:

Machine CS
Variables st
Invariants
$st \in \{ok, nok\}$
Events
$init \mathrel{\widehat{=}}$
begin
$st := ok$
end
$step \mathrel{\widehat{=}}$
when
$st = ok$
then
$st :\in \{ok, nok\}$
end
$shutdown \mathrel{\widehat{=}}$
when
$st = nok$
then
$skip$
end

The variable st abstractly models a state of the system that can be in either operational (ok) or failed (nok). The event *step* models one iteration of the system execution. As a result of this event, the system can stay operational or fail. In the first case, the system continues to its next iteration. In the latter case, the system shutdown is initiated (specified by the event *shutdown*). Here the system shutdown is abstractly modelled as infinite stuttering, keeping the system in the failed state forever. Let us note that Event-B does not support explicit modelling of real time. Therefore, in our specification we assume that the controller is sufficiently fast to react on the changes of the environment in the timely manner.

We can refine the abstract specification CS by introducing specific details for any concrete control system. For example, we may explicitly introduce new events modelling the environment as well as reading the sensors or setting the actuators. The event *step* can be also refined into, e.g., *detection* operation, which decides whether the system can continue its normal operation or has to shutdown due to some unrecoverable failure.

3.2 Event-B Traces

One of convenient ways to reason about the behaviour of an Event-B specification is by using the notion of event traces. Essentially, this allows us to define a formal semantics in terms of traces of observable events of the system.

Formally, an event trace is a sequence of events (event labels) that can be observed during the system execution. For example, this is one possible trace of the system CS described above:

$$< step, step, step, shutdown, shutdown, ... > .$$

The initialisation event of an Event-B machine is executed only ones when the system starts to function. It defines the initial state of any event trace, but it is not a part of a trace as such. Moreover, we assume that all the assignments of the *init* event are deterministic and hence the initial state of the system is determined unambiguously. Conventionally, the infinite stuttering at the end of a trace is omitted. Therefore, the above trace can be simply written as

$$< step, step, step, shutdown > .$$

Due to branching and non-determinism that could be present in a system model, the system behaviour cannot be described as a single trace. Instead, a set of all possible traces is used to define its execution.

We introduce the operator $Traces$ that, for a given model, returns the set of all its observable traces. For example, for the system CS, the set of all its traces is

$$\begin{aligned} Traces(CS) \quad = \quad & \{< step_1, ..., step_i, shutdown > \mid \ i \in \mathbb{N}_1\} \\ & \cup \\ & \{step_1, step_2, step_3, ...\}. \end{aligned}$$

Therefore, all possible CS traces include execution of a finite number of *step* events followed by *shutdown*, or, in ideal case, infinite number of *step* events while never failing.

The system CS is a really simple one. For more complicated cases, the system traces would include all possible interleavings of different system events. However, the simplicity level of CS is chosen intentionally to emphasise the cyclic nature of such systems.

A machine M' is *trace refinement* of machine M, denoted $M \sqsubseteq_{tr} M'$, if any trace of M' is also a trace of M, i.e., any trace that is observable for the concrete system can be also observed in the abstract system. Formally,

$$M \sqsubseteq_{tr} M' \ \textit{iff} \ Traces(M') \ \subseteq \ Traces(M).$$

It has been shown [6] that the proof obligations defined for standard Event-B refinement are sufficient conditions for trace refinement as well. Formally, for any two models M and M', where M' is a refinement of M, denoted as $M \sqsubseteq M'$,

$$M \sqsubseteq M' \quad \Rightarrow \quad M \sqsubseteq_{tr} M'.$$

Refined Event-B models typically add new events that operate on newly introduced variables, thus increasing granularity of model execution. In the abstract specification such events correspond to unobservable, internal events, which can be modelled as *skip* statements. While showing that one Event-B model is trace refined by another, these events are excluded from concrete traces. In other words, the abstract specification determines which events are considered observable. Only those events (or their refined versions) are taken into account while demonstrating that the concrete model is a trace refinement of the abstract one.

The machine CS describes a very abstract control system, emphasising its cyclic nature. Any refinement of such a specification preserves this property. This can be formulated as the following simple theorem.

THEOREM 1. *Let M be an Event-B machine such that $CS \sqsubseteq M$. Then, for any positive natural number k, $k \in \mathbb{N}_1$, and a concrete trace tr, $tr \in Traces(M)$, that contains k first consecutive step events,*

$$tr \in Traces(CS).$$

Proof Directly follows from the fact that

$$CS \sqsubseteq M \quad \Rightarrow \quad CS \sqsubseteq_{tr} M$$

and the definition of trace refinement. □

Therefore, we can compare the refined and abstract systems by executing (simulating) both of them side by side exactly the same number of iterations. The additional events of the refined system are treated as internal and can be dismissed.

4. STOCHASTIC MODELLING IN EVENT-B.

Let us observe that Event-B is a state-based formalism. The state space of the system specified in Event-B is formed by the values of the state variables. The transitions between states are determined by the actions of the system events. The states that can be reached as a result of event execution are defined by the current state. If we augment Event-B specification with the probabilities of reaching the next system state from the current one, we obtain a probabilistic transition system [3]. In case the events are mutually exclusive, i.e., only one event is enabled at each system state, the specification can be represented by a Markov chain. Otherwise, it corresponds to a Markov Decision process [9, 12, 25]. More specifically, it is a discrete time Markov process since we can only use it to describe the states at certain instances of time.

While augmenting an Event-B specification with probabilities, we replace non-deterministic assignments in some event actions with their probabilistic counterparts. While the result of executing a non-deterministic assignment is arbitrary (within the given set), the result of executing the probabilistic one is more predictable. Indeed, it has been shown that such probabilistic choice always refines its corresponding non-deterministic counterpart [14]. Therefore, augmenting Event-B specifications with probabilities constitutes a valid refinement step.

Incorporation of probabilistic information into Event-B models also allows us extend the notion of event traces for probabilistic systems. In this paper we focus our attention on completely probabilistic Event-B models, i.e. we assume that all non-deterministic system behaviour has been refined by probabilistic one. We propose the following trace notation for the Event-B systems augmented with probabilistic information:

$$< evt_1.p_1,\ evt_2.p_2,\ \ldots,\ evt_n.p_n >,$$

where p_i are probabilities of executing evt_{i+1} as the next system event. If evt_n is the last event of a finite trace then $p_n = 1$, i.e. the last event evt_n is assumed to be a stuttering event. For a model M, the set of all such traces is denoted as $PTraces(M)$.

To illustrate probabilistic traces, let us go back to our abstract CS specification. Assume that the nondeterministic assignment $st :\in \{ok, nok\}$ has been refined by the probabilistic choice between $st := ok$, with probability p, and $st := nok$, with probability $1 - p$. Then

$$\begin{aligned} PTraces(CS) \quad = \quad & \{< step_1.p, \ldots, \\ & step_{k-1}.p, step_k.(1-p), shutdown.1 > \mid \ k \in \mathbb{N}_1\}. \end{aligned}$$

If we assume that the system CS can fail with a non-zero probability (i.e., $0 < 1 - p$), this fact implies that the infinite trace containing only *step* events is impossible, i.e., its probability is 0, and thus can be excluded from consideration. Let us explain how we can calculate such trace probabilities.

For any trace tr, we define its length $|tr|$ as a number of all its events excluding the final stuttering event. Then the overall probability of a probabilistic trace tr is defined as

$$pr(tr) = \prod_{i=1}^{|tr|} p_i.$$

Moreover, we require that for all possible traces

$$\sum_{tr} pr(tr) = \sum_{tr} \prod_{i=1}^{|tr|} p_i = 1.$$

The last property simply states that the set of probabilistic traces should be complete. It can be derived from the (more) basic requirement that, for all execution branches creating multiple traces, the sum of all probabilities to choose a different branch (event) is equal to 1.

Let us assume that we have two Event-B models M and M' such that $M \sqsubseteq_{tr} M'$. Moreover, we have extended both models with certain probabilistic information. Having this quantitative information about the model behaviour allows us to strengthen the notion of trace refinement by requiring that the refined model is executed longer with a higher probability.

DEFINITION 1. *For two Event-B models M and M', we say that M' is a probabilistic trace refinement of M, denoted $M \sqsubseteq_{ptr} M'$, iff*

(i) *M' is a trace refinement of M' ($M \sqsubseteq_{tr} M'$)*

(ii) *for any $t \in \mathbb{N}$,*

$$\sum_{\substack{tr \in PTraces(M') \\ |tr| \le t}} pr(tr) \ \le \sum_{\substack{tr \in PTraces(M) \\ |tr| \le t}} pr(tr).$$

The condition (ii) requires that that the probability of the refined system to finish its execution in t or less execution steps should be less than the one for the abstract system.

The quantitative requirement (ii) explicitly makes this definition of probabilistic trace refinement biased towards the systems where longer execution is considered advantageous. We choose this definition because in this paper we focus on reliability assessment of control systems. As we will show below, there is a clear connection between the quantitative requirement (ii) and probabilistic modelling of system reliability.

Sometimes the inequality from (ii) does not hold for every t but for only some finite interval $[\,0, \bar{t}\,]$.

DEFINITION 2. *We say that M' is a partial probabilistic trace refinement of M' for $t \in [\,0, \bar{t}\,]$ iff*

(i) *M' is a trace refinement of M' ($M \sqsubseteq_{tr} M'$)*

(iii) *for any $t \in \mathbb{N}$ such that $t \le \bar{t}$,*

$$\sum_{\substack{tr \in PTraces(M') \\ |tr| \le t}} pr(tr) \ \le \sum_{\substack{tr \in PTraces(M) \\ |tr| \le t}} pr(tr).$$

Let us note that the inequalities form (ii) and (iii) can be rewritten as follows

(iv) $$\sum_{\substack{tr \in PTraces(M) \\ |tr| > t}} pr(tr) \ \le \sum_{\substack{tr \in PTraces(M') \\ |tr| > t}} pr(tr).$$

In this formulation, the property simply states that the probability of the refined system to execute longer should be higher.

As it was mentioned before, we are interested in assessing reliability of control systems. In engineering, reliability [24, 17] is generally measured by the probability that an entity $\mathcal{E}$ can perform a required function under given conditions for the time interval $[0, t]$:

$$R(t) = \mathbf{P}[\mathcal{E} \textit{ not failed over time } [0, t]].$$

Let $\mathcal{T}(M)$ be a random variable measuring the number of iterations of a control system M before the system shutdown, and $F_M(t)$ is its cumulative distribution function. Since the system M functions properly while it stays operational, we can define reliability function of such a system in the following way:

$$R_M(t) \ = \ \mathbf{P}\{\mathcal{T}(M) > t\}.$$

Then $R_M(t)$ and $F_M(t)$ are related as follows:

$$R_M(t) = \mathbf{P}[\mathcal{T}(M) > t] = 1 - \mathbf{P}[\mathcal{T}(M) \leq t] = 1 - F_M(t).$$

We intentionally specified our abstract control system CS in such a way that its traces directly "record" the system iterations. Then the number of events in the system trace (excluding the final event modelling system shutdown) corresponds to the number of iterations the system is operational before its failure. If we assume that one system iteration is executed in one time unit, it is easy to notice that the cumulative distribution function $F_M(t)$ is equal to the probability that the length of the event trace is less or equal to the number of execution steps t, i.e.,

$$F_M(t) \ = \sum_{\substack{tr \in PTraces(M) \\ |tr| \leq t}} pr(tr).$$

That means that the condition (ii) can be rewritten as

$$F_{M'}(t) \leq F_M(t)$$

for any $t \in \mathbb{N}$. On the other hand, the system reliability $R_M(t)$

$$R_M(t) \ = \ 1 - F_M(t) \ = \sum_{\substack{tr \in PTraces(M) \\ |tr| > t}} pr(tr). \qquad (1)$$

This allows us to rewrite the equivalent condition (iv) into

$$R_M(t) \leq R_{M'}(t)$$

for any $t \in \mathbb{N}$. Therefore, the conditions (ii), (iii) and (iv) essentially require that the reliability of the refined system should improve.

Let us note that we are not modelling time explicitly in our Event-B specification. However, we can assume that an execution of the control cycle takes constant predefined amount of time, i.e. the duration length of the control cycle is constant. Therefore, we can extrapolate our estimation of reliability in terms of iterations into classic reliability estimation in terms of real time. In general, reliability analysis is a complex mathematical and engineering problem. In the next section we demonstrate how the Event-B modelling framework extended with probabilistic model checking can be used to tackle this problem.

5. STOCHASTIC REASONING IN PRISM

As mentioned before, Event-B models augmented with probabilistic information generally correspond to Markov processes. However, the current Event-B framework does not support modelling of stochastic behaviour, while probabilistic model checking is one of available techniques widely used for analysis of Markov models. In particular, the probabilistic model checking framework developed by Kwiatkowska et al. [13] supports verification of Discrete-Time Markov Chains (DTMC) and Markov Decision Processes (MDP). The framework also has mature tool support – the PRISM model checker [18]. To enable quantitative dependability analysis of Event-B models, it would be advantageous to bridge Event-B modelling with the PRISM model checking. Next we briefly describe modelling in PRISM.

The PRISM modelling language is a high-level state-based language. It relies on the Reactive Modules formalism of Alur and Henzinger [3]. PRISM supports the use of constants and variables that can be integers, doubles (real numbers) and Booleans. Constants are used, for instance, to define the probabilities associated with variable updates. The variables in PRISM are finite-ranged and strongly typed. They can be either local or global. The definition of an initial value of a variable is usually attached to its declaration. The state space of a PRISM model is defined by the set of all variables, both global and local.

In general, a PRISM specification looks as follows:

dtmc
const double $p_{11} = \ldots$;
$\ldots$
global $v : Type$ **init** $\ldots$;
module M_1
$v_1 : Type$ **init** $\ldots$;
$[]\ g_{11} \rightarrow p_{11} : act_{11} + \cdots + p_{1n} : act_{1n}$;
$[sync]\ g_{12} \rightarrow q_{11} : act'_{11} + \cdots + q_{1m} : act'_{1m}$;
$\ldots$
endmodule
module M_2
$v_2 : Type$ **init** $\ldots$;
$[sync]\ g_{21} \rightarrow p_{21} : act_{21} + \cdots + p_{2k} : act_{2k}$;
$[]\ g_{22} \rightarrow q_{21} : act'_{21} + \cdots + q_{2l} : act'_{2l}$;
$\ldots$
endmodule

A system specification in PRISM is constructed as a parallel composition of modules. They can be independent of each other or interact with each other. Each module has a number of local variables – denoted as v_1, v_2 in the specification above – and a set of guarded commands that determine its dynamic behaviour. The guarded commands can have names (labels). Similarly to the events of Event-B, a guarded command can be executed if its guard evaluates to $TRUE$. If several guarded commands are enabled then the choice between them can be either non-deterministic, in case of MDP, or probabilistic (according to the uniform distribution), in case of DTMC. In general, the body of a guarded command is a probabilistic choice between deterministic as-

signments. It is of the following form:

$$p_1 : act_1 + \cdots + p_n : act_n,$$

where + denotes the probabilistic choice, $act_1, \ldots, act_n$ are deterministic assignments, and $p_1, \ldots, p_n$ are the corresponding probabilities.

Synchronisation between modules is defined via guarded commands with the matching names. For instance, in the PRISM specification shown above, the modules M_1 and M_2 have the guarded commands labelled with the same name *sync*. Whenever both commands are enabled, the modules M_1 and M_2 synchronise by simultaneously executing the bodies of these commands. It is easy to notice that the guarded command synchronisation in PRISM essentially models the same operational behaviour as the event fusion in Event-B.

While analysing a PRISM model, we define a number of temporal logic properties and systematically check the model to verify them. Properties of discrete-time PRISM models, i.e, DTMC and MDP, are expressed formally in the probabilistic computational tree logic [11]. The PRISM property specification language supports a number of different types of such properties. For example, the **P** operator is used to refer to the probability of a certain event occurrence. The operator **G**, when used inside the operator **P**, allows us to express *invariant* properties, i.e., properties maintained by the system globally. In particular, the property $\mathbf{P}_{=?}[\mathbf{G} \leq t\ prop]$ returns a probability that the predicate *prop* remains $TRUE$ in all states within the period of time t.

Let $\mathcal{OP}$ be a predicate defining a subset of operational system states. Then, the PRISM property

$$\mathbf{P}_{=?}[\mathbf{G} \leq t\ \mathcal{OP}] \tag{2}$$

gives us the probability that the system will stay operational during the first t iterations, i.e, it is the probability that, during that time, the system will stay within the subset of operational states. In the previous section we have introduced the random variable $\mathcal{T}(M)$ that measures the uptime of the probabilistic Event-B model M, i.e. the number of iterations the system M stays operational before its failure. Therefore, the property (2) clearly corresponds to our definition of reliability (1) for probabilistic Event-B traces.

In the next section we exemplify the proposed approach by considering a case study – the heater control system.

6. CASE STUDY

Our case study is a simple control system with two main entities – a tank containing some liquid and a heater controller. The controller tries to keep a temperature (tmp) of the liquid between the minimal (tmp_{min}) and the maximal (tmp_{max}) boundaries. Periodically, the temperature sensor produces temperature measurements that are read by the controller. We assume that the sensor is faulty and can fail (and eventually recover) with predefined probabilities on any control cycle. Obviously, such a stochastic sensor behaviour affects the overall system reliability. The controller has a fault detector that analyses each reading to detect whether the sensor functions properly or it has failed. If no fault is detected, the controller makes routine control and the system continues to operate in the same iterative manner. This constitutes the normal (fault-free) system state. However, if the fault detector discovers a sensor failure then the system enters the degraded state. In this state it outputs the last good sensor reading. At the same time, it keeps periodically reading the sensor outputs to detect whether it has recovered. The system can stay in the degraded state for a limited period of time. Specifically, it cannot exceed N iterations, where N is a threshold calculated by the controller. The threshold equals to the minimal difference between the last good output of the sensor and the upper or lower temperature limits. If sensor recovers from its failure within the allowed time limit, the system returns to the normal state and its normal operation is resumed. Otherwise, the system aborts.

The most abstract Event-B specification that models such a system is our simple CS machine presented in Section 3. The specification below (the machine MCH_1) is one possible refinement of it. The variable st models the current state of the system, which can be operating (ok) or failed (nok). We also introduce the variable $phase$ that models the phases that the system goes through within one iteration: first modelling of the environment, i.e. changes of the temperature, then reading the sensor, detecting sensor failure, and, finally, executing of the routine control or aborting the system.

Machine MCH_1
Variables $st, s, tmp, tmp_{est}, phase, heat, cnt, N$
Invariants $s \in \{0,1\} \wedge tmp \in \mathbb{N} \wedge tmp_{est} \in \mathbb{N} \wedge cnt \in \mathbb{N}$
$N \in \mathbb{N} \wedge phase \in \{env, read, det, cont\} \wedge heat \in \{on, off\}$
$phase = cont \Rightarrow tmp_{min} \leq tmp \leq tmp_{max}$
Initialisation $st := ok \,\|\, s := 1 \,\|\, tmp := tmp_0 \,\|\, tmp_{est} := tmp_0 \,\|\, phase := env \,\|\, heat := on \,\|\, cnt := 0 \,\|\, N := tmp_{max}$
Event $plant_{on} \mathrel{\widehat{=}}$
  when
    $phase = env \wedge heat = on$
  then
    $tmp := tmp + 1 \parallel phase := read$
  end
Event $plant_{off} \mathrel{\widehat{=}}$
  when
    $phase = env \wedge heat = off$
  then
    $tmp := tmp - 1 \parallel phase := read$
  end
Event $sensor_{ok} \mathrel{\widehat{=}}$
  when
    $phase = read \wedge s = 1$
  then
    $s :\in \{0,1\} \parallel phase := det$
  end
Event $sensor_{nok} \mathrel{\widehat{=}}$
  when
    $phase = read \wedge s = 0$
  then
    $s :\in \{0,1\} \parallel phase := det$
Event $detection_{ok} \mathrel{\widehat{=}}$
  when
    $phase = det \wedge s = 1$
  then
    $N := tmp_{max} \parallel tmp_{est} := tmp \parallel cnt := 0 \parallel$
    $phase := cont$
  end
Event $detection_{nok} \mathrel{\widehat{=}}$
  when
    $phase = det \wedge s = 0$
  then
    $N := \mathbf{min}(tmp_{max} - tmp_{est}, tmp_{est} - tmp_{min}) \parallel$
    $cnt := cnt + 1 \parallel phase := cont$
  end

Event $switch_{on} \widehat{=}$
  when
    $phase = cont \wedge st = ok \wedge cnt < N \wedge tmp_{est} \leq tmp_{min}$
  then
    $heat := on \parallel phase := env$
  end
Event $switch_{off} \widehat{=}$
  when
    $phase = cont \wedge st = ok \wedge cnt < N \wedge tmp_{est} \geq tmp_{max}$
  then
    $heat := off \parallel phase := env$
  end
Event $switch_{ok} \widehat{=}$
  when
    $phase = cont \wedge st = ok \wedge cnt < N \wedge$
    $tmp_{min} < tmp_{est} < tmp_{max}$
  then
    $phase := env$
  end
Event $switch_{nok} \widehat{=}$
  when
    $phase = cont \wedge st = ok \wedge cnt \geq N$
  then
    $st := nok$
  end
Event $shutdown \widehat{=}$
  when
    $st = nok$
  then
    $skip$
  end

The variable *tmp* measures the actual temperature of the liquid. We intentionally simplify the environment behaviour by assuming that the temperature increases by 1, when the heater is switched on, and decreases by 1, when it is switched off. The variable s models the sensor status. When s equals 1, the sensor is "healthy" and its value tmp_{est} equals to the actual temperature *tmp*. The value of variable *cnt* corresponds to the number of successive iterations while the sensor has remained faulty, i.e., when the variable s has had value 0. Let us note that we could have merged the events $sensor_{ok}$ and $sensor_{nok}$ into a single event by dropping the second conjuncts from their guards. The combined event would model sensor reading irrespectively whether the sensor has been faulty or healthy. However, we deliberately decided to model them separately here to be able to attach different probabilities while translating this specification to PRISM, thus distinguishing between the cases when the sensor fails and when it recovers.

The faulty detector checks the sensor status and sends its output to the switching mechanism. When no error is detected, the switcher performs the routine control – it either switches the heater on, if the temperature has reached the lower bound, or switches heater off, if the temperature has reached the upper bound, or does nothing, if the value of tmp_{est} is in the allowable limits. However, if the error is detected and the variable *cnt* already reached the value N, then the switcher shutdowns the system.

While translating the Event-B specification into the corresponding PRISM specification, the non-deterministic sensor behaviour is replaced by its probabilistic counterpart. Specifically, we explicitly introduce the constants f and r to model the corresponding probabilities of sensor failure and recovery. The type of the variable *phase* is converted to enumerated integers. In addition, the events modelling system behaviour at each phase are grouped into the corresponding PRISM modules. The justification for this conversion will be given when we will discuss our next refinement step. The PRISM model resulting from this rather straightforward translation is shown below.

dtmc MCH_1
const double $f = 0.01$;
const double $r = 0.9$;
const int $tmp_{min} = 0$;
const int $tmp_{max} = 20$;
const int $tmp_0 = 10$
smallskip **global** *phase* : [0..3] **init** 0;

module *plant*
  $tmp : [tmp_{min}..tmp_{max}]$ **init** tmp_0;
  [] $(phase = 0)\&(s = 1)\&(tmp < tmp_{max}) \rightarrow$
    $(tmp' = tmp + 1)\&(phase' = 1)$;
  [] $(phase = 0)\&(s = 0)\&(tmp > tmp_{min}) \rightarrow$
    $(tmp' = tmp - 1)\&(phase' = 1)$;

endmodule
module *sensor*
  $s : [0..1]$ **init** 1;
  [] $(phase = 1)\&(s = 1) \rightarrow f : (s' = 0)\&(phase' = 2)$
    $+ (1 - f) : (s' = 1)\&(phase' = 2)$;
  [] $(phase = 1)\&(s = 0) \rightarrow r : (s' = 1)\&(phase' = 2)$
    $+ (1 - r) : (s' = 0)\&(phase' = 2)$;

endmodule
module *detection*
  $tmp_{est} : [tmp_{min}..tmp_{max}]$ **init** tmp_0;
  $N : [tmp_{min}..tmp_{max}]$ **init** tmp_{max};
  $cnt : [0..tmp_{max}]$ **init** 0;
  [] $(phase = 2)\&(s = 1) \rightarrow (N' = tmp_{max})\ \&$
    $(cnt' = 0)\&(tmp'_{est} = tmp)\&(phase' = 3)$;
  [] $(phase = 2)\&(s = 0)\&(cnt < tmp_{max}) \rightarrow$
  $(N' = min(tmp_{max} - tmp_{est}, tmp_{est} - tmp_{min}))\ \&$
    $(cnt' = cnt + 1)\&(phase' = 3)$;

endmodule
module *switch*
  $st : [0..1]$ **init** 1;
  $heat : [0..1]$ **init** 1;
  [] $(phase = 3)\&(cnt < N)\&(tmp_{est} = tmp_{max})\ \&$
    $(st = 1) \rightarrow (heat' = 0)\&(phase' = 0)$;
  [] $(phase = 3)\&(cnt < N)\&(tmp_{est} = tmp_{min})\ \&$
    $(st = 1) \rightarrow (heat' = 1)\&(phase' = 0)$;
  [] $(phase = 3)\&(cnt < N)\&(tmp_{est} > tmp_{min})\ \&$
    $(tmp_{est} < tmp_{max})\&(st = 1) \rightarrow (phase' = 0)$;
  [] $(phase = 3)\&(cnt \geq N)\&(st = 1) \rightarrow (st' = 0)$;
  [] $(phase = 3)\&(st = 0) \rightarrow (st' = 0)$;

endmodule

To evaluate the reliability of the system, we notice that our specification makes a clear distinction between the operating

and failed system states. Indeed, in our case the operating states are the states where the variable st has the value ok. Correspondingly, the failed states are the states where the variable st has the value nok. Therefore, the operational states of our systems are defined by the predicate $st = ok$, i.e., $\mathcal{OP} \mathrel{\widehat{=}} (st = ok)$. Then, according to (2), the PRISM property

$$\mathbf{P}_{=?}[\mathbf{G} \leq t\ (st = ok)] \tag{3}$$

denotes the reliability of our systems within the time t. The resulting system reliability is graphically presented in Figure 1 compared against the reliability of a control system that instead of a single sensor employs two sensors in the hot spare arrangement, as we explain next.

The hot spare arrangement is a standard fault tolerance mechanism [22]. We introduce an additional sensor – a spare – that works in parallel with the main one. When a fault is detected, the system automatically switches to read the data produced by a spare sensor.

Introduction of the fault tolerance mechanisms by refinement is a rather standard refinement step often performed in the development of dependable systems. The machine MCH_2 is a result of refining the machine MCH_1 to introduce a hot spare. An excerpt of MCH_2 machine is given below.

Machine MCH_2
...
Invariants $s_1 + s_2 > 0 \Leftrightarrow s = 1 \ \wedge \ s_1 + s_2 = 0 \Leftrightarrow s = 0$
...
Event $sensor_{ok_1} \mathrel{\widehat{=}}$
when
$phase = read \wedge s_1 = 1$
then
$s_1 :\in \{0, 1\} \parallel phase := det$
end
Event $sensor_{nok_1} \mathrel{\widehat{=}}$
when
$phase = read \wedge s_1 = 0$
then
$s_1 :\in \{0, 1\} \parallel phase := det$
end
Event $sensor_{ok_2} \mathrel{\widehat{=}}$
when
$phase = read \wedge s_2 = 1$
then
$s_2 :\in \{0, 1\} \parallel phase := det$
end
Event $sensor_{nok_2} \mathrel{\widehat{=}}$
when
$phase = read \wedge s_2 = 0$
then
$s_2 :\in \{0, 1\} \parallel phase := det$
end
Event $detection_{ok} \mathrel{\widehat{=}}$
when
$phase = det \wedge s_1 + s_2 > 0$
then
...
end
Event $detection_{nok} \mathrel{\widehat{=}}$
when
$phase = det \wedge s_1 + s_2 = 0$
then
...
end
...

In the refined specification, we replace the sensor s by two sensors s_1 and s_2. The behaviour of these sensors is the same as the behaviour of s. The gluing invariant $s_1 + s_2 > 0 \Leftrightarrow s = 1$ describes the refinement relationship between the corresponding variables – the system would output the actual sensor readings only if no more than one sensor has failed. In order to take this into account, we modify the guards of the *detection* events. The events modelling the environment and the switcher are the same as in the MCH_1.

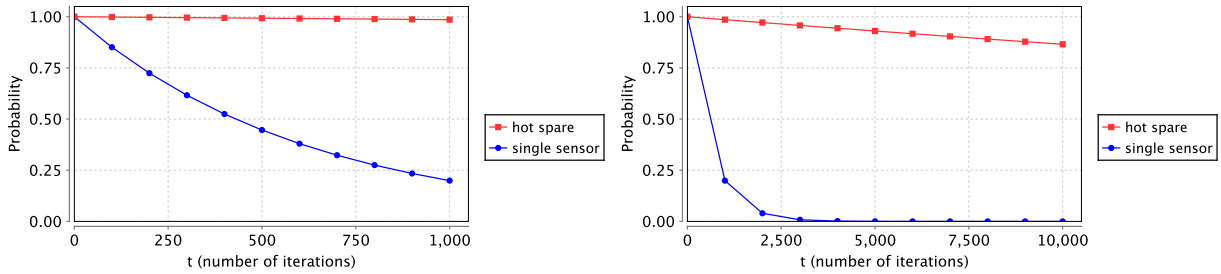
To show that MCH_2 is indeed a refinement of MCH_1, we use the event fusion technique described in Section 2. Namely, we prove that the event $sensor_{nok}$ is refined by parallel composition of the events $(sensor_{nok_1} \parallel sensor_{nok_2})$, while the event $sensor_{ok}$ is refined by any of the following parallel compositions: $(sensor_{ok_1} \parallel sensor_{ok_2})$, $(sensor_{ok_1} \parallel sensor_{nok_2})$ and $(sensor_{nok_1} \parallel sensor_{ok_2})$.

The event fusion technique significantly simplifies translation to PRISM. The behaviour of each sensor is represented by the corresponding module in PRISM. The synchronising guarded commands (labelled ss) are used to model parallel work of sensors. The translation then follows the rules described in Section 3. In the module $sensor_2$ we additionally update the global variable $phase$ to model transition of the system to the detection phase. The modules $sensor_1$ and $sensor_2$ of corresponding PRISM specification are presented below.

dtmc MCH_2
...
module $sensor_1$
$s_1 : [0..1]$ **init** 1;
$[ss]\ (phase = 1)\&(s_1 = 1) \rightarrow$
$f : (s_1' = 0) + (1 - f) : (s_1' = 1);$
$[ss]\ (phase = 1)\&(s_1 = 0) \rightarrow$
$r : (s_1' = 1) + (1 - r) : (s_1' = 0);$
endmodule
module $sensor_2$
$s_2 : [0..1]$ **init** 1;
$sw : bool$ **init** $true$;
$[ss]\ (phase = 1)\&(s_2 = 1)\&(sw) \rightarrow$
$f : (s_2' = 0)\&(sw' = false) +$
$(1 - f) : (s_2' = 1)\&(sw' = false);$
$[ss]\ (phase = 1)\&(s_2 = 0)\&(sw) \rightarrow$
$r : (s_2' = 1)\&(sw' = false) +$
$(1 - r) : (s_2' = 0)\&(sw' = false);$
$[]\ (phase = 1)\&(!sw) \rightarrow (sw' = true)\&(phase' = 2);$
endmodule
...

Let us note that, to model parallel work of sensors, we represent them by the corresponding synchronising modules. In principle, we could have avoided introducing modules until this stage.

In the refined specification the system operational states are still confined by the predicate $st = ok$. Hence, to as-



(a) 1000 iterations

(b) 10000 iterations

Figure 1: Case study results by PRISM ($f = 0.01, r = 0.9, tmp_{min} = 0, tmp_{max} = 20, tmp_0 = 10$)

sess reliability, we again check the property (3). Figure 1 shows the results of evaluation for the more abstract and refined systems. The results clearly demonstrate that the redundant hot spare system always gives a significantly better reliability. In this paper we omit comparison between various fault tolerance mechanisms that could be used in our system. Such a comparison for a similar system can be found in [23].

7. RELATED WORK

The Event-B framework has been extended by Hallerstede and Hoang [10] to take into account probabilistic behaviour. They introduce qualitative probabilistic choice operator to reason about almost certain termination. This operator attempts to bound demonic non-determinism that, for instance, allows us to demonstrate convergence of certain protocols. However, this approach is not suitable for reliability assessment since explicit quantitative representation of reliability is not supported.

Several researches have already used quantitative model checking for dependability evaluation. For instance, Kwiatkowska et al. [13] have proposed an approach to assessing dependability of control systems using continuous time Markov chains. The general idea is similar to ours – to formulate reliability as a system property to be verified. However, this approach aims at assessing reliability of already developed systems. However, dependability evaluation late at development cycle can be perilous and in case of poor results lead to system redevelopment that would mean significant financial and time loss. In our approach reliability assessment proceeds hand-in-hand with the system development by refinement. It allows us to assess dependability of designed system on the early stages of development, for instance, every time when we need to estimate impact of unreliable component on the system reliability level. This allows a developer to make an informed decision about how to guarantee a desired system reliability.

A similar topic in the context of refinement calculus has been explored by Morgan et al. [15, 14]. In this approach the probabilistic refinement has been used to assess system dependability. Such an approach is much stronger than the approach described in this paper. Probabilistic refinement allows the developers to obtain algebraic solutions even without pruning the system state space. Meanwhile, probabilistic verification gives us only numeric solutions for restricted system models. In a certain sense, our approach can be seen as a property-wise refinement evaluation. Indeed, while evaluating dependability, we essentially check that, for the same samples of system parameters, the probability of system to hold a certain property is not decreased by refinement.

8. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed a simple pragmatic approach to quantitative dependability assessment in Event-B. Our approach integrates two frameworks: Event-B and probabilistic model checking. Event-B supported by the RODIN tool platform provides us with a suitable framework for development of complex industrial-size systems. By integrating probabilistic verification supported by the PRISM model checker we open a possibility to reason quantitatively also about non-functional system requirements in the refinement process.

In general continuous-time Markov processes are more often used for dependability evaluation. However, the theory of refinement of systems with continuous behaviour has not reached maturity yet and suffers from poor scalability and lack of tool support [4, 16]. In this paper we have capitalised on similarities between Event-B and PRISM DTMC modelling. Since the Event-B modelling language is richer than the PRISM one, we have shown how to restrict it to achieve compatibility with PRISM. The restrictions are formulated as a number of rules for the syntactic translation. In the future we are planning to define rigorously correspondence between the semantics of these formalisms and define the theory that would enable automatic translation from Event-B to PRISM. As a more challenging task, it would be interesting to extend Event-B with the notion of continuous time and correspondingly enable dependability evaluation using continuous time Markov chains.

Furthermore, in our future work it would be interesting to further explore the connection between Event-B modelling and dependability assessment. In particular, additional studies are required to establish a complete formal semantic basis for converting Event-B models into their probabilistic counterparts. We see our work on probabilistic event traces as the first steps on this road. Furthermore, it would be interesting to explore the topic of probabilistic data refinement in connection with dependability assessment.

9. ACKNOWLEDGMENTS

This work is partially supported by the FP7 IP Deploy Project. We also wish to thank the anonymous reviewers for their helpful comments.

10. REFERENCES

[1] J.-R. Abrial. Extending B without changing it (for developing distributed systems). In H. Habiras, editor, *First Conference on the B method*, pages 169–190. IRIN Institut de recherche en informatique de Nantes, 1996.

[2] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 2005.

[3] R. Alur and T. Henzinger. Reactive modules. In *Formal Methods in System Design*, pages 7–48, 1999.

[4] R. J. R. Back, L. Petre, and I. Porres. Generalizing Action Systems to Hybrid Systems. In *FTRTFT 2000, LNCS 1926*, pages 202–213. Springer, 2000.

[5] M. Butler. Decomposition Structures for Event-B. In *Integrated Formal Methods IFM2009, LNCS 5423*, pages 20–38, 2009.

[6] M. Butler. Incremental Design of Disteributed Systems with Event-B. In *Engineering Methods and Tools for Software Safety and Security*. IOS Press, 2009.

[7] D. Craigen, S. Gerhart, and T.Ralson. Case study: Paris metro signaling system. In *IEEE Software*, pages 32–35, 1994.

[8] EU-project DEPLOY. online at http://www.deploy-project.eu/.

[9] W. Feller. *An Introduction to Probability Theory and its Applications*, volume 1. John Wiley & Sons, 1967.

[10] S. Hallerstede and T. S. Hoang. Qualitative probabilistic modelling in Event-B. In J. Davies and J. Gibbons, editors, *IFM 2007, LNCS 4591*, pages 293–312, 2007.

[11] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. In *Formal Aspects of Computing*, pages 512–535, 1994.

[12] J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. D. Van Nostrand Company, 1960.

[13] M. Kwiatkowska, G. Norman, and D. Parker. Controller dependability analysis by probabilistic model checking. In *Control Engineering Practice*, pages 1427–1434, 2007.

[14] A. K. McIver and C. C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2005.

[15] A. K. McIver, C. C. Morgan, and E. Troubitsyna. The probabilistic steam boiler: a case study in probabilistic data refinement. In *Proc. International Refinement Workshop, ANU, Canberra*. Springer, 1998.

[16] L. Meinicke and G. Smith. A Stepwise Development Process for Reasoning about the Reliability of Real-Time Systems. In *Integrated Formal Methods IFM2007, LNCS 4591*, pages 439–456. Springer, 2007.

[17] P. D. T. O'Connor. *Practical Reliability Engineering, 3rd ed*. John Wiley & Sons, 1995.

[18] PRISM. Probabilistic symbolic model checker. online at http://www.prismmodelchecker.org/.

[19] Rigorous Open Development Environment for Complex Systems (RODIN). IST FP6 STREP project, online at http://rodin.cs.ncl.ac.uk/.

[20] Rigorous Open Development Environment for Complex Systems (RODIN). Deliverable D7, Event-B Language, online at http://rodin.cs.ncl.ac.uk/.

[21] RODIN. Event-B platform. online at http://www.event-b.org/.

[22] N. Storey. *Safety-Critical Computer Systems*. Addison-Wesley, 1996.

[23] A. Tarasyuk, E. Troubitsyna, and L. Laibinis. Reliability assessment in Event-B. Technical Report 932, Turku Centre for Computer Science, 2009.

[24] A. Villemeur. *Reliability, Availability, Maintainability and Safety Assessment*. John Wiley & Sons, 1995.

[25] D. J. White. *Markov Decision Processes*. John Wiley & Sons, 1993.

Paper VII

From Formal Specification in Event-B to Probabilistic Reliability Assessment

Anton Tarasyuk, Elena Troubitsyna and Linas Laibinis

Originally published in: *Proceedings of 3rd International Conference on Dependability (DEPEND 2010)*, 24–31, IEEE Computer Society Press, 2010

2010 Third International Conference on Dependability

From Formal Specification in Event-B to Probabilistic Reliability Assessment

Anton Tarasyuk and Elena Troubitsyna and Linas Laibinis
Department of Information Technologies
Åbo Akademi University
Joukahaisenkatu 3-5A, 20520 Turku, Finland
Email: {anton.tarasyuk, elena.troubitsyna, linas.laibinis}@abo.fi

***Abstract*—Formal methods, in particular the B Method and its extension Event-B, have proven their worth in the development of many complex software-intensive systems. However, while providing us with a powerful development platform, these frameworks poorly support quantitative assessment of dependability attributes. Yet, such an assessment would facilitate not only system certification but also system development by guiding it towards the design optimal from the dependability point of view. In this paper we demonstrate how to integrate reliability assessment performed by model checking into refinement process in Event-B. Such an integration allows us to combine logical reasoning about functional correctness with probabilistic reasoning about reliability. Hence we obtain a method that enables building the systems that are not only correct-by-construction but also have a predicted level of reliability.**

***Keywords*-Reliability assessment; formal modelling; Markov processes; refinement; probabilistic model checking**

I. Introduction

Formal verification techniques provide us with rigorous and powerful methods for establishing correctness of complex systems. The advances in expressiveness, usability and automation of these techniques enable their use in the design of wide range of complex dependable systems. For instance, the B Method [1] and its extension Event-B [2] provide us with a powerful framework for developing systems correct-by-construction. The top-down development paradigm based on stepwise refinement adopted by these frameworks has proven its worth in several industrial projects [3], [4].

While developing system by refinement, we start from an abstract system specification and, in a number of refinement steps, introduce the desired implementation decisions. While approaching the final implementation, we decrease the abstraction level and reduce non-determinism inherently present in the abstract specifications. In general, an abstract specification can be refined in several different ways because usually there are several ways to resolve its non-determinism. These refinement alternatives are equivalent from the correctness point of view, i.e., they faithfully implement functional requirements. Yet they might be different from the point of view of non-functional requirements, e.g., reliability, performance etc. Early quantitative assessment of various design alternatives is certainly useful and desirable. However, within the current refinement frameworks we cannot perform it. In this paper we propose an approach to overcoming this problem.

We propose to integrate stepwise development in Event-B with probabilistic model checking [5] to enable reliability assessment already at the development stage. Reliability is a probability of system to function correctly over a given period of time under a given set of operating conditions [6], [7], [8]. Obviously, to assess reliability of various design alternatives, we need to model their behaviour stochastically. In this paper we demonstrate how to augment (non-deterministic) Event-B models with probabilistic information and then convert them into the form amenable to probabilistic verification. Reliability is expressed as a property that we verify by probabilistic model checking. To illustrate our approach, we assess reliability of refinement alternatives that model different fault tolerance mechanisms.

We believe that our approach can facilitate the process of developing dependable systems by enabling evaluation of design alternatives at early development stages. Moreover, it can also be used to demonstrate that the system adheres to the desired dependability levels, for instance, by proving statistically that the probability of a catastrophic failure is acceptably low. This application is especially useful for certifying safety-critical systems.

The remainder of the paper is structured as follows. In Section II, we give a brief overview of our modelling formalism – the Event-B framework. In Section III, we give an example of refinement in Event-B. In Section IV, we demonstrate how to augment Event-B specifications with probabilistic information and convert them into specifications of the PRISM model checker [9]. In Section V, we define how to assess reliability via probabilistic verification and compare the results obtained by model checking with algebraic solutions. Finally, in Section VI, we discuss the obtained results, overview the related work and propose some directions for the future work.

II. Modelling and Refinement in Event-B

The B Method is an approach for the industrial development of highly dependable software. Event-B is an extension of the B Method to model parallel, distributed and reactive systems. The Rodin platform [10] provides automated tool support for modelling and verification (by theorem proving)

978-0-7695-4090-0/10 $26.00 © 2010 IEEE
DOI 10.1109/DEPEND.2010.12

in Event-B. Currently Event-B is used in the EU project Deploy [11] to model several industrial systems from automotive, railway, space and business domains.

Event-B uses the Abstract Machine Notation [12] for constructing and verifying system models. An abstract machine encapsulates the state (the variables) of a model and defines operations on its state. A simple abstract machine has the following general form:

Machine AM
Variables v
Invariants I
Events
$init$
evt_1
$\cdots$
evt_N

The machine is uniquely identified by its name AM. The state variables of the machine, v, are declared in the **Variables** clause and initialised in $init$ event. The variables are strongly typed by constraining predicates of invariants I given in the **Invariants** clause. The invariant is usually defined as a conjunction predicates and the predicates defining the properties of the system that should be preserved during system execution.

The dynamic behaviour of the system is defined by the set of atomic events specified in the **Events** clause. An event is defined as follows:

$$evt \mathrel{\widehat{=}} \textbf{when } g \textbf{ then } S \textbf{ end}$$

where the guard g is conjunction of predicates over the state variables v, and the action S is an assignment to the state variables.

The guard defines the conditions under which the action can be executed, i.e., when the event is *enabled*. If several events are enabled then any of them can be chosen for execution non-deterministically. If none of the events is enabled then the system deadlocks.

In general, the action of an event is a composition of variable assignments executed simultaneously (simultaneous execution is denoted as $\|$). Variable assignments can be either deterministic or non-deterministic. The deterministic assignment is denoted as $x := E(v)$, where x is a state variable and $E(v)$ expression over the state variables v. The non-deterministic assignment can be denoted as $x :\in S$ or $x :| \ Q(v, x')$, where S is a set of values and $Q(v, x')$ is a predicate. As a result of non-deterministic assignment, x gets any value from S or it obtains such a value x' that $Q(v, x')$ is satisfied.

The semantics of Event-B events is defined using so called before-after predicates [12]. It is a variation of the weakest precondition semantics [13]. A before-after predicate describes a relationship between the system states before and after execution of an event. The formal semantics provides us with a foundation for establishing correctness of Event-B specifications. To verify correctness of a specification we need to prove that its initialization and all events preserve the invariant.

The formal semantics also establishes a basis for system refinement – the process of developing systems correct by construction. The basic idea underlying formal stepwise development by refinement is to design the system implementation gradually, by a number of correctness preserving steps, called *refinements*. The refinement process starts from creating an abstract, albeit unimplementable, specification and finishes with generating executable code. The intermediate stages yield the specifications containing a mixture of abstract mathematical constructs and executable programming artifacts.

Assume that the refinement machine AM' is a result of refinement of the abstract machine AM:

Machine AM **Variables** v **Invariants** I **Events** $init$ evt_1 $\cdots$ evt_N	$\sqsubseteq$	**Machine** AM' **Variables** v' **Invariants** I' **Events** $init'$ evt'_1 $\cdots$ evt'_K

The machine AM' might contain new variables and events as well as replace the abstract data structures of *AM* with the concrete ones. The invariants of AM' – I' – define not only the invariant properties of the refined model, but also the connection between the state spaces of AM and AM'. For a refinement step to be valid, every possible execution of the refined machine must correspond (via I') to some execution of the abstract machine. To demonstrate this, we should prove that $init'$ is a valid refinement of $init$, each event of AM' is a valid refinement of its counterpart in AM and that the refined specification does not introduce additional deadlocks. In the next section we illustrate modelling and refinement in Event-B by an example.

III. Example of Refinement in Event-B

Control and monitoring systems constitute a large class of dependable systems. Essentially, the behaviour of these systems is periodic. Indeed, a control system periodically executes a control cycle that consists of reading sensors and setting actuators. The monitoring systems periodically perform certain measurements. Due to faults (e.g., caused by random hardware failures) inevitably present in any system, the system can fail to perform its functions. In this paper we focus on modelling fail-safe systems, i.e., the systems that shut down upon occurrence of failure.

In general, the behaviour of such system can be represented as shown in the specification below.

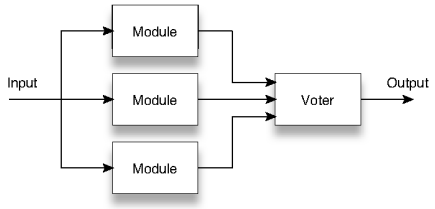


Figure 1. A Triple Modular Redundancy Arrangement

Machine $System$
Variables res
Invariants
$inv_1 : res \in BOOL$
Events
$init \mathrel{\widehat{=}}$
begin
$res := TRUE$
end
$output \mathrel{\widehat{=}}$
when
$res = TRUE$
then
$res :\in BOOL$
end

For the sake of simplicity, we omit the detailed modelling of the system functionality. The variable res abstractly models success or failure to perform the required functions at each iteration. Each iteration of the system corresponds to the execution of the event $output$. If no failure has occurred then, as a result of the non-deterministic assignment, the variable res obtains the value $TRUE$. In this case the next iteration can be executed. However, if a failure has occurred then res obtains the value $FALSE$ and the system deadlocks.

In the initial specification, we have deliberately abstracted away from modelling system components and their failures. In the next refinement step we introduce explicit representation of system components and introduce fault tolerance mechanisms. These mechanisms allow the system to perform its functions even in the presence of certain faults [6]. Fault tolerance is usually achieved by introducing redundancy into the system design. The redundancy can be either static or dynamic. When static redundancy is used, the redundant components work in parallel the main ones. In dynamic redundancy activation of the redundant components occurs only after the main ones have failed.

Refining a system by introducing the fault tolerance mechanisms is a rather standard model transformation frequently performed in the development of dependable systems. Next we show by examples how to introduce various fault tolerance mechanisms by refinement.

Triple Modular Redundancy (TMR) [6] is a well-known mechanism based on static redundancy. The general principle is to triplicate a system module and introduce the majority voting to obtain a single result of the module, as shown in Figure 1. Such an arrangement allows us to mask failures of a single module. TMR can be introduced into a system specification by refinement it as explained below. We introduce variables m_1, m_2 and m_3 to model the results produced by the redundant modules. The variable $phase$ models the phases of TMR execution – first reading the results produced by the modules and then voting.

Machine $System_{TMR}$
Variables
$res, m_1, m_2, m_3, phase, flag_1, flag_2, flag_3$
Invariants
$inv_{1..3} : m_1, m_2, m_3 \in \{0, 1\}$
$inv_4 : phase \in \{reading, voting\}$
$inv_{5..7} : flag_1, flag_2, flag_3 \in \{0, 1\}$
$inv_6 : m_1 + m_2 + m_3 > 1 \Rightarrow res = TRUE$
$\ldots$
$module_{ok_1} \mathrel{\widehat{=}}$
when
$m_1 = 1 \wedge flag_1 = 1 \wedge phase = reading$
then
$m_1 :\in \{0, 1\} \parallel flag_1 := 0$
end
$module_{failed_1} \mathrel{\widehat{=}}$
when
$m_1 = 0 \wedge flag_1 = 1 \wedge phase = reading$
then
$flag_1 := 0$
end
$\ldots$
$synchr \mathrel{\widehat{=}}$
when
$flag_1 = 0 \wedge flag_1 = 0 \wedge flag_3 = 0 \ \wedge phase = reading$
then
$phase := voting$
end
$\ldots$
$voter_{ok} \mathrel{\widehat{=}}$
refines output
when
$res = TRUE \wedge phase = voting \ \wedge m_1 + m_2 + m_3 > 1$
then
$phase := reading \parallel flag_1, flag_2, flag_3 := 1, 1, 1$
end
$voter_{nok} \mathrel{\widehat{=}}$
refines output
when
$res = TRUE \wedge phase = voting \ \wedge m_1 + m_2 + m_3 \leq 1$
then
$res := FALSE$
end

The modules work in parallel. In our specification it is reflected by the fact that all the events modelling module behaviour are enabled simultaneously. Each event disables itself after being executed once. When all the modules complete their execution, the event $synchr$ enables the events modelling voting. Let us observe that the invariant

$$m_1 + m_2 + m_3 > 1 \Rightarrow res = TRUE$$

relates the abstract and refined systems, i.e, it requires that the correct output can be produced only if no more than one module has failed.

In general, we can introduce any fault tolerance mechanism by refinement. Below we show other alternatives.

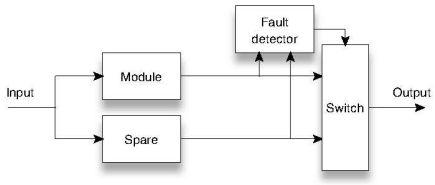


Figure 2. A Standby Spare Arrangement

For instance, instead of the TMR arrangement we can introduce a standby spare mechanism shown in Figure 2. In this mechanism, every result produced by an active (main) module is checked by a fault detector. If an error is detected then the result produced by the failed module is ignored and the system switches to accepting the results produced by the spare. The spare can be *hot* meaning that the main module and spare work in parallel. In this case the switch to spare happens almost instantly. The spare also can be *cold*, i.e., the spare is in the standby mode and is activated only after the main module fails.

Below we present another refinement alternative – the specification that refines *System* to model dynamic redundancy. Here the values in and out of the variable $phase$ correspond to the values $reading$ and $voting$ in the TMR specification. The additional execution phase det is introduced to model failure detection. The events that model the behaviour at this phase for the hot spare arrangement are presented below.

$detection_{ok_1} \mathrel{\widehat{=}}$
when
$m_1 = 1 \wedge phase = det$
then
$phase := out \parallel m := m_1 \parallel flag_1, flag_2 := 1, 1$
end
$detection_{ok_2} \mathrel{\widehat{=}}$
when
$m_1 = 0 \wedge m_2 = 1 \wedge phase = det$
then
$phase := out \parallel m := m_2 \parallel flag_2 := 1$
end
$detection_{nok} \mathrel{\widehat{=}}$
when
$m_1 = 0 \wedge m_2 = 0 \wedge phase = det$
then
$phase := out \parallel m := 0$
end
...

The output can be produced successfully if at least one module functions correctly. If an error is detected then the system switches the failed module off.

We can also introduce a hybrid arrangement, which combines static and dynamic redundancy, as shown in Figure 3. The system works as TMR until a failure of a module occurs. Then the system activates the spare to "replace" the failed module. The full Event-B specifications of this and the previous arrangements can be found in [14].

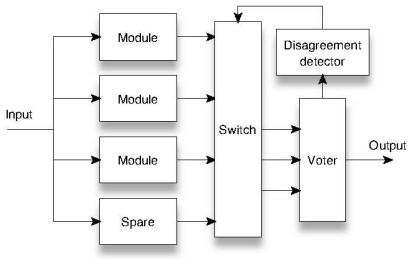


Figure 3. TMR with a Spare Arrangement

Let us observe that any specification described above is a valid refinement of our abstract specification $System$. However, even though the fault tolerance mechanisms were introduced to increase system reliability, we cannot evaluate which of the specifications is more optimal from the point of view of reliability. This problem is caused by the non-deterministic modelling of the failure occurrence – the only possible modelling currently available in Event-B. To evaluate reliability, we need to replace the non-deterministic modelling of failure occurrence by the probabilistic ones and use the suitable techniques for reliability evaluation. Next we present our approach for achieving this.

IV. From Event-B Modelling to Probabilistic Model Checking

To enable formal, probabilistic analysis of reliability in Event-B we can choose several options. The first and the most powerful is to rely on probabilistic weakest pre-condition semantics [15] and use probabilistic refinement technique [16] to evaluate reliability. This technique allows us to express algebraically the reliability of the system as a function of reliabilities of its components. However, for complex industrial-size systems finding this function might be very complex or even analytically intractable. A simpler and more scalable solution is to use probabilistic model checking to obtain numeric solution. To achieve this we need to augment Event-B models with probabilities in such way that they would become amenable for probabilistic verification. Then we need to establish connection between probabilistic verification and reliability assessment.

To tackle the first problem let us observe that Event-B is a state-based formalism. The state space of the system specified in Event-B is formed by the values of the state variables. The transitions between states are determined by the actions of the system events. The states that can be reached as a result of event execution are defined by the current state. If we augment Event-B specification with the probabilities of reaching the next system state from the current one then we obtain a probabilistic automaton [17]. In case the events are mutually exclusive, i.e., only one event is enabled at each system state then the specification can be represented

by a Markov chain. Otherwise, it corresponds to a Markov Decision process [18], [19], [20]. More specifically, it is a discrete time Markov process since we can use it to describe the states at certain instances of time.

The probabilistic model checking framework developed by Kwiatkowska et al. [5] supports verification of Discrete-Time Markov Chains (DTMC) and Markov Decision Processes (MDP). The framework has a mature tool support – the PRISM model checker [9]

The PRISM modelling language is a high-level state-based language. It relies on the Reactive Modules formalism of Alur and Henzinger [17]. PRISM supports the use of constants and variables that can be integers, doubles (real numbers) and Booleans. Constants are used, for instance, to define the probabilities associated with variable updates. The variables in PRISM are finite-ranged and strongly typed. They can be either local or global. The definition of an initial value of a variable is usually attached to its declaration. The state space of a PRISM model is defined by the set of all variables, both global and local.

In general, a PRISM specification looks as follows:

dtmc
const double $p_{11} = \ldots;$
$\ldots$
global $v : Type$ **init** $\ldots;$
module M_1
$v_1 : Type$ **init** $\ldots;$
$[]\ g_{11} \rightarrow p_{11} : act_{11} + \cdots + p_{1n} : act_{1n};$
$[sync]\ g_{12} \rightarrow q_{11} : act'_{11} + \cdots + q_{1m} : act'_{1m};$
$\ldots$
endmodule
module M_2
$v_2 : Type$ **init** $\ldots;$
$[sync]\ g_{21} \rightarrow p_{21} : act_{21} + \cdots + p_{2k} : act_{2k};$
$[]\ g_{22} \rightarrow q_{21} : act'_{21} + \cdots + q_{2l} : act'_{2l};$
$\ldots$
endmodule

A system specification in PRISM is constructed as a parallel composition of modules. Modules work in parallel. They can be independent of each other or interact with each other. Each module has a number of local variables $v_1,\ v_2$ and a set of guarded commands that determine its dynamic behaviour. The guarded commands can have names. Similarly to the events of Event-B, a guarded command can be executed if its guard evaluates to $TRUE$. If several guarded commands are enabled then the choice between them can be non-deterministic in case of MDP or probabilistic (according to the uniform distribution) in case of DTMC. In general, the body of a guarded command is a probabilistic choice between deterministic assignments.

The guarded commands define not only the dynamic behaviour of a stand-alone module but can also be used to define syncronisation between modules. If several modules synchronise then each of them should contain a command defining the syncronisation condition. These commands should have identical names. For instance, in our general PRISM specification shown above, the modules M_1 and M_2 synchronise. They contain the corresponding guarded commands labelled with the name $sync$. The guarded commands that provide synchronisation with other modules cannot modify the global variables. This allows to avoid read-write and write-write conflicts on the global variables.

With certain restrictions on the Event-B modelling language, converting of an Event-B specification into a PRISM model is rather straightforward. When converting the Event-B model into its counterpart, we need to restrict the types of variables and constants to the types supported by PRISM. Moreover, PRISM lacks of support relations and functions. The invariants that describe system properties can be represented as a number of temporal logic formulas in a list of properties of the model and then can be verified by PRISM if needed. While converting events into the PRISM guarded commands, we identify four classes of events: initilisation events, events with parallel deterministic assignment, non-deterministic assignment and parallel non-deterministic assignment. The conversion of an Event-B event to a PRISM guarded command depends on its class:

- The initialisation events are used to form the initialisation part of the corresponding variable declaration. Hence the initialisation does not have a corresponding guarded command in PRISM;
- An event with a parallel deterministic assignment

  $$evt \mathrel{\widehat{=}} \textbf{when}\ g\ \textbf{then}\ x := x_1 \parallel y := y_1\ \textbf{end}$$

  can be represented by the following guarded command in PRISM:

  $$[]\ g \rightarrow (x' = x_1)\ \&\ (y' = y_1)$$

  Here $\&$ denotes the parallel composition;
- An event with a non-deterministic assignment

  $$evt \mathrel{\widehat{=}} \textbf{when}\ g\ \textbf{then}\ x :\in \{x_1, \ldots x_n\}\ \textbf{end}$$

  can be represented as

  $$[]\ g \rightarrow p_1 : (x' = x_1) + \cdots + p_n : (x' = x_n)$$

  where $p_1, ..., p_n$ are defined according to a certain probability distribution;
- An event with a parallel non-deterministic assignment

  $$evt \mathrel{\widehat{=}} \textbf{when}\ g\ \textbf{then}\ x :\in \{x_1, \ldots x_n\} \parallel y :\in \{y_1, \ldots y_m\}\ \textbf{end}$$

  can be represented using the PRISM synchronisation mechanism. It corresponds to a set of the guarded commands modelling syncronisation. These commands have the identical guards. Their bodies are formed from the assignments used in the parallel composition of the Event-B action.

module X

$x : Type$ **init** $\ldots$;

$[name]\ g \rightarrow p_1 : (x' = x_1) + \cdots + p_n : (x' = x_n)$;

endmodule
module Y

$y : Type$ **init** $\ldots$;

$[name]\ g \rightarrow q_1 : (y' = y_1) + \cdots + q_m : (y' = y_m)$;

endmodule.

To demonstrate the conversion of an Event-B specification into a PRISM specification, below we present an excerpt from the PRISM counterpart of the TMR specification. Here we assume that at each iteration step a module successfully produces a result with a constant probability p.

$System_{TMR}$
module $module_1$
$m_1 : [0..1]$ **init** 1;
$f : [0..1]$ **init** 0;

$[m]\ (phase = 0)\ \&\ (m_1 = 1)\ \&\ (f = 0) \rightarrow$
$p : (m_1' = 1)\ \&\ (f' = 1) +$
$(1 - p) : (m_1' = 0)\ \&\ (f' = 1)$;

$[m]\ (phase = 0)\ \&\ (m_1 = 0)\ \&\ (f = 0) \rightarrow (f' = 1)$;

$[]\ (phase = 0)\ \&\ (f = 1) \rightarrow (phase' = 1)\ \&\ (f' = 0)$;

endmodule
module $module_2$ $\ldots$
module $module_3$ $\ldots$
module $voter$
res : ***bool*** **init** $true$;

$[]\ (phase = 1)\&(m_1{+}m_2{+}m_3 > 1) \rightarrow (phase' = 0)$;

$[]\ (phase = 1)\&(m_1{+}m_2{+}m_3 \leq 1) \rightarrow (res' = false)$;

endmodule

While converting an Event-B model into PRISM we could have modelled the parallel work of the system modules in the same way as we have done it in the Event-B specifications, i.e., using non-determinism to represent parallel behaviour and explicitly modelling the phases of system execution. However, we can also directly use the synchronisation mechanism of PRISM because all the modules update only their local variables and no read-write conflict can occur. This solution is presented in the excerpt above. In the $System_{TMR}$ specification, the guarded commands of the modules $module_1$, $module_2$ and $module_3$ are synchronised (as designated by the m label). In the $module_1$ we additionally update the global variable $phase$ to model transition of the system to the voting phase.

V. Reliability Assessment via Probabilistic Model Checking

In engineering, reliability [7], [8] is generally measured by the probability that an entity $\mathcal{E}$ can perform a required function under given conditions for the time interval $[0, t]$:

$$R(t) = P[\mathcal{E} \textit{ not failed over time } [0, t]].$$

The analysis of the abstract and refined specification shows that we can clearly distinguish between two classes of systems states: operating and failed. In our case the operating states are the states where the variable res has the value $TRUE$. Correspondingly, the failed states are the states where the variable res has the value $FALSE$. While the system is in an operating state, it continues to iterate. When the system fails it deadlocks. Therefore, we define *reliability of the system as a probability of staying operational for a given number of iterations*.

Let $\mathcal{T}$ be the random variable measuring the number of iterations before the deadlock is reached and $F(t)$ its cumulative distribution function. Then clearly $R(t)$ and $F(t)$ are related as follows:

$$R(t) = P[\mathcal{T} > t] = 1 - P[\mathcal{T} \leq t] = 1 - F(t).$$

It is straightforward to see that our definition corresponds to the standard definition of reliability given above. Now let us discuss how to employ PRISM model checking to assess system reliability.

While analysing a PRISM model we define a number of temporal logic properties and systematically check the model to verify them. Properties of discrete-time PRISM models, i.e, DTMC and MDP, are expressed formally in the probabilistic computational tree logic [21]. The PRISM property specification language supports a number of different types of properties. For example, the $\mathbf{P}$ operator is used to refer to the probability of a certain event occurrence.

Since we are interested in assessment of system reliability, we have to verify *invariant* properties, i.e., properties maintained by the system globally. In the PRISM property specification language, the operator $\mathbf{G}$ is used inside the operator $\mathbf{P}$ to express properties of such type. In general, the property

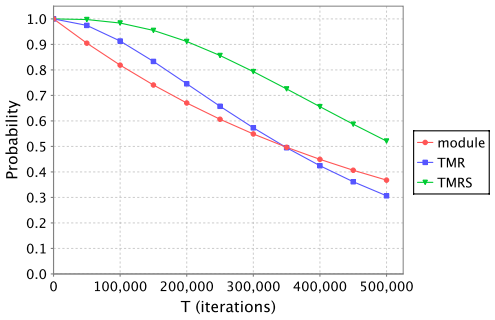
$$\mathbf{P}_{=?}[\mathbf{G} \leq t\ prop]$$

returns a probability that the predicate $prop$ remains $TRUE$ in all states within the period of time t.

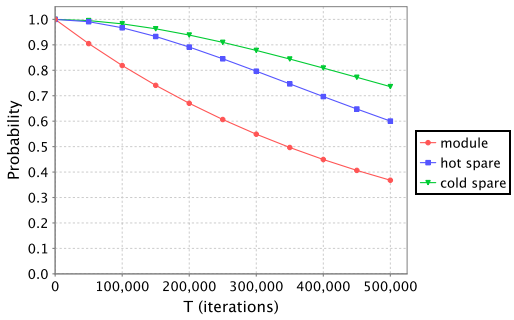
To evaluate reliability of a system we have to assess a probability of system staying operational within time t. We define a predicate $\mathcal{OP}$ that defines a subset of all system states where the system is operational. Then, the PRISM property

$$\mathbf{P}_{=?}[\mathbf{G} \leq T\ \mathcal{OP}] \tag{1}$$

gives us the probability that the system will stay operational during the first T iterations, i.e, it is a probability that



(a) Resulting Reliabilities



(b) Resulting Reliabilities

Figure 4. Case Study Results by PRISM

any state in which the system will be during this time belongs to the subset of operational states. In other words, the property (1) defines the reliability function of the system.

Let us return to our examples. As we discussed previously, the operational states of our systems are defined by the predicate $res = true$, i.e., $\mathcal{OP} \widehat{=} res = true$. Then the PRISM property

$$\mathbf{P}_{=?}[\mathbf{G} \leq T\ (res = true)] \tag{2}$$

denotes the reliability of our systems within time T.

To evaluate reliability of our refinement system, let us assume that a module produces a result successfully with the probability p equal to 0.999998. In Figure 4 we present the results of analysis of reliability up to 500000 iterations. Figure 4 (a) shows the comparative results between single-module and both of TMR systems. The results show that the triple modular redundant system with a spare always gives better reliability. Note that using the simple TMR arrangement is better comparing to a single module only up to approximately 350000 iterations. In Figure 4 (b) we compare single-module and standby spare arrangements. The results clearly indicate that the better reliability is provided by the dynamic redundancy systems and that using of the cold spare arrangement is always more reliable.

It would be interesting to evaluate precision of the results obtained by the model checking with PRISM. For our case study it is possible to derive analytical representations of reliability functions, which then can be used for comparison with verification results of property (2). It is well-known that the reliability of a single module system is $R_M(t) = p^t$ and it is easy to show that the reliability of a TMR system, consists of three identical modules, is

$$R_{TMR}(t) = R_M^3(t) + 3R_M^2(t)(1 - R_M(t)) = \\ = 3R_M^2(t) - 2R_M^3(t) = 3p^{2t} - 2p^{3t}.$$

Indeed, we can also calculate that the standby spare arrangement with a faulty detector has the resulting reliability

$$R_{HSS} = 1 - (1 - p^t)^2$$

for the hot spare, and the resulting reliabilty

$$R_{CSS} = p^t(1 + t(1 - p))$$

for the cold spare module. Finally, for the TMR arrangement with a spare, the resulting reliability is given by the expression

$$R_{TMRS} = (6t - 8)p^{3t} - 6tp^{3t-1} + 9p^{2t}.$$

It is easy to verify that the results obtained by the model checking are identical to those can be calculated from the formulas presented above. This fact demonstrates the feasibility of using the PRISM model checker for reliability assessment.

VI. Conclusion

In this paper, we have proposed an approach to integrating reliability assessment into the refinement process. The proposed approach enables reliability assessment at early design phases that allows the designers to evaluate reliability of different design alternatives already at the development phase.

Our approach integrates two frameworks: refinement in Event-B and probabilistic model checking. Event-B supported by the RODIN tool platform provides us with a suitable framework for development of complex industrial-size systems. By integrating probabilistic verification supported by PRISM model checker we open a possibility to reason about non-functional system requirements in the refinement process.

The Event-B framework has been extended by Hallerstede and Hoang [22] to take into account probabilistic behaviour. They introduce qualitative probabilistic choice operator to reason about almost certain termination. This operator attempts to bound demonic non-determinism that, for instance, allows to demonstrate convergence of certain protocols. However, this approach is not suitable for reliability assessment since explicit quantitative representation of reliability would not be supported.

Kwiatkowska et al. [5] proposed an approach to assessing dependability of control systems using continuous time Markov chains. The general idea is similar to ours – to formulate reliability as a system property to be verified. However, this approach aims at assessing reliability of already developed system. In our approach reliability assessment proceeds hand-in-hand with system development.

The similar topic in the context of refinement calculus has been explored previously by Morgan et al. [16], [15]. In this approach the probabilistic refinement was used to assess system dependability. However, this work does not have the corresponding tool support, so the use of this approach in industrial practice might be cumbersome. In our approach we see a great benefit in integrating frameworks that have mature tool support [10], [9].

When using model checking we need to validate whether the analysed model represents the behaviour of the real system accurately enough. For example, the validation can be done if we demonstrate that model checking provides a good approximation of the corresponding algebraic solutions. In this paper we deliberately chosen the examples for which algebraic solutions can be provided. The experiments have demonstrated that the results obtained by model checking accurately match the algebraic solutions.

In our future work it would be interesting to further explore the connection between Event-B modeling and dependability assessment. In particular, an additional study are required to establish a formal basis for converting all types of non-deterministic assignments into the probabilistic ones. Furthermore, it would be interesting to explore the topic of probabilistic data refinement in connection with dependability assessment.

ACKNOWLEDGMENT

This work is partially supported by the FP7 IP Deploy.

REFERENCES

[1] J.-R. Abrial, *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 2005.

[2] J.-R. Abrial, "Extending B without changing it (for developing distributed systems)," in *First Conference on the B method*, H. Habiras, Ed. IRIN Institut de recherche en informatique de Nantes, 1996, pp. 169–190.

[3] Rigorous Open Development Environment for Complex Systems (RODIN), IST FP6 STREP project, online at http://rodin.cs.ncl.ac.uk/.

[4] D. Craigen, S. Gerhart, and T. Ralson, "Case study: Paris metro signaling system," in *IEEE Software*, 1994, pp. 32–35.

[5] M. Kwiatkowska, G. Norman, and D. Parker, "Controller dependability analysis by probabilistic model checking," in *Control Engineering Practice*, 2007, pp. 1427–1434.

[6] N. Storey, *Safety-Critical Computer Systems*. Addison-Wesley, 1996.

[7] A. Villemeur, *Reliability, Availability, Maintainability and Safety Assessment*. John Wiley & Sons, 1995.

[8] P. D. T. O'Connor, *Practical Reliability Engineering, 3rd ed*. John Wiley & Sons, 1995.

[9] PRISM. Probabilistic symbolic model checker, online at http://www.prismmodelchecker.org/.

[10] RODIN. Event-B platform, online at http://www.event-b.org/.

[11] EU-project DEPLOY, online at http://www.deploy-project.eu/.

[12] Rigorous Open Development Environment for Complex Systems (RODIN), Deliverable D7, Event-B Language, online at http://rodin.cs.ncl.ac.uk/.

[13] E. W. Dijkstra, *A Discipline of Programming*. Prentice-Hall, 1976.

[14] A. Tarasyuk, E. Troubitsyna, and L. Laibinis, "Reliability assessment in Event-B," Turku Centre for Computer Science, Tech. Rep. 932, 2009.

[15] A. K. McIver and C. C. Morgan, *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2005.

[16] A. K. McIver, C. C. Morgan, and E. Troubitsyna, "The probabilistic steam boiler: a case study in probabilistic data refinement," in *Proc. International Refinement Workshop, ANU, Canberra*, J. Grundy, M. Schwenke, and T. Vickers, Eds. Springer-Verlag, 1998.

[17] R. Alur and T. Henzinger, "Reactive modules," in *Formal Methods in System Design*, 1999, pp. 7–48.

[18] W. Feller, *An Introduction to Probability Theory and its Applications*. John Wiley & Sons, 1967, vol. 1.

[19] J. G. Kemeny and J. L. Snell, *Finite Markov Chains*. D. Van Nostrand Company, 1960.

[20] D. J. White, *Markov Decision Processes*. John Wiley & Sons, 1993.

[21] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," in *Formal Aspects of Computing*, 1994, pp. 512–535.

[22] S. Hallerstede and T. S. Hoang, "Qualitative probabilistic modelling in Event-B," in *IFM 2007, LNCS 4591*, J. Davies and J. Gibbons, Eds., 2007, pp. 293–312.

Paper VIII

Quantitative Reasoning about Dependability in Event-B: Probabilistic Model Checking Approach

Anton Tarasyuk, Elena Troubitsyna and Linas Laibinis

Originally published in: Luigia Petre, Kaisa Sere, Elena Troubitsyna (Eds.), *Dependability and Computer Engineering: Concepts for Software-Intensive Systems*, 459–472, IGI Global, 2012

Chapter 19
Quantitative Reasoning About Dependability in Event-B:
Probabilistic Model Checking Approach

Anton Tarasyuk
Åbo Akademi University, Finland & Turku Centre for Computer Science, Finland

Elena Troubitsyna
Åbo Akademi University, Finland

Linas Laibinis
Åbo Akademi University, Finland

ABSTRACT

Formal refinement-based approaches have proved their worth in verifying system correctness. Often, besides ensuring functional correctness, we also need to quantitatively demonstrate that the desired level of dependability is achieved. However, the existing refinement-based frameworks do not provide sufficient support for quantitative reasoning. In this chapter, we show how to use probabilistic model checking to verify probabilistic refinement of Event-B models. Such integration allows us to combine logical reasoning about functional correctness with probabilistic reasoning about reliability.

INTRODUCTION

Formal approaches provide us with rigorous methods for establishing correctness of complex systems. The advances in expressiveness, usability and automation offered by these approaches enable their use in the design of wide range of complex dependable systems. For instance, Event-B (Abrial, 1996; Abrial, 2010) provides us with a powerful framework for developing systems correct-by-construction. The top-down development paradigm based on stepwise refinement adopted by Event-B has proved its worth in several industrial projects (Craigen, Gerhart, & Ralson, 1994; RODIN: IST FP6 project, 2004).

While developing system by refinement, we start from an abstract system specification and, in a number of correctness-preserving refinement steps, implement the system's functional requirements. In our recent work (Tarasyuk, Troubitsyna,

DOI: 10.4018/978-1-60960-747-0.ch019

& Laibinis, 2010) we have extended the Event-B modelling language with probabilistic assignment. Moreover, we have strengthened the notion of Event-B refinement by additionally requiring that the refined model would be more *reliable*. However, while the Event-B framework provides us with a powerful development platform (RODIN Platform), quantitative assessment of non-functional system requirements is sorely lacking. In this chapter we demonstrate how to overcome this problem. Specifically, we show how the Event-B development process can be complemented by probabilistic model checking to ensure the correctness of probabilistic refinement. We exemplify our approach by refinement and reliability evaluation of a simple monitoring system.

The remainder of the chapter is structured as follows. We start with a short introduction into our modelling formalism – the Event-B framework. We continue by briefly overviewing our approach to probabilistic modelling in Event-B. Next, we explain how probabilistic verification of Event-B models can be done using the PRISM symbolic model checker and also summarise our approach proposing a number of modelling guidelines. In the last two sections we exemplify our approach by presenting a case study and give concluding remarks, respectively.

INTRODUCTION TO EVENT-B

The B method (Abrial, 1996) is an approach for the industrial development of highly dependable software. The method has been successfully used in the development of several complex real-life applications. Event-B is a formal framework derived from the B Method to model parallel, distributed and reactive systems. The Rodin platform provides automated tool support for modelling and verification (by theorem proving) in Event-B. Currently Event-B is used in the EU project Deploy (DEPLOY: IST FP7 project, 2008) to model several industrial systems from automotive, railway, space and business domains.

Figure 1. An Event-B machine

Machine M
Variables v
Invariants I
Events
$init$
evt_1
$\cdots$
evt_N

In Event-B a system specification is defined using an abstract (state) machine notion. An abstract machine encapsulates the state (the variables) of a model and defines operations on its state. The general form of an Event-B machine is shown on Figure 1.

The machine is uniquely identified by its name *M*. The state variables, *v*, are declared in the Variables clause and initialised in the *init* event. The variables are strongly typed by the constraining predicates *I* given in the Invariants clause. The invariant clause might also contain other predicates defining properties that should be preserved during system execution.

The dynamic behaviour of the system is defined by the set of atomic events specified in the Events clause. Generally, an event can be defined as follows:

evt ≙ **when** *g* **then** *S* **end**,

where the guard *g* is a conjunction of predicates over the state variables *v* and the action *S* is an assignment to the state variables. In its general form, an event can also have local variables as well as parameters. The guard defines the conditions under which the action can be executed, i.e., when the event is *enabled*. If several events are enabled at the same time, any of them can be chosen for execution non-deterministically. If none of the events is enabled then the system deadlocks.

In general, the action of an event is a parallel composition of assignments. The assignments can be either deterministic or non-deterministic. A deterministic assignment, $x:=E(x,y)$, has the standard syntax and meaning. A nondeterministic assignment is denoted either as $x:\in Q$, where Q is a set of values, or $x:|(x,y,x')$, where P is a predicate relating initial values of x and y to some final value of x'. As a result of such a nondeterministic assignment, x can get any value belonging to Q or according to P. The choice of this value is considered *demonic*, i.e., we do not have any control over it.

Event-B employs top-down refinement-based approach to system development. Development starts from an abstract system specification that models the most essential functional requirements. While capturing more detailed requirements, each refinement step typically introduces new events and variables into the abstract specification. These new events correspond to stuttering steps that are not visible at the abstract level. By verifying correctness of refinement, we ensure that all invariant properties of (more) abstract machines are preserved. A detailed description of formal semantics of Event-B and foundations of the verification process can be found in (Abrial, 2010).

STOCHASTIC MODELING IN EVENT-B

In general, while refining a system in Event-B, we gradually introduce certain design decisions into the system specification. Sometimes there are several refinement alternatives that can adequately implement a certain functional requirement. These alternatives can have different impact on non-functional system requirements, such as dependability, performance etc. Obviously, it would be advantageous to evaluate this impact already at the development stage to ensure that the most optimal solutions are chosen. To achieve this, we would need to perform quantitative analysis of, e.g., system dependability. In this chapter we focus on assessment of the system reliability as one of the most important dependability attributes. To assess system reliability, it is necessary to evaluate the *probability* of system functioning correctly over the given period of time. Hence, we need to integrate the notion of probability into the Event-B modelling framework.

In our recent work (Tarasyuk, Troubitsyna, & Laibinis, 2010), we have extended the existing Event-B framework with two new features to enable quantitative reasoning about system dependability attributes. The first one is a new operator – *(quantitative) probabilistic choice (assignment)* – that assigns new values to variables with some precise probabilities. In other words, it behaves according to some known (discrete) probabilistic distribution. For instance, the quantitative probabilistic assignment

$$x :\oplus | x_1 @ p_1; \ldots; x_n @ p_n,$$

where $\sum_{i=1}^{n} p_i = 1$ assigns to the variable x a new value x_1 with the corresponding non-zero probability p_i. The extension is conservative, i.e. the new operator can be introduced only to replace a nondeterministic choice (assignment) statement in the event actions. Such an extension requires only minor modifications of the existing framework because all the proof obligations generated for abstract events with nondeterministic assignments are also valid for the refined probabilistic ones. It has been shown that any probabilistic choice statement always refines its demonic nondeterministic counterpart (McIver & Morgan, 2005). Hence such an extension is not interfering with traditional refinement process.

The other feature we have introduced is a new clause Operational guards containing state predicates precisely defining the subset of *operational* system states. This is a shorthand notation implicitly adding the corresponding guard condi-

tions to all events except initialisation. In general, operational states of a system, i.e., the states where the system functions properly are defined by some predicate over the system variables. Usually, essential properties of the system (such as safety, fault tolerance, liveness properties) can be guaranteed only while the system stays in the operational states. The operational guard *J(v)* partitions the system state space *S* into two disjoint classes of states – *operational* (S_{op}) and *non-operational* (S_{nop}) states, where $S_{op} \triangleq \{s \in S \mid J.s\}$ and $S_{nop} \triangleq S \setminus S_{op}$. We assume that, like model invariants, operational guards are inherited in all refined models.

The extension of Event-B for stochastic modelling necessitates strengthening of the notion of refinement as well. In (Tarasyuk, Troubitsyna, & Laibinis, 2010) we have done this for Event-B refinement by additionally requiring that the refined model would be more reliable. In engineering, reliability (Villemeur, 1996; O'Connor, 1995) is generally measured by the probability that an entity *E* can perform a required function under given conditions for the time interval *[0,t]*:

$R(t) = \mathbf{P}\{\text{E not failed over time } [0,t]\}$.

To introduce some "notion of time" to Event-B, we focus on modelling systems with cyclic behaviour, i.e. the systems that iteratively execute a predefined sequence of steps. Typical representatives of such cyclic systems are control and monitoring systems. An iteration of a control system includes reading the sensors that monitor the controlled physical processes, processing the obtained sensor values, and setting actuators according to a predefined control algorithm. In principle, the system could operate in this way indefinitely long. However, different failures may affect the normal system functioning and lead to a shutdown. Hence, during each iteration the system status should be re-evaluated to decide whether it can continue its operation. Hence reliability can be expressed as the probability that the operational guard *J* remains *TRUE* during a certain number of iterations, i.e., the probability of system staying operational for *t* iterations:

$R(t) = \boldsymbol{P}\{\Box^{\leq t} J\}$.

Here we use the modal operator □ borrowed from a temporal logic (liner temporal logic (LTL) (Pnueli, 1977) or probabilistic computational tree logic (PCTL) (Hansson & Jonsson, 1994), for instance). The formula $(\{\Box^{\leq t} J)$ means that *J* holds *globally* for the first *t* iterations. It is straightforward to see that this property corresponds to the standard definition of reliability given above.

Furthermore, in (Tarasyuk, Troubitsyna, & Laibinis, 2010) we have shown that behavioural semantics of probabilistic Event-B models is defined by Markov process – a *discrete time Markov chain* (DTMC) (Kemeny & Snell, 1960) for a fully probabilistic model and a *Markov decision process* (MDP) (White, 1993) for a probabilistic model with non-determinism.

FROM EVENT-B TO PROBABILISTIC MODEL CHECKING

Development and verification of Event-B models is supported by the Rodin Platform – an integrated extensible development environment for Event-B. However, at the moment the support for quantitative verification is sorely missing. To prove probabilistic refinement of Event-B models, we need to extend the Rodin platform with a dedicated plug-in or integrate some external tool. One of the available automated techniques widely used for analysing systems that exhibit probabilistic behaviour is probabilistic model checking (Baier & Katoen, 2008; Kwiatkowska, 2007). In particular, the probabilistic model checking frameworks like PRISM or MRMC (PRISM model checker; MRMC model checker) provide good tool support for formal modelling and verification of discrete- and continuous-time Markov processes. To enable

the quantitative reliability analysis of Event-B models, it would be advantageous to develop a Rodin plug-in enabling automatic translation of Event-B models to existing probabilistic model checking frameworks. In this chapter we choose the PRISM probabilistic symbolic model checker to verify probabilistic refinement in Event-B.

Similarly to Event-B, the PRISM modelling language is a high-level state-based language. It relies on the Reactive Modules formalism of Alur and Henzinger (Alur & Henzinger, 1999). PRISM supports the use of constants and variables that can be integers, doubles (real numbers) and Booleans. Constants are used, for instance, to define the probabilities associated with variable updates. The variables in PRISM are finite-ranged and strongly typed. They also can be local (i.e., associated with a particular module) or global ones.

An example of a system specification in PRISM is shown below. A PRISM specification is constructed as a parallel composition of modules. Modules can be independent of each other or interact with each other. Each module has a number of local variables – denoted as v_1, v_2 – and a set of guarded commands that determine its dynamic behaviour. The guarded commands can have names (labels). Similarly to events of Event-B, a guarded command can be executed if its guard evaluates to *TRUE*. If several guarded commands are enabled then the choice between them can be non-deterministic in the case of MDP or probabilistic (according to the uniform distribution) in the case of DTMC. In general, the body of a guarded command is expressed as a probabilistic choice between deterministic assignments. Figure 2 shows and example of DTMC model in PRISM.

Synchronisation between modules is defined via guarded commands with the matching names. For instance, in the PRISM specification shown above, the modules M_1 and M_2 have the guarded commands labelled with the same name *l*. Whenever both commands are enabled, the modules M_1 and M_2 synchronise by simultaneously executing the bodies of these commands.

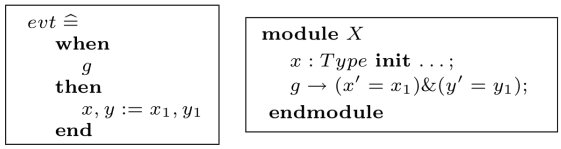
Figure 2. DTMC model in PRISM

```
dtmc
const double p11 = ...;
  ...
global v : Type init ...;
  ...
module M1
  v1 : Type init ...;
  [] g11 → p11 : act11 + ··· + p1n : act1n;
  [l] g12 → q11 : act'11 + ··· + q1m : act'1m;
  ...
endmodule
module M2
  v2 : Type init ...;
  [l] g21 → p21 : act21 + ··· + p2k : act2k;
  [] g22 → q21 : act'21 + ··· + q2l : act'2l;
  ...
endmodule
```

Because of the similarities between the languages, converting an Event-B model into its PRISM counterpart is rather straightforward. First, we need to restrict the types of Event-B variables to the types supported by PRISM. The Event-B invariants can be represented as a separate list of PRISM properties (expressed as temporal logic formulas) and then can be verified by PRISM if needed. While converting events into the PRISM guarded commands, we distinguish four different cases: the initialisation events, the events with a (parallel) deterministic assignment, the events with a single probabilistic assignment, and, finally, the events with a parallel probabilistic assignment:

- The initialisation events are converted into the corresponding initialisation part of the variable declaration;
- An event with a (parallel) deterministic assignment can be represented in PRISM as shown on Figure 3 (*&* in PRISM guarded commands denotes the parallel composition).
- An event with a probabilistic assignment can be represented as shown on Figure 4.
- An event with a parallel probabilistic assignment can be represented by synchronising the guarded commands with the

Figure 3. Conversion of deterministic assignment



identical names and guards. The bodies of these commands are formed from the actions of the corresponding events (see Figure 5).

While analysing a PRISM model, we define a number of temporal logic properties and systematically check the model to verify them. Properties of discrete-time PRISM models, i.e., DTMC and MDP, are expressed formally in the probabilistic computational tree logic. The PRISM property specification language supports a number of different types of such properties. For example, the **P** operator is used to refer to the probability of a certain event occurrence. The operator **G**, when used inside the operator **P**, allows us to express invariant properties, i.e., properties maintained by the system globally. Then, obviously, the property (1) that defines the reliability function of a system can be specified in PRISM in a following way:

$$.P_{=?}[G{\leq}tJ] \qquad (1)$$

Essentially, our approach to the development and assessment of cyclic dependable systems can be described by the following guidelines:

1. Abstractly specify the system behaviour. To achieve this:
   a. Create an abstract model of the system behaviour;
   b. Define the operational guard J that characterises operational states of the system;
   c. Strengthen the guards of events to ensure that, when J is not satisfied, the system deadlocks.
2. Refine the system to introduce the required implementation details:
   a. If a refinement step introduces the representation of unreliable components into the system specification, explicitly (non-deterministically) model both the faulty and fault-free behaviour of the system;
   b. In the model invariant, explicitly define the connection between the existing and newly introduced variables (gluing invariants). Furthermore, prove that the refined system does not introduce additional deadlocks;
   c. At each refinement step reformulate J in the terms of the newly introduced variables and functionality;
   d. If needed, refine the nondeterministic behaviour of unreliable components by probabilistic one.

Figure 4. Conversion of probabilistic assignment

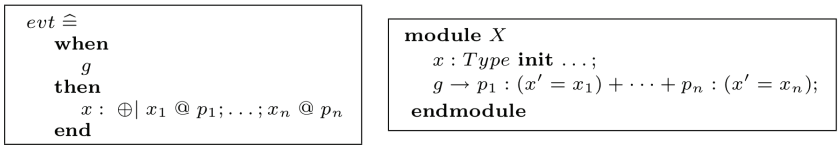
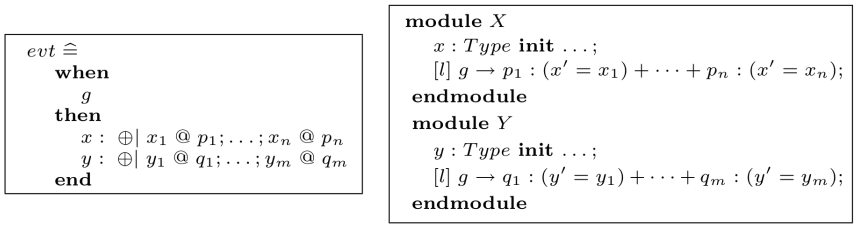


Figure 5. Conversion of parallel probabilistic assignment



3. To verify probabilistic refinement and evaluate the impact of unreliable components on the system reliability, convert the Event-B specification into its PRISM counterpart:
   a. Translate all the events of the Event-B specification into the corresponding PRISM guarded commands. Explicitly define the synchronisation points between modules;
   b. Verify the property (1) to evaluate system reliability;
   c. Compare the results of probabilistic model checking to ensure the correctness of the refinement step.
4. Continue the refinement process until the desired abstraction level is achieved.

In the next section we exemplify the proposed approach by considering a simple case study.

CASE STUDY

Our case study is a simple monitoring system. A sensor produces certain measurements that are periodically read by a controller. The controller analyses each reading to detect whether the sensor functions properly or it has failed. If no fault is detected then the system outputs the obtained sensor reading and continues to iterate in the same way. This constitutes a normal (fault-free) system state.

However, if the controller detects a sensor failure then the system enters a *degraded* state. In this state it outputs the last good sensor reading. At the same time, it keeps periodically reading the sensor outputs to detect whether it has recovered. The system can stay in the degraded state for a limited period of time (we assume it cannot exceed N iterations). If sensor recovers from its failure within the allowed time limit then the system gets back to the normal state and its normal function is resumed. Otherwise, the system aborts. A graphical representation of the system behaviour is given in Figure 6.

In the most abstract way an Event-B machine that nondeterministically models such a system can be specified as presented on Figure 7.

The variable *st* defines the state of the system. Initially the system is in the *ok* state. We have two events that abstractly model behaviour of the monitoring system. The first of them – $system_{progress}$ – models behaviour of the system in *ok* and *degraded* modes. The event $system_{failure}$ models system shutdown in case the system has failed to recover. The operational guard J implies that the system stays operational while $st \neq failed$. The specification shown on Figure 8 (the machine MCH_1) is one possible refinement of our abstract model.

Figure 6. The behaviour of a monitoring system

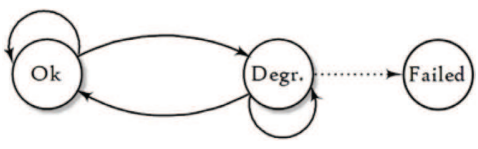


Figure 7. Case study: the abstract specification

Machine MCH_0
Variables st
Invariants $I : st \in \{ok, degraded, failed\}$
Operational guards $J : st \neq failed$
Initialisation $st := ok$
Event $system_{progress} \widehat{=}$
  when
    $st \neq failed$
  then
    $st :\in \{ok, degraded\}$
  end
Event $system_{failure} \widehat{=}$
  when
    $st = degraded$
  then
    $st := failed$
  end

In MCH_1 we introduce the variable *phase* that models the phases that the system goes through within one iteration: first reading the sensor, then detecting sensor failure, and finally outputting either the sensor reading or the last good value. The variable *s* models sensor status. When *s* equals 1, the sensor is "healthy". The value of the variable *cnt* corresponds to the number of successive iterations when the sensor remained faulty, i.e., the variable *s* has had the value 0. The system failure occurs when *cnt* exceeds the value *N*. In this case the variable *phase* gets the value *abort* and the specification deadlocks. For the sake of simplicity, we omit representation of the output produced by the system.

Let us note that we could have merged the events $sensor_{ok}$ and $sensor_{nok}$ into a single event by dropping the second conjuncts from their guards. The combined event would model sensor reading irrespectively whether the sensor has been faulty or healthy. However, we deliberately decided to model them separately here to be able to attach different probabilities when we further refine these events, thus distinguishing between the cases when the sensor fails and recovers (see Figure 9).

Figure 8. Case study: the first refinement

Machine MCH_1
Variables $st, s, phase, cnt$
Invariants $I_1 : s \in \{0,1\}$ $I_2 : cnt \in \mathbb{N}$
  $I_3 : phase \in \{in, det, out, abort\}$
  $I_4 : cnt \leq N \Rightarrow st \neq failed$
  $I_5 : phase = abort \Leftrightarrow st = failed$
Operational guards $J : st \neq failed$
Initialisation
  $st := ok \parallel s := 1 \parallel phase := in \parallel cnt := 0$
Event $sensor_{ok} \widehat{=}$
  when
    $phase = in \wedge s = 1$
  then
    $s :\in \{0,1\} \parallel phase := det$
  end
Event $sensor_{nok} \widehat{=}$
  when
    $phase = in \wedge s = 0$
  then
    $s :\in \{0,1\} \parallel phase := det$
  end
Event $fd_{ok} \widehat{=}$
  when
    $phase = det \wedge s = 1$
  then
    $cnt := 0 \parallel phase := out$
  end
Event $fd_{nok} \widehat{=}$
  when
    $phase = det \wedge s = 0$
  then
    $cnt := cnt + 1 \parallel phase := out$
  end
Event $return_{ok} \widehat{=}$
  refines $system_{progress}$
  when
    $phase = out \wedge cnt = 0$
  then
    $st := ok \parallel phase := in$
  end
Event $return_{degraded} \widehat{=}$
  refines $system_{progress}$
  when
    $phase = out \wedge cnt > 0 \wedge cnt \leq N$
  then
    $st := degraded \parallel phase := in$
  end
Event $return_{failed} \widehat{=}$
  refines $system_{failure}$
  when
    $phase = out \wedge cnt > N$
  then
    $st := failed \parallel phase := abort$
  end

Figure 9. Case study: the second refinement

Machine MCH_2
...
Event $sensor_{ok} \widehat{=}$
refines $sensor_{ok}$
when
$phase = in \wedge s = 1$
then
$s \oplus| \ 0 \ @ \ f; 1 \ @ \ 1-f$
end

Event $sensor_{nok} \widehat{=}$
refines $sensor_{nok}$
when
$phase = in \wedge s = 0$
then
$s \oplus| \ 1 \ @ \ r; 0 \ @ \ 1-r$
end
...

In this refinement, we probabilistically refine the events $sensor_{ok}$ and $sensor_{nok}$. The constants f and r are used to model correspondingly the probabilities of sensor failure and recovery. Please note that, while translating the Event-B specification MCH_2 into the corresponding PRISM specification, we convert the type of the variable *phase* to enumerated integers. The events modelling the system behaviour at each phase are grouped together into the corresponding PRISM modules. The PRISM model resulting from this rather straightforward translation is shown on Figure 10.

To evaluate reliability of the system, we have to verify the PRISM property (1) using the operational guard $J \triangleq st \neq failed$:

$$P_{=?}[G{\leq}t \ st{\neq}failed] \quad (2)$$

Figure 10. Case study: PRISM model of the second refinement

dtmc MCH_2
const double $f = \ldots;$
const double $r = \ldots;$
const int $N = \ldots;$
global $phase : [0..3]$ **init** $0;$
module $sensor$
$s : [0..1]$ **init** $1;$
$[] \ (phase = 0)\&(s = 1) \rightarrow f : (s' = 0)\&(phase' = 1) + (1 - f) : (s' = 1)\&(phase' = 1);$
$[] \ (phase = 0)\&(s = 0) \rightarrow r : (s' = 1)\&(phase' = 1) + (1 - r) : (s' = 0)\&(phase' = 1);$
endmodule
module fd
$cnt : [0..N + 1]$ **init** $0;$
$[] \ (phase = 1)\&(s = 1) \rightarrow (cnt' = 0)\&(phase' = 2);$
$[] \ (phase = 1)\&(s = 0) \rightarrow (cnt' = cnt + 1)\&(phase' = 2);$
endmodule
module $return$
$st : [0..2]$ **init** $0;$
$[] \ (phase = 2)\&(cnt = 0) \rightarrow (st' = 0)\&(phase' = 0);$
$[] \ (phase = 2)\&(cnt > 0)\&(cnt \leq N) \rightarrow (st' = 1)\&(phase' = 0);$
$[] \ (phase = 2)\&(cnt > N) \rightarrow (st' = 2)\&(phase' = 3);$
endmodule

Figure 11. Case study: the third refinement

Machine MCH_3
Variables $st, s_1, s_2, phase, cnt$
Invariants $I_6 : s_1 \in \{0,1\}$ $I_7 : s_2 \in \{0,1\}$
$I_8 : s_1 + s_2 > 0 \Leftrightarrow s = 1$
$I_9 : s_1 + s_2 = 0 \Leftrightarrow s = 0$
Operational guards $J : st \neq failed$
Initialisation
$\ldots \quad s_1, s_2 := 1, 1$
Event $sensor_{both\_ok} \widehat{=}$
refines $sensor_{ok}$
when
$phase = in \wedge s_1 = 1 \wedge s_2 = 1$
then
$s_1 \oplus\!| \ 0 \ @ \ f; 1 \ @ \ 1-f$
$s_2 \oplus\!| \ 0 \ @ \ f; 1 \ @ \ 1-f$
$phase := det$
end
Event $sensor_{ok\_1} \widehat{=}$
refines $sensor_{ok}$
when
$phase = in \wedge s_1 = 1 \wedge s_2 = 0$
then
$s_1 \oplus\!| \ 0 \ @ \ f; 1 \ @ \ 1-f$
$s_2 \oplus\!| \ 1 \ @ \ r; 0 \ @ \ 1-r$
$phase := det$
end
Event $sensor_{ok\_2} \widehat{=}$
refines $sensor_{ok}$
when
$phase = in \wedge s_1 = 0 \wedge s_2 = 1$
then
$s_1 \oplus\!| \ 1 \ @ \ r; 0 \ @ \ 1-r$
$s_2 \oplus\!| \ 0 \ @ \ f; 1 \ @ \ 1-f$
$phase := det$
end
Event $sensor_{nok} \widehat{=}$
refines $sensor_{nok}$
when
$phase = in \wedge s_1 = 0 \wedge s_2 = 0$
then
$s_1 \oplus\!| \ 1 \ @ \ r; 0 \ @ \ 1-r$
$s_2 \oplus\!| \ 1 \ @ \ r; 0 \ @ \ 1-r$
$phase := det$
end
Event $fd_{ok} \widehat{=}$
refines fd_{ok}
when
$phase = det \wedge s_1 + s_2 > 0$
then
$cnt := 0 \parallel phase := out$
end
Event $fd_{nok} \widehat{=}$
refines fd_{nok}
when
$phase = det \wedge s_1 + s_2 = 0$
then
$cnt := cnt + 1 \parallel phase := out$
end
$\ldots$

The resulting reliability evaluation is presented in Figure 13 together with the reliability of a monitoring system that, instead of a single sensor, employs two sensors arranged in a hot spare, as we explain next.

A hot spare is a standard fault tolerance mechanism. In our model, we introduce an additional sensor – a spare – that works in parallel with the main one. When a fault is detected, the system switches to reading of the data produced by a spare.

An introduction of the fault tolerance mechanisms by refinement is a rather standard refinement step often performed in the development of dependable systems. The machine MCH_3 is a result of refining the machine MCH_2 to introduce a hot spare. In the refined specification, we replace the sensor s by two sensors s_1 and s_2. The behaviour of these sensors is the same as the behaviour of s. The gluing invariant I_8 describes the refinement relationship between the corresponding variables – the system would output the actual sensor readings only if no more than one sensor have failed. The third refinement step is shown on Figure 11.

In the last refinement step we split the abstract event $sensor_{ok}$ into three events $sensor_{both\_ok}$, $sensor_{ok\_1}$ and $sensor_{ok\_2}$, and also refined the abstract event $sensor_{nok}$. Moreover, in the next two *detection* events the abstract guards s=1 and s=2 have been replaced by guards s_1+s_2>0 and s_1+s_2=0 respectively. It is easily provable that this is a valid Event-B refinement step.

To prove that MCH_3 is a probabilistic refinement of MCH_2, we start with conversion of the Event-B machine MCH_3 to its PRISM counterpart. The corresponding module represents the behaviour of each sensor. The synchronised guarded commands (labelled *ss*) are used to model parallel work of sensors. In the module $sensor_2$ we addi-

Figure 12. Case study: PRISM model of the third refinement

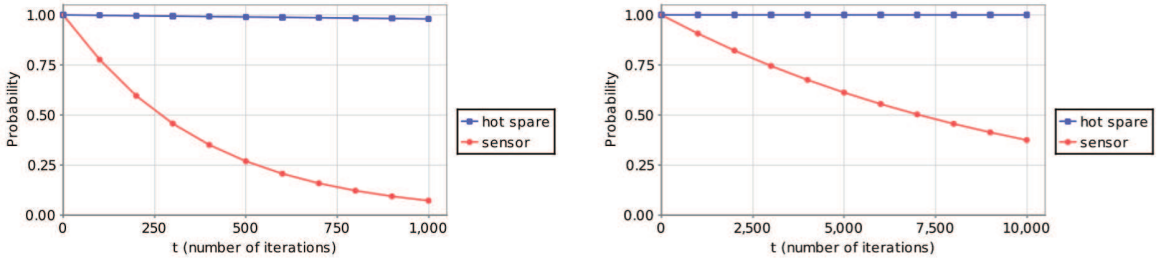
```
dtmc MCH_3
const double f = ...;
const double r = ...;
const int N = ...;
global phase : [0..3] init 0;
module sensor_1
  s_1 : [0..1] init 1;
  [ss] (phase = 0)&(s_1 = 1) → f : (s_1' = 0) + (1 − f) : (s_1' = 1);
  [ss] (phase = 0)&(s_1 = 0) → r : (s_1' = 1) + (1 − r) : (s_1' = 0);
endmodule
module sensor_2
  s_2 : [0..1] init 1;
  sw : bool init true;
  [ss] (phase = 0)&(s_2 = 1)&(sw) →
                    f : (s_2' = 0)&(sw' = false) + (1 − f) : (s_2' = 1)&(sw' = false);

  [ss] (phase = 0)&(s_2 = 0)&(sw) →
                    r : (s_2' = 1)&(sw' = false) + (1 − r) : (s_2' = 0)&(sw' = false);

  [] (phase = 0)&(!sw) → (sw' = true)&(phase' = 1);
endmodule
module fd
  cnt : [0..N + 1] init 0;
  [] (phase = 1)&(s_1 + s_2 > 0) → (cnt' = 0)&(phase' = 2);
  [] (phase = 1)&(s_1 + s_2 = 0) → (cnt' = cnt + 1)&(phase' = 2);
endmodule
...
```

tionally update the global variable *phase* to model transition of the system to the detection phase. An excerpt from the resulting PRISM system is given in Figure 12.

Further we need to compare the results obtained from model checking to show that (from reliability viewpoint) MCH_3 is indeed a valid probabilistic refinement of MCH_2. In the refined specification the system operational states are still confined by the predicate $st \neq failed$. Hence, to assess reliability, we again check the property (2). Figure 13 shows the results of evaluation for the more abstract and refined systems. The results clearly show that the redundant hot spare system always gives a significantly better reliability. Due to the lack of space, we omit comparison between various fault tolerance mechanisms that could be used in our system. Such a comparison for a similar system can be found in (Tarasyuk, Troubitsyna, & Laibinis, 2009).

While illustrating our approach for reliability assessment, we have deliberately chosen a simple example. Since the Markov models for these examples are relatively simple, we can also obtain the analytical representation of the reliability functions instead of use of model checking. Obviously, for large systems, the corresponding Markov models would be much more complex. Be-

Figure 13. Case study: results of PRISM modelling (left: f=0.1, r=0.5, N=5; right: f=0.2, r=0.6, N=8)



sides, our reliability assessment relies on the time bound reachability analysis, which is known to be a difficult problem per se. Hence, in general, the analytical assessment of reliability could be unfeasible, making us to rely on the results of probabilistic model checking to prove Event-B refinement from reliability the point of view.

CONCLUSION

In this chapter we have proposed an approach that allows us to incorporate reliability assessment into the refinement process. The proposed approach enables reliability assessment at early design phases, which permits the designers to evaluate reliability of different design alternatives already at the development phase.

Our approach integrates two frameworks: refinement in Event-B and probabilistic model checking. By integrating probabilistic verification supported by the PRISM model checker, we open a possibility to reason about non-functional system requirements in the refinement process. Such integration not only guarantees functional correctness but also ensures that reliability of refined model is preserved or improved.

Several researches have already used quantitative model checking for dependability evaluation. For instance, Kwiatkowska et al. (Kwiatkowska, Norman, & Parker, 2007) have proposed an approach to assessing dependability of control systems using continuous time Markov chains. The general idea is similar to ours – to formulate reliability as a system property to be verified. This approach differs from ours because it aims at assessing reliability of already developed systems. However, dependability evaluation late at the development cycle can be perilous and, in case of poor results, may lead to major system redevelopment causing significant financial and time losses. In our approach reliability assessment proceeds hand-in-hand with the system development by refinement. It allows us to assess dependability of designed system on the early stages of development, for instance, every time when we need to estimate impact of unreliable component on the system reliability level. This allows a developer to make an informed decision about how to guarantee desired system reliability.

ACKNOWLEDGMENT

This work is supported by IST FP7 DEPLOY project. We also wish to thank the anonymous reviewers for their helpful comments.

REFERENCES

Abrial, J.-R. (1996). *Extending B without changing it* (for developing distributed systems). 1st Conference on the B method, (pp. 169-190).

Abrial, J.-R. (1996). *The B-Book: Assigning programs to meanings*. Cambridge, UK: Cambridge University Press. doi:10.1017/CBO9780511624162

Abrial, J.-R. (2010). *Modeling in Event-B*. Cambridge, UK: Cambridge University Press.

Alur, R., & Henzinger, T. (1999). Reactive modules. *Formal Methods in System Design*, 7–48. doi:10.1023/A:1008739929481

Baier, C., & Katoen, J.-P. (2008). *Principles of model checking*. The MIT Press.

Craigen, D., Gerhart, S., & Ralson, T. (1994). Case study: Paris metro signaling system. *IEEE Software*, 32–25.

DEPLOY. IST FP7 project. (2008). *European Commission Information and Communication Technologies FP7 DEPLOY project*. Retrieved from http://www.deploy-project.eu/

Hansson, H., & Jonsson, B. (1994). A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 512-535.

Kemeny, J. G., & Snell, J. L. (1960). *Finite Markov chains*. Van Nostrand.

Kwiatkowska, M. (2007). *Quantitative verification: Models techniques and tools*. ESEC/FSE 2007, European Software Engineering Conference and Symposium on the Foundations of Software Engineering, (pp. 449-458).

Kwiatkowska, M., Norman, G., & Parker, D. (2007). Controller dependability analysis by probabilistic model checking. *Control Engineering Practice*, 1427–1434. doi:10.1016/j.conengprac.2006.07.003

McIver, A., & Morgan, C. C. (2005). *Abstraction, refinement and proof for probabilistic systems*. Springer. MRMC model checker, ver.1.4.1. (2010). *MRMC – Markov reward model Checker*. Retrieved 2010, from http://www.mrmc-tool.org/

O'Connor, P. (1995). *Practical reliability engineering* (3rd ed.). John Wiley & Sons.

Platform, R. O. D. I. N. ver. 2.0. (2010). *RODIN Event-B platform*. Retrieved 2010 from http://www.event-b.org/

Pnueli, A. (1977). *The temporal logic of programs. FOCS 1977* (pp. 46–57). Foundations of Computer Science.

PRISM model checker, ver. 3.3.1. (2010). *PRISM – Probabilistic symbolic model checker*. Retrieved 2010, from http://www.prismmodelchecker.org/

RODIN. IST FP6 project. (2004). *Rigorous open development environment for complex systems*. Retrieved 2010 from http://rodin.cs.ncl.ac.uk/

Tarasyuk, A., Troubitsyna, E., & Laibinis, L. (2009). *Reliability assessment in Event-B,* (TUCS Technical Report N932). Turku Centre for Computer Science.

Tarasyuk, A., Troubitsyna, E., & Laibinis, L. (2010). *Towards probabilistic modelling in Event-B. IFM 2010, Integrated Formal Methods* (pp. 275–289). Springer-Verlag.

Villemeur, A. (1996). *Reliability, availability, maintainability and safety assessment*. John Wiley & Sons.

White, D. J. (1993). *Markov decision processes*. John Wiley & Sons.

KEY TERMS AND DEFINITIONS

B Method: A rigorous, state-based formal framework supporting the correct-by-construction system development.

Program Refinement: (Stepwise) verifiable model transformation process of an abstract formal specification into a concrete specification that is close to the desired implementation.

Event-B: A formal framework derived from the B Method by adopting the event-based modeling style that facilitates development of reactive and distributed systems.

Reliability: The ability of a system to perform a required function under given conditions for a given time interval.

Discrete Time Markov Chain: A discrete-time stochastic process with the property that a future state of the process only depends on the current process state and not on its past history (Markov property).

Markov Decision Process: A discrete-time stochastic control process characterized by a set of states, actions, and transition probability matrices that depend on the actions chosen within a given state.

Probabilistic Model Checking: A formal technique for analysing and verifying the correctness of finite-state systems that exhibit stochastic behaviour.

Turku Centre for Computer Science
TUCS Dissertations

1. **Marjo Lipponen**, On Primitive Solutions of the Post Correspondence Problem
2. **Timo Käkölä**, Dual Information Systems in Hyperknowledge Organizations
3. **Ville Leppänen**, Studies on the Realization of PRAM
4. **Cunsheng Ding**, Cryptographic Counter Generators
5. **Sami Viitanen**, Some New Global Optimization Algorithms
6. **Tapio Salakoski**, Representative Classification of Protein Structures
7. **Thomas Långbacka**, An Interactive Environment Supporting the Development of Formally Correct Programs
8. **Thomas Finne**, A Decision Support System for Improving Information Security
9. **Valeria Mihalache**, Cooperation, Communication, Control. Investigations on Grammar Systems.
10. **Marina Waldén**, Formal Reasoning About Distributed Algorithms
11. **Tero Laihonen**, Estimates on the Covering Radius When the Dual Distance is Known
12. **Lucian Ilie**, Decision Problems on Orders of Words
13. **Jukkapekka Hekanaho**, An Evolutionary Approach to Concept Learning
14. **Jouni Järvinen**, Knowledge Representation and Rough Sets
15. **Tomi Pasanen**, In-Place Algorithms for Sorting Problems
16. **Mika Johnsson**, Operational and Tactical Level Optimization in Printed Circuit Board Assembly
17. **Mats Aspnäs**, Multiprocessor Architecture and Programming: The Hathi-2 System
18. **Anna Mikhajlova**, Ensuring Correctness of Object and Component Systems
19. **Vesa Torvinen**, Construction and Evaluation of the Labour Game Method
20. **Jorma Boberg**, Cluster Analysis. A Mathematical Approach with Applications to Protein Structures
21. **Leonid Mikhajlov**, Software Reuse Mechanisms and Techniques: Safety Versus Flexibility
22. **Timo Kaukoranta**, Iterative and Hierarchical Methods for Codebook Generation in Vector Quantization
23. **Gábor Magyar**, On Solution Approaches for Some Industrially Motivated Combinatorial Optimization Problems
24. **Linas Laibinis**, Mechanised Formal Reasoning About Modular Programs
25. **Shuhua Liu**, Improving Executive Support in Strategic Scanning with Software Agent Systems
26. **Jaakko Järvi**, New Techniques in Generic Programming – C++ is more Intentional than Intended
27. **Jan-Christian Lehtinen**, Reproducing Kernel Splines in the Analysis of Medical Data
28. **Martin Büchi**, Safe Language Mechanisms for Modularization and Concurrency
29. **Elena Troubitsyna**, Stepwise Development of Dependable Systems
30. **Janne Näppi**, Computer-Assisted Diagnosis of Breast Calcifications
31. **Jianming Liang**, Dynamic Chest Images Analysis
32. **Tiberiu Seceleanu**, Systematic Design of Synchronous Digital Circuits
33. **Tero Aittokallio**, Characterization and Modelling of the Cardiorespiratory System in Sleep-Disordered Breathing
34. **Ivan Porres**, Modeling and Analyzing Software Behavior in UML
35. **Mauno Rönkkö**, Stepwise Development of Hybrid Systems
36. **Jouni Smed**, Production Planning in Printed Circuit Board Assembly
37. **Vesa Halava**, The Post Correspondence Problem for Market Morphisms
38. **Ion Petre**, Commutation Problems on Sets of Words and Formal Power Series
39. **Vladimir Kvassov**, Information Technology and the Productivity of Managerial Work
40. **Frank Tétard**, Managers, Fragmentation of Working Time, and Information Systems

41. **Jan Manuch**, Defect Theorems and Infinite Words
42. **Kalle Ranto**, Z_4-Goethals Codes, Decoding and Designs
43. **Arto Lepistö**, On Relations Between Local and Global Periodicity
44. **Mika Hirvensalo**, Studies on Boolean Functions Related to Quantum Computing
45. **Pentti Virtanen**, Measuring and Improving Component-Based Software Development
46. **Adekunle Okunoye**, Knowledge Management and Global Diversity – A Framework to Support Organisations in Developing Countries
47. **Antonina Kloptchenko**, Text Mining Based on the Prototype Matching Method
48. **Juha Kivijärvi**, Optimization Methods for Clustering
49. **Rimvydas Rukšėnas**, Formal Development of Concurrent Components
50. **Dirk Nowotka**, Periodicity and Unbordered Factors of Words
51. **Attila Gyenesei**, Discovering Frequent Fuzzy Patterns in Relations of Quantitative Attributes
52. **Petteri Kaitovaara**, Packaging of IT Services – Conceptual and Empirical Studies
53. **Petri Rosendahl**, Niho Type Cross-Correlation Functions and Related Equations
54. **Péter Majlender**, A Normative Approach to Possibility Theory and Soft Decision Support
55. **Seppo Virtanen**, A Framework for Rapid Design and Evaluation of Protocol Processors
56. **Tomas Eklund**, The Self-Organizing Map in Financial Benchmarking
57. **Mikael Collan**, Giga-Investments: Modelling the Valuation of Very Large Industrial Real Investments
58. **Dag Björklund**, A Kernel Language for Unified Code Synthesis
59. **Shengnan Han**, Understanding User Adoption of Mobile Technology: Focusing on Physicians in Finland
60. **Irina Georgescu**, Rational Choice and Revealed Preference: A Fuzzy Approach
61. **Ping Yan**, Limit Cycles for Generalized Liénard-Type and Lotka-Volterra Systems
62. **Joonas Lehtinen**, Coding of Wavelet-Transformed Images
63. **Tommi Meskanen,** On the NTRU Cryptosystem
64. **Saeed Salehi**, Varieties of Tree Languages
65. **Jukka Arvo**, Efficient Algorithms for Hardware-Accelerated Shadow Computation
66. **Mika Hirvikorpi**, On the Tactical Level Production Planning in Flexible Manufacturing Systems
67. **Adrian Costea**, Computational Intelligence Methods for Quantitative Data Mining
68. **Cristina Seceleanu**, A Methodology for Constructing Correct Reactive Systems
69. **Luigia Petre**, Modeling with Action Systems
70. **Lu Yan**, Systematic Design of Ubiquitous Systems
71. **Mehran Gomari**, On the Generalization Ability of Bayesian Neural Networks
72. **Ville Harkke**, Knowledge Freedom for Medical Professionals – An Evaluation Study of a Mobile Information System for Physicians in Finland
73. **Marius Cosmin Codrea**, Pattern Analysis of Chlorophyll Fluorescence Signals
74. **Aiying Rong**, Cogeneration Planning Under the Deregulated Power Market and Emissions Trading Scheme
75. **Chihab BenMoussa**, Supporting the Sales Force through Mobile Information and Communication Technologies: Focusing on the Pharmaceutical Sales Force
76. **Jussi Salmi**, Improving Data Analysis in Proteomics
77. **Orieta Celiku**, Mechanized Reasoning for Dually-Nondeterministic and Probabilistic Programs
78. **Kaj-Mikael Björk**, Supply Chain Efficiency with Some Forest Industry Improvements
79. **Viorel Preoteasa**, Program Variables – The Core of Mechanical Reasoning about Imperative Programs
80. **Jonne Poikonen**, Absolute Value Extraction and Order Statistic Filtering for a Mixed-Mode Array Image Processor
81. **Luka Milovanov**, Agile Software Development in an Academic Environment
82. **Francisco Augusto Alcaraz Garcia**, Real Options, Default Risk and Soft Applications
83. **Kai K. Kimppa**, Problems with the Justification of Intellectual Property Rights in Relation to Software and Other Digitally Distributable Media
84. **Dragoş Truşcan**, Model Driven Development of Programmable Architectures
85. **Eugen Czeizler**, The Inverse Neighborhood Problem and Applications of Welch Sets in Automata Theory

86. **Sanna Ranto**, Identifying and Locating-Dominating Codes in Binary Hamming Spaces
87. **Tuomas Hakkarainen**, On the Computation of the Class Numbers of Real Abelian Fields
88. **Elena Czeizler**, Intricacies of Word Equations
89. **Marcus Alanen**, A Metamodeling Framework for Software Engineering
90. **Filip Ginter**, Towards Information Extraction in the Biomedical Domain: Methods and Resources
91. **Jarkko Paavola**, Signature Ensembles and Receiver Structures for Oversaturated Synchronous DS-CDMA Systems
92. **Arho Virkki**, The Human Respiratory System: Modelling, Analysis and Control
93. **Olli Luoma**, Efficient Methods for Storing and Querying XML Data with Relational Databases
94. **Dubravka Ilić**, Formal Reasoning about Dependability in Model-Driven Development
95. **Kim Solin**, Abstract Algebra of Program Refinement
96. **Tomi Westerlund**, Time Aware Modelling and Analysis of Systems-on-Chip
97. **Kalle Saari**, On the Frequency and Periodicity of Infinite Words
98. **Tomi Kärki**, Similarity Relations on Words: Relational Codes and Periods
99. **Markus M. Mäkelä**, Essays on Software Product Development: A Strategic Management Viewpoint
100. **Roope Vehkalahti**, Class Field Theoretic Methods in the Design of Lattice Signal Constellations
101. **Anne-Maria Ernvall-Hytönen**, On Short Exponential Sums Involving Fourier Coefficients of Holomorphic Cusp Forms
102. **Chang Li**, Parallelism and Complexity in Gene Assembly
103. **Tapio Pahikkala**, New Kernel Functions and Learning Methods for Text and Data Mining
104. **Denis Shestakov**, Search Interfaces on the Web: Querying and Characterizing
105. **Sampo Pyysalo**, A Dependency Parsing Approach to Biomedical Text Mining
106. **Anna Sell**, Mobile Digital Calendars in Knowledge Work
107. **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
108. **Tero Säntti**, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
109. **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
110. **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
111. **Camilla J. Hollanti**, Order-Theoretic Mehtods for Space-Time Coding: Symmetric and Asymmetric Designs
112. **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
113. **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
114. **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
115. **Petri Salmela**, On Commutation and Conjugacy of Rational Languages and the Fixed Point Method
116. **Siamak Taati**, Conservation Laws in Cellular Automata
117. **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
118. **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
119. **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
120. **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
121. **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
122. **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip
123. **Eeva Suvitie**, On Inner Products Involving Holomorphic Cusp Forms and Maass Forms
124. **Linda Mannila**, Teaching Mathematics and Programming – New Approaches with Empirical Evaluation
125. **Hanna Suominen**, Machine Learning and Clinical Text: Supporting Health Information Flow
126. **Tuomo Saarni**, Segmental Durations of Speech
127. **Johannes Eriksson**, Tool-Supported Invariant-Based Programming

128. **Tero Jokela**, Design and Analysis of Forward Error Control Coding and Signaling for Guaranteeing QoS in Wireless Broadcast Systems
129. **Ville Lukkarila**, On Undecidable Dynamical Properties of Reversible One-Dimensional Cellular Automata
130. **Qaisar Ahmad Malik**, Combining Model-Based Testing and Stepwise Formal Development
131. **Mikko-Jussi Laakso**, Promoting Programming Learning: Engagement, Automatic Assessment with Immediate Feedback in Visualizations
132. **Riikka Vuokko**, A Practice Perspective on Organizational Implementation of Information Technology
133. **Jeanette Heidenberg**, Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches
134. **Yong Liu**, Solving the Puzzle of Mobile Learning Adoption
135. **Stina Ojala**, Towards an Integrative Information Society: Studies on Individuality in Speech and Sign
136. **Matteo Brunelli**, Some Advances in Mathematical Models for Preference Relations
137. **Ville Junnila**, On Identifying and Locating-Dominating Codes
138. **Andrzej Mizera**, Methods for Construction and Analysis of Computational Models in Systems Biology. Applications to the Modelling of the Heat Shock Response and the Self-Assembly of Intermediate Filaments.
139. **Csaba Ráduly-Baka**, Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments
140. **Jari Kyngäs**, Solving Challenging Real-World Scheduling Problems
141. **Arho Suominen**, Notes on Emerging Technologies
142. **József Mezei**, A Quantitative View on Fuzzy Numbers
143. **Marta Olszewska**, On the Impact of Rigorous Approaches on the Quality of Development
144. **Antti Airola**, Kernel-Based Ranking: Methods for Learning and Performace Estimation
145. **Aleksi Saarela**, Word Equations and Related Topics: Independence, Decidability and Characterizations
146. **Lasse Bergroth**, Kahden merkkijonon pisimmän yhteisen alijonon ongelma ja sen ratkaiseminen
147. **Thomas Canhao Xu**, Hardware/Software Co-Design for Multicore Architectures
148. **Tuomas Mäkilä**, Software Development Process Modeling – Developers Perspective to Contemporary Modeling Techniques
149. **Shahrokh Nikou**, Opening the Black-Box of IT Artifacts: Looking into Mobile Service Characteristics and Individual Perception
150. **Alessandro Buoni**, Fraud Detection in the Banking Sector: A Multi-Agent Approach
151. **Mats Neovius**, Trustworthy Context Dependency in Ubiquitous Systems
152. **Fredrik Degerlund**, Scheduling of Guarded Command Based Models
153. **Amir-Mohammad Rahmani-Sane**, Exploration and Design of Power-Efficient Networked Many-Core Systems
154. **Ville Rantala**, On Dynamic Monitoring Methods for Networks-on-Chip
155. **Mikko Pelto**, On Identifying and Locating-Dominating Codes in the Infinite King Grid
156. **Anton Tarasyuk**, Formal Development and Quantitative Verification of Dependable Systems

Turku Centre *for* Computer Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www. tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics and Statistics

Turku School of Economics
- Institute of Information Systems Science

Åbo Akademi University
Division for Natural Sciences and Technology
- Department of Information Technologies

ISBN 978-952-12-2832-2
ISSN 1239-1883

Anton Tarasyuk

Formal Development and Quantitative Verification of Dependable Systems