SCHOOL OF
COMPUTING

Title: Proceedings of the Event-B Day 2018 in Tokyo

Names: Fuyuki Ishikawa, Tsutomu Kobayashi, Alexander Romanovsky

TECHNICAL REPORT SERIES

No. CS-TR- 1525 December 2018

TECHNICAL REPORT SERIES

No. CS-TR- 1525 Date December 2018

Title: Proceedings of the Event-B Day 2018 in Tokyo

Authors: Fuyuki Ishikawa, Tsutomu Kobayashi, Alexander Romanovsky

Abstract: The Event-B Day 2018 in Tokyo was organised on November 10, 2018 in National Institute of Informatics, Tokyo, Japan. The event was supported by NII, Japan and Newcastle University, UK.
Event-B is a formal method for the system level modelling and analysis of dependable applications. It is supported by an open and extendable Eclipse based toolset called Rodin (http://www.event-b.org/), which has been developed in a series of European projects (Rodin, Deploy, Advance).
This one day event aimed to bring the community of Event-B/Rodin users and developers together to discuss new and emerging issues in applying and advancing both the Event-B method and the Rodin platform as well as to address challenges that industrial takers are facing while deploying them.

Printed and published by Newcastle University,
School of Computing, Urban Sciences Building,
Newcastle upon Tyne, NE4 5TG, England.

Bibliographical

Title and Authors: Proceedings of the Event-B Day 2018 in Tokyo
Fuyuki Ishikawa, Tsutomu Kobayashi, Alexander Romanovsky

NEWCASTLE UNIVERSITY
School of Computing. Technical Report Series. CS-TR- 1525

Abstract: The Event-B Day 2018 in Tokyo was organised on November 10, 2018 in National Institute of Informatics, Tokyo, Japan. The event was supported by NII, Japan and Newcastle University, UK.
Event-B is a formal method for the system level modelling and analysis of dependable applications. It is supported by an open and extendable Eclipsebased toolset called Rodin (http://www.event-b.org/), which has been developed in a series of European projects (Rodin, Deploy, Advance).
This one day event aimed to bring the community of Event-B/Rodin users and developers together to discuss new and emerging issues in applying and advancing both the Event-B method and the Rodin platform as well as to address challenges that industrial takers are facing while deploying them.

About the authors :

Fuyuki Ishikawa - Fuyuki Ishikawa is an Associate Professor at the National Institute of Informatics (NII) in Tokyo, Japan.
He received the PhD degree in information science and technology from the University of Tokyo in Japan, in 2007. He is a visiting associate professor at the University of Electro-Communications in Japan. His research interests include service-oriented computing and software engineering. He has served as the leader of 6 funded projects and published more than 100 papers.

Tsutomu Kobayashi - Dr Tsutomu Kobayashi is a researcher at National Institute of Informatics (NII), Japan. He received his Ph.D. degree in Information Science and Technology from the University of Tokyo, Japan, in 2017. His research interests include formal methods, software engineering, and requirements engineering. Dr Kobayashi teaches formal methods and software engineering at NII, Japan Advanced Institute of Science and Technology (JAIST), and Waseda University. He is a member of

Formal Methods Europe and Information Processing Society of Japan.

Alexander Romanovsky - Alexander (Sascha) Romanovsky is a Professor in the Centre for Software and Reliability, Newcastle University. He is the leader of the Dependability Group at the School of Computing Science. His main research interests are system dependability, fault tolerance, software architectures, exception handling, error recovery, system structuring and verification of fault tolerance.He received a M.Sc. degree in Applied Mathematics from Moscow State University and a PhD degree in Computer Science from St. Petersburg State Technical University. He was with this University from 1984 until 1996, doing research and teaching. In 1991 he worked as a visiting researcher at ABB Ltd Computer Architecture Lab Research Center, Switzerland. In 1993 he was a visiting fellow at Istituto di Elaborazione della Informazione, CNR, Pisa, Italy. In 1993-94 he was a post-doctoral fellow with the Department of Computing Science, University of Newcastle upon Tyne. Alexander is now the Principle Investigator of the TrAmS-2 EPSRC/UK platform grant on Trustworthy Ambient Systems (2012-16) and of the EPSRC/RSSB research project SafeCap on Overcoming the Railway Capacity Challenges without Undermining Rail Network Safety (2011-14), and the Co-investigator of the EPSRC PRiME program grant (2013-18) and of the FP7 COMPASS Integrated Project (2011-14).

Suggested keywords: Formal Verification_Stepwise Development_Event-B

Proceedings of the Event-B Day 2018 in Tokyo

Fuyuki Ishikawa, Tsutomu Kobayashi, Alexander Romanovsky

2018

Proceedings of the Event-B Day 2018 in Tokyo

Fuyuki Ishikawa, Tsutomu Kobayashi, Alexander Romanovsky

The Event-B Day 2018 in Tokyo was organised on November 10, 2018 in National Institute of Informatics, Tokyo, Japan. The event was supported by NII, Japan and Newcastle University, UK.

Event-B is a formal method for the system level modelling and analysis of dependable applications. It is supported by an open and extendable Eclipse-based toolset called Rodin (http://www.event-b.org/), which has been developed in a series of European projects (Rodin, Deploy, Advance).

This one day event aimed to bring the community of Event-B/Rodin users and developers together to discuss new and emerging issues in applying and advancing both the Event-B method and the Rodin platform as well as to address challenges that industrial takers are facing while deploying them.

The organizers of the event were

- Fuyuki Ishikawa (National Institute of Informatics, Japan)
- Tsutomu Kobayashi (National Institute of Informatics, Japan)
- Alexander Romanovsky (Newcastle University, UK)
- Thierry Lecomte (ClearSy, France)

The event was attended by 25 participants from Europe, Japan and Canada, including 6 from industry and 19 from academia. The programme included twelve presentations – the slides of several presentations could be downloaded from the event web site.

This is the list of the presentations given at the event:

Ilya Shchepetkov (Ivannikov Institute for System Programming of the Russian Academy of Sciences, Russia)
Using Refinement in Formal Development of OS Security Policy Model

Shin-ya Katsumata (NII, Japan)
Towards Categorical Event-B

Daichi Morita (The University of Tokyo, Japan)
Approximate Simulation as Refinement in Hybrid Event-B

Colin Snook (University of Southampton, UK)
Behaviour driven formal model development

Paulius Stankaitis (Newcastle University, UK)
Modelling and Verification of Distributed Resource Allocation Protocol

Laurent Voisin (Systerel, France)
Update on the Rodin group

Yamine Ait Ameur (IRIT/INPT-ENSEEIHT, France)
Design of distributed systems from choreographies specifications. A refinement based approach

Alexei Iliasov (Newcastle University, UK)
Safecap Verification Framework

Elena Troubitsyna (KTH, Sweden)
Developing autonomous resilient systems

Philipp Körner (University of Dusseldorf, Germany)
Embedding Formal Specifications as Libraries into Applications

Shinnosuke Saruwatari (The University of Tokyo, Japan)
Change Impact Analysis for Refinement-based Formal Specification

Laurent Voisin (Systerel, France)
On lexicographic variants in Event-B

This has been a successful event and we are grateful to all the attendees for their participation and contributions. Our special thanks to the presenters.

More information about the event could be found in on event web site:
http://research.nii.ac.jp/eventb2018/

This rest of the report includes brief summaries of the presentations given at the event:

Overview of Enable-S3 project work involving Event-B

Dr C. F. Snook

University of Southampton, Southampton, United Kingdom
cfs@soton.ac.uk

1 Outline of Enable-S3 Project

Enable-S3 is a EU funded Horizon 2020 project. The overall objective of the project is to"establish cost-efficient cross-domain virtual and semi-virtual V&V platforms and methods for *Autonomous Cyber-Physical Systems* (ACPS) ". A major part of the project is focused on improving simulation for testing purposes. This involves modelling in order to simulate a good coverage of scenarios in an efficient manner. *University of Southampton* (SOTON) differs in that it aims to use Event-B based modelling to provide correct by construction systems. The models are primarily used to detect problems early in the requirements by theorem proving of abstract concepts rather than by testing artefacts using scenario simulation. However, the Event-B models may also be useful in a scenario testing context as outlined later. in this sense, the Event-B models contribute a complimentary alternative approach to V&V of ACPS as well as integrating with the main philosophy of the Enable-S3 project.

2 Event-B in the Enable-S3 Project

The Enable-S3 project is industry driven and involves many industrial domains. SOTON is participating in three of these domains: Maritime, Avionics and Rail.

2.1 Maritime

In the maritime domain, a company called NAVTOR are developing a shore-based bridge for autonomous control of merchant shipping. SOTON is contributing to this use case by analysing the security of the communications link using a combination of *Systems Theorietic Process Analysis* (STPA) and Event-B.

2.2 Avionics

In the avionics domain, a company called AIRBUS are developing a system for controlling aircraft servicing equipment by the serviced aircraft. SOTON is contributing to this use case by analysing the security of the *Touch And Go Authentication* (TAGA) authentication process using a combination of iUML-B and Event-B

2.3 Rail

In the railway domain, a company called Thales are developing railway control products. SOTON is contributing to this use case by analysing the safety of the railway control products using a combination of iUML-B and Event-B. To support this use case, SOTON is also making improvements to the Event-B methodology and tools. Improvements to the modelling tools are outlined in section 3. One such improvement to the Event-B methodology is described in section 4.

3 Improvements to Event-B modelling tools

3.1 Improvements to iUML-B

Several features have been added to the iUML-B Class diagram tools. These include:

- subtype groups - groups of subtypes csan be specified to be disjoint or partitions,
- initialisation - improved features for initialising attributes and associations,
- enumerations - enumerated sets can be specified to be the instances of classes,
- placement of constraints - constraints can be specified to be axioms in a context, even if the diagram is in a machine.

3.2 Text editor

A text-based serialisation of Event-B has been developed and a text-based editor is provided. This allows true text-based model comparison e.g. in a repository.

3.3 Component-based model structuring (inclusion)

A model structuring mechanism has been developed. This allows Event-B models to be structured in a hierarchical component fashion by including re-usable component machines in parent machines. Multiple instances of a machine may be included.

4 Behaviour Driven Formal Model Development

Formal systems modelling offers a rigorous system-level analysis resulting in a precise and reliable specification. However, some issues remain: Modellers need to understand the requirements in order to formulate the models, formal verification may focus on safety properties rather than temporal behaviour, domain experts need to validate the final models to ensure they fit the needs of stakeholders. To address these issues we have introduced a method that that applies the principles of Behaviour-Driven Development (BDD) to formal systems modelling and validation. We propose a process (Figure 1) where manually authored

scenarios are used initially to support the requirements and help the modeller. The same scenarios are used to verify behavioural properties of the model. The model is then mutated to automatically generate scenarios that have a more complete coverage than the manual ones. These automatically generated scenarios are used to animate the model in a final acceptance stage. For this acceptance stage, it is important that a domain expert decides whether or not the behaviour is useful.

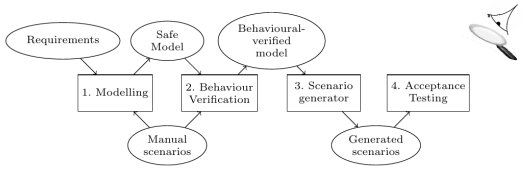

Fig. 1. A behaviour driven process for formal model development

Developing distributed resource allocation protocol with Event-B

Paulius Stankaitis[1], Alexei Iliasov[1], Alexander Romanovsky[1], Yamine Aït-Ameur[2], Tsutomu Kobayashi[3], Fuyuki Ishikawa[3]

[1] Newcastle University, Newcastle upon Tyne, United Kingdom
[2] INPT-ENSEEIHT, 2 Rue Charles Camichel, Toulouse, France
[3] National Institute of Informatics, Tokyo, Japan
Corresponding author: p.stankaitis@ncl.ac.uk

Abstract. In this presentation we present the development of the distributed resource allocation protocol in Event-B. Ensuring deadlock freedom of the protocol is the main challenge of this work and by using proof and stochastic simulation techniques we demonstrate the continues progress of the protocol.

Keyword: Formal Verification · Stepwise Development · Event-B

1 Introduction

Developing distributed systems is an intricate process which requires rigorous methods due concurrent nature of the distributed system. Formal methods - a mathematical model driven methods provide a systematic approach for developing complex systems. They offer an approach to specify systems precisely via a mathematically defined syntax and semantics as well as formally validate them by using semi-automatic or automatic techniques. In particular formal notations such as Event-B [1] are thought to be well suited for development and verification of various protocols. The stepwise and proof driven development provided by such methods is attractive to developers and can significantly reduce the modelling and verification effort.

In this research we attempted to develop and verify a new distributed protocol for resource allocation. By only considering an environment where message exchanges are allowed between agents and resources, and messages are allowed to arrive in any order (message delays) we identified a handful of scenarios how a distributed system could deadlock. Throughout the research we modified the protocol numerous times as additional intricate deadlock scenarios were discovered by inspecting undischarged proof-obligations or using ProB model checker [3]. The final protocol version was proved to be correct under some environment assumptions with the Rodin [4] platform whereas system liveness property was demonstrated using PRISM tool [2]. In the presentation we present our results on modelling and verification of the distributed protocol. Furthermore, we discuss potential modelling patterns for distributed systems and Event-B verification challenges we endured while developing the protocol.

Acknowledgments. This work is supported by an iCASE studentship (EPSRC and Siemens Rail Automation).

References

1. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, New York, NY, USA, 2013.
2. Andrew Hinton, Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: A tool for automatic verification of probabilistic systems. In Holger Hermanns and Jens Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 441–444, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
3. Michael Leuschel and Michael Butler. Prob: A model checker for b. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, pages 855–874, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
4. The RODIN platform. 2006. Available at https://sourceforge.net/projects/rodin-b-sharp/files/Core_Rodin_Platform/.

Design of distributed systems from choreographies specifications. A refinement based approach

Yamine Aït-Ameur[1], Sarah Benyagoub[1], Meriem Ouderni[1], and Atif Mashkoor[2]

[1] INPT-ENSEEIHT/ IRIT, University of Toulouse, France
[2] Software Competence Center Hagenberg
{yamine, sarah.benyagoub, meriem.ouederni }@enseeiht.fr
atif.mashkoor@scch.at

Today's interaction-based systems are more often built by reuse of existing distributed peers which have to coordinate with each other to fulfil client and environment requirements. Here, choreographies is a common formalism which describes the coordination as a set of conversations, i.e. the sequences of messages exchanged between the communicating peers. Checking choreography realisability is mandatory to built the third-party application with no coordination error (no deadlocks, missing messages, erroneous messaging order, etc.)

Our work proposes a stepwise development method allowing a designer to build correct-by-construction distributed systems specified as choreographies modelled with so called conversation protocols. The approach relies on a set of correct-by-construction composition operators that preserve realisability. Additionally, we consider the incremental repairability of CPs identified as non-realisable using the set of defined composition operators that satisfy sufficient conditions for realisability preservation. Reparation consists in identifying a set of changes completing intermediate non-realisable CPs so as the resulting CP becomes realisable. Our approach is validated through a successful application to CPs borrowed from the literature.

Multi-Layered Safety Architecture of Autonomous Systems: Formalising Coordination Perspective

Inna Vistbakka[1], Elena Troubitsyna[1,2], and Amin Majd[1]

[1] Åbo Akademi University, Turku, Finland
[2] KTH, Sweden
{inna.vistbakka, amin.majd}@abo.fi, elenatro@kth.se

Abstract. A pressure to deploy autonomous systems in real-life is increasing. Since exhaustive verification of safety of autonomous systems is unfeasible, the emphasis should be put on safety optimisation and run-time safety-monitoring techniques. In this paper, we propose a multi-layered architecture of autonomous systems. We define the notions of strategic, tactic and active safety – the complementary mechanisms for achieving safety. We take a swarm of drones as an example and formally define a multi-layered safety architecture and associated coordination mechanisms and underlying communication model to implement the defined complementary safety mechanisms. The derived coordination logic and communication model is formalised in Event-B framework.

Keywords: Safety, autonomous systems, coordination, formal modelling, Event-B, swarm of drones

1 Summary

Autonomous systems are capable of delivering their services in a highly independent way, i.e., without human supervision. Currently the autonomous systems, such as self-driving cars, drones etc. are getting ready for real-life deployment. Typically, autonomous systems rely on machine learning, e.g., to realise computer vision, and hence, the system behaviour continuously evolves. This makes the exhaustive pre-deployment verification of autonomous systems unfeasible. Therefore, to achieve safety of autonomous systems, we should rely on a combination of pre-deployment and run-time safety-monitoring mechanisms.

In this paper, we propose and formalise a novel multi-layered architecture to ensuring safety of autonomous systems. The main idea of our approach is to guarantee safety of autonomous systems by combining "safety-maximising" mission planning – *strategic* safety, proactive run-time safety maintenance – *tactical* safety and emergency response aiming at mitigating or removing hazard – *active* safety. Each type of safety mechanisms is implemented at the corresponding architectural layer.

While designing autonomous systems, typically we should trade off safety with other system characteristics, such as performance, quality of service, etc.

For instance, let us consider a swarm of drones. To accomplish the required mission, the drones should have a high probability of surviving throughout the entire mission, i.e., they should not prematurely deplete their batteries. This requires minimisation of energy consuming functions, e.g., travel distance, communication etc. However, such a minimisation might result in planning a mission that has a high risk of drone collision. In this context, strategic safety can be implemented by an optimisation algorithm that uses safety, i.e., the risk of collisions, and the travel distance as the parameters of the optimisation. Strategic safety relies on high performance optimisation algorithms and hence can be implemented in cloud.

When the swarm starts to move, the drones can deviate from the predefined routes due to unforseen environmental factors, e.g., wind, or internal problems, e.g., transient hardware failures of drones. The positions of the drones should be continuously monitored and their routes recalculated to maintain efficiency and safe distance between the drones. Such activities are called tactical safety. Tactical safety can be implemented at the edge.

Finally, an unexpected obstacle might appear in the flight zone of the swarm. To avoid a collision, the active safety – an emergency stopping or safe maneuver should be executed. The active safety requires fast response and should be implemented locally, at the drone level.

Such a multi-layered distributed implementation of the safety mechanisms requires a complex coordination scheme that cater to variety of situations: drone and communication failures, changes in the external environment such as unexpected appearance of obstacles etc. To design the proposed multi-layered safety architecture, we rely on formal modelling in Event-B [1].

Event-B is a state-based approach to correct-by-construction system development. The main development technique – refinement – supports stepwise construction and verification of complex specifications. We start the development by creating a high-level abstract specification, which is incrementally augmented to unfold the entire multi-layered architecture and verify correctness of the interplay between the safety mechanisms at different architectural levels. Abstraction, refinement and proofs as well as automated tool support allow us to scale the formal development to such complex problem.

References

1. Abrial, J.R.: Modeling in Event-B. Cambridge University Press (2010)
2. Aniculaesei, A., Arnsberger, D., Howar, F., Rausch, A.: Towards the Verification of Safety-critical Autonomous Systems in Dynamic Environments. In: V2CPS. pp. 79–90. EPTCS 232 (2016)
3. Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Ilic, D., Latvala, T.: Supporting Reuse in Event-B Development: Modularisation Approach. In: ABZ 2010. pp. 174–188. Springer (2010)
4. Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Väisänen, P., Ilic, D., Latvala, T.: Verifying Mode Consistency for On-Board Satellite Software. In: SAFECOMP 2010. pp. 126–141. Springer (2010)
5. Iliasov, A., Romanovsky, A., Laibinis, L., Troubitsyna, E., Latvala, T.: Augmenting Event-B Modelling with Real-time Verification. In: Proc. FormSERA 2012. pp. 51–57. IEEE (2012)
6. Ilic, D., Troubitsyna, E.: Formal Development of Software for Tolerating Transient Faults. In: Proc. (PRDC 2005. pp. 140–150. IEEE Computer Society (2005)
7. K.Macek, D.Vasquez, T., R.Siegwart: Safe Vehicle Navigation in Dynamic Urban Scenarios. In: Proc. of 11th International IEEE Conference on Intelligent Transportation Systems. pp. 482–489. IEEE (2008)
8. Laibinis, L., Troubitsyna, E., Iliasov, A., Romanovsky, A.: Rigorous Development of Fault-Tolerant Agent Systems. In: M. Butler, C. Jones, A.R., Troubitsyna, E. (eds.) Rigorous Development of Complex Fault-Tolerant Systems. LNCS, vol. 4157, pp. 241–260. Springer (2006)
9. Laibinis, L., Pereverzeva, I., Troubitsyna, E.: Formal Reasoning about Resilient Goal-Oriented Multi-Agent Systems. Sci. Comput. Program. 148, 66–87 (2017)
10. Laibinis, L., Troubitsyna, E.: Fault Tolerance in a Layered Architecture: A General Specification Pattern in B. In: Proc. (SEFM 2004). pp. 346–355. IEEE Computer Society (2004)
11. Leveson, N., Pinnel, L.D., Sandys, S.D., Koga, S., Reese, J.D.: Analyzing Software Specifications for Mode Confusion Potential. In: Human Error and System Development. pp. 132–146 (1997)
12. Majd, A., Ashraf, A., Troubitsyna, E., Daneshtalab, M.: Integrating Learning, Optimization, and Prediction for Efficient Navigation of Swarms of Drones. In: Proc. PDP 2018. pp. 101–108. IEEE Computer Society (2018)
13. Majd, A., Ashraf, A., Troubitsyna, E., Daneshtalab, M.: Using optimization, learning, and drone reflexes to maximize safety of swarms of drones. In: Proc. 2018 IEEE Congress on Evolutionary Computation, CEC 2018. pp. 1–8. IEEE (2018)
14. Majd, A., Troubitsyna, E.: Integrating safety-aware route optimisation and runtime safety monitoring in controlling swarms of drones. In: Proc. ISSRE Workshops. pp. 94–95. IEEE Computer Society (2017)
15. Majd, A., Troubitsyna, E., Daneshtalab, M.: Safety-Aware Control of Swarms of Drones. In: Proc. SAFECOMP 2017. LNCS, vol. 10489, pp. 249–260. Springer (2017)
16. Pereverzeva, I., Troubitsyna, E.: Formalizing Goal-Oriented Development of Resilient Cyber-Physical Systems. In: Alexander Romanovsky, F.I. (ed.) Trustworthy Cyber-Physical Systems Engineering, chap. 6 (2017)
17. Pereverzeva, I., Troubitsyna, E., Laibinis, L.: A Case Study in Formal Development of a Fault Tolerant Multi-robotic System. In: Proc. SERENE 2012. LNCS, vol. 7527, pp. 16–31. Springer (2012)

18. Pereverzeva, I., Troubitsyna, E., Laibinis, L.: Development of Fault Tolerant MAS with Cooperative Error Recovery by Refinement in Event-B. CoRR abs/1210.7035 (2012), http://arxiv.org/abs/1210.7035
19. Pereverzeva, I., Troubitsyna, E., Laibinis, L.: Formal Development of Critical Multi-agent Systems: A Refinement Approach. In: Proc. EDCC 2012. pp. 156–161. IEEE Computer Society (2012)
20. Pereverzeva, I., Troubitsyna, E., Laibinis, L.: Formal Goal-Oriented Development of Resilient MAS in Event-B. In: Proc. Ada-Europe 2012. LNCS, vol. 7308, pp. 147–161. Springer (2012)
21. Pereverzeva, I., Troubitsyna, E., Laibinis, L.: A Refinement-Based Approach to Developing Critical Multi-Agent Systems. IJCCBS 4(1), 69–91 (2013)
22. Petti, S., Fraichard, T.: Partial Motion Planning Framework for Reactive Planning within Dynamic Environments. In: Proc. of the IFAC/AAAI Int. Conf. on Informatics in Control, Automation and Robotics (2005)
23. Prokhorova, Y., Laibinis, L., Troubitsyna, E., Varpaaniemi, K., Latvala, T.: Derivation and Formal Verification of a Mode Logic for Layered Control Systems. In: Proc. APSEC 2011. pp. 49–56. IEEE Computer Society (2011)
24. Rodin: Event-B platform: online at `http://www.event-b.org/`
25. Siegwart, R., I.R.Nourbakhsh: Introduction to Autonomous Mobile Robots. MIT Press (2004)
26. Tarasyuk, A., Pereverzeva, I., Troubitsyna, E., Latvala, T., Nummila, L.: Formal Development and Assessment of a Reconfigurable On-board Satellite System. In: Proc. SAFECOMP 2012. pp. 210–222. Springer (2012)
27. Tarasyuk, A., Pereverzeva, I., Troubitsyna, E., Laibinis, L.: Formal Development and Quantitative Assessment of a Resilient Multi-robotic System. In: Proc. SERENE 2013. LNCS, vol. 8166, pp. 109–124. Springer (2013)
28. Tarasyuk, A., Pereverzeva, I., Troubitsyna, E., Latvala, T.: The Formal Derivation of Mode Logic for Autonomous Satellite Flight Formation. In: Proc. SAFECOMP 2015. LNCS, vol. 9337, pp. 29–43. Springer (2015)
29. Tarasyuk, A., Troubitsyna, E., Laibinis, L.: Towards Probabilistic Modelling in Event-B. In: Proc. IFM 2010. LNCS, vol. 6396, pp. 275–289. Springer (2010)
30. Tarasyuk, A., Troubitsyna, E., Laibinis, L.: Integrating Stochastic Reasoning into Event-B Development. Formal Asp. Comput. 27(1), 53–77 (2015)
31. Th.Fraichard: A Short Paper about Motion Safety. In: Proc. Of the IEEE Int. Conf. on Robotics and Automation. IEEE (2007)
32. Troubitsyna, E.: Reliability Assessment through Probabilistic Refinement. Nord. J. Comput. 6(3), 320–342 (1999)
33. Troubitsyna, E.: Stepwise Development of Dependable Systems. In: Ph.D. thesis No.29. Turku Centre for Computer Science (2000)
34. Troubitsyna, E.: Developing Fault-Tolerant Control Systems Composed of Self-Checking Components in the Action Systems Formalism. In: Proc. FACS 2003. pp. 167–186 (2003)
35. Vistbakka, I., Majd, A., Troubitsyna, E.: Deriving Mode Logic for Autonomous Resilient Systems. In: Proc. ICFEM 2018. LNCS, vol. 11232, pp. 320–336. Springer (2018)
36. Vistbakka, I., Majd, A., Troubitsyna, E.: Multi-layered Approach to Safe Navigation of Swarms of Drones. In: Proc. SAFECOMP 2018 Workshops. LNCS, vol. 11094. Springer (2018)
37. Yanmaz, E., Yahyanejad, S., Rinner, B., Hellwagner, H., Bettstetter, C.: Drone networks: Communications, coordination, and sensing. Ad Hoc Networks 68, 1–15 (2018)

Embedding Formal Specifications as Libraries into Applications

Philipp Körner ✉, Michael Leuschel

Institut für Informatik, Universität Düsseldorf
Universitätsstr. 1, D-40225 Düsseldorf, Germany
p.koerner@uni-duesseldorf.de
leuschel@cs.uni-duesseldorf.de

Abstract. In the past decades, the increase in memory and computational power gave rise to executable specifications and associated techniques, e.g., model checking. Yet, executing a particular path fragment in production usually requires code generation.
Here, we discuss another approach: Instead of generating code, we embed a model checker into applications to directly execute formal models. In particular, we use ProB 2.0, a Java API to the ProB animator and model checker. We present this API as well as several projects that use it to interact with a formal specification at run-time.

1 Introduction

Animators and model checkers are well-known tools that are used during development of a formal model in order to ensure correctness. They require specifications to be executable though. When considering executable specifications, usually one has code generation in mind. In the case of B and Event-B, which are high-level formalisms, code generators usually require refinement to an implementation level which is both cumbersome and very low-level.

The idea presented here is as follows: model checkers for these formalisms, e.g., ProB, can deal with high-level constructs via constraint solving techniques. With a high-level API, one can execute state transitions (aka operations or events) individually and access the successor state. This allows to abuse the specification languages as a high-level *programming* language, and to ship formal specifications as libraries. For this, we use the aforementioned specification languages, B and Event-B, and the model checker ProB. To our knowledge, this approach is unique and has not been used in combination with any other formalism or animation tool.

In the following, we briefly introduce two high-level specification language, B and Event-B, as well as ProB, an animator and model checker for these languages. Afterwards, we present ProB 2.0, an API that allows us to interact with ProB in Section 2. Following, we discuss several applications based on ProB 2.0

1.1 B, Event-B and ProB

Both B [2] and its successor, Event-B [1], are state-based specification languages that allow for high levels of abstraction. They are based on first order logic and set theory. Further, they make use of general substitution for state modifications, and of refinement calculus [3,4] to describe models at different levels of abstraction [5].

The idea behind both formalisms is that, starting from an abstract state machine, the model is iteratively refined to an implementation level that can be used for code generation. Yet, this process is often cumbersome, requiring proof linking the different refinement levels as well as elimination of high-level constructs such as sets, quantifiers and non-determinism.

ProB [10] is an animator and model checker for both B and Event-B. Its core is developed mainly in SICStus Prolog [7], with some parts being implemented in C and Java, and makes use of co-routines and SICStus' CLP(FD)library [6]. Besides B, ProB offers support for several other formalisms as well, including TLA$^+$ (via translation to B), Z, CSP and many more.

2 ProB 2.0

As ProB is written in Prolog, which admittedly is neither the most popular nor the easiest to pick up language, it is hard for formal method experts to use anything but the default animation and model checking capabilities. Thus, a main design goal of ProB 2.0 was to offer access to the API of ProB via a scripting language that allows easy embedding of domain specific languages (DSLs). For this, we picked and embraced Groovy, a dynamic programming language running on the JVM, which is (almost) a superset of Java. There are two APIs, a high-level and a low-level one. Both are documented in the handbook available under `https://www3.hhu.de/stups/handbook/prob2/prob_developer.html`.

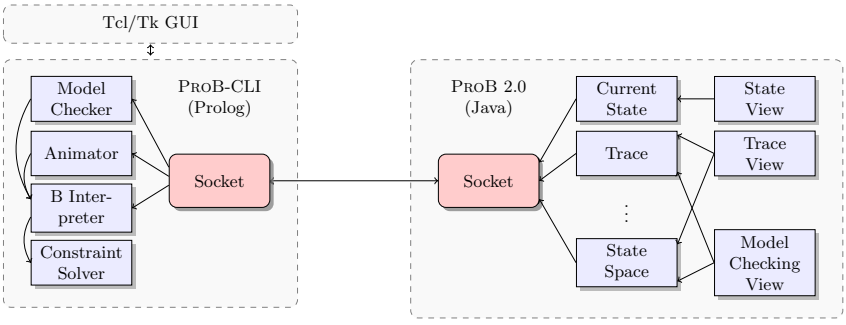

Fig. 1: Overview of the ProB Ecosystem

A general overview of ProB 2.0 is given in Fig. 1. For each B model that is interacted with, an instance of the ProB-CLI (command line interface), which

actually loads the model, is started in socket-mode. This means that the ProB-CLI listens on a socket for commands to execute (whitelisted) Prolog code. The whitelist offers fine-grained access to ProB's constraint solving, animation and model checking capabilities as well as ProB's preferences and machine components.

The high-level API abstracts away from ProB's internals in Prolog. Two objects in ProB 2.0 play a central role for animation: the state space and the history. Each state discovered during model checking is part of a state space, which is a representation of the underlying labelled transition system. Exploring the state space by executing operations adds transitions and newly encountered states. The history behaves like a browser history: It is append-only, but it is possible to "go back in time" and start a new fork. Executing an operation during animation automatically appends the successor state to the history.

In order for states to be meaningful, they require a representation in ProB 2.0. By default, ProB provides a string representation to ProB 2.0. Yet, they can be translated into Java data structures: For example, B integers are translated into BigIntegers, B sets correspond to Java sets and sequences to Java lists. Naturally, this translation does not work for infinite sets.

3 Examples

In this section, we describe different use cases based on several examples. The first couple of examples we want to discuss are student projects implementing two well-known games: Pac-Man and Chess. Furthermore, the approach found use in two more complex projects, namely a safety critical, industrial application for the ETCS Hybrid Level 3 concept, and a timetable planner for university courses. In all four examples we translate the current state of the model into Java data structures in order to provide an (interactive) visualisation.

3.1 Real-Time Animation: Pac-Man

Our first example application is based on a formal model of Pac-Man. Our version of the game has two modes: firstly, the user can control the Pac-Man by pressing the arrow keys. Then, the model is automatically driven (read: animated) by a loop and can be run in real-time. Note that the model is non-deterministic in the sense that there are multiple available operations, one for each direction the Pac-Man may move. The user can change the direction and (according to the rules) the Pac-Man will keep moving into that direction until it hits a wall. In the second gamemode, the game plays itself, i.e., the Pac-Man is moved by a heuristic implemented in Groovy. In both cases, the movement of all ghosts is controlled by Pac-Man rules implemented in Groovy code.

Performance can be an issue for real-time applications. In the case of Pac-Man, on modern machines it usually runs fairly well as the calculation of the next-state function is very fast.

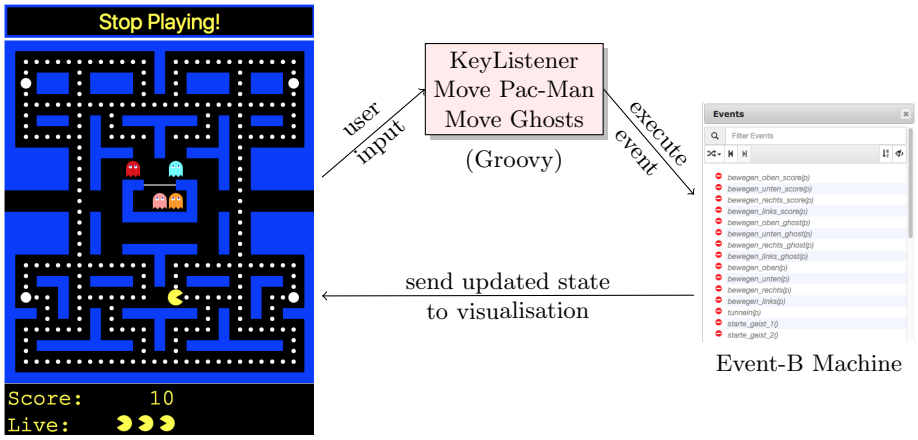

Fig. 2: Architecture of a Pac-Man Game Based on a Formal Model

3.2 Predicting the Future: Chess

In the chess example, we have two use cases. Firstly, we want two (human) players to be able to play against each other. Secondly, a (simple) chess AI should be available to play against. As with Pac-Man, we use the formal specification in order to specify the rules of the game. The model offers all possible moves as enabled actions, checkmate is encoded as an invariant violation. Then, we can use the vanilla ProB animator to play chess (preferably with an additional visualisation of the current state).

The more interesting part is the AI which is hard to specify but somewhat easy to implement. Thus, we write the AI in Java using ProB 2.0: we implemented a Minimax algorithm with alpha-beta pruning. The calculated tree has the current state at its root, its children are the successor states (representing all valid turns by the AI), their children are their corresponding successor states (where each state represents a turn by the human player) and so forth. For termination, we limit the depth of the state space that should be explored. This depth determines the AI's strength. We use a simple evaluation function (that only depends on a single state) in order to assign a weight to each state. Then, we pick the turn suggested by Minimax.

This strategy is very similar to bounded model checking. Yet, we do not care for invariant violations in particular (an invariant violation suggests that one player wins the game which the AI accounts for). Instead, we continue execution of the model using an action that guarantees the most favourable outcome.

This case study offers worse results than Pac-Man. Due to the state space explosion caused by the sheer amount of possible moves, generating all successor as deep as required by a strong chess engine is infeasible. An implementation in, e.g., plain C or Java is orders of magnitudes faster. Only a small part of the

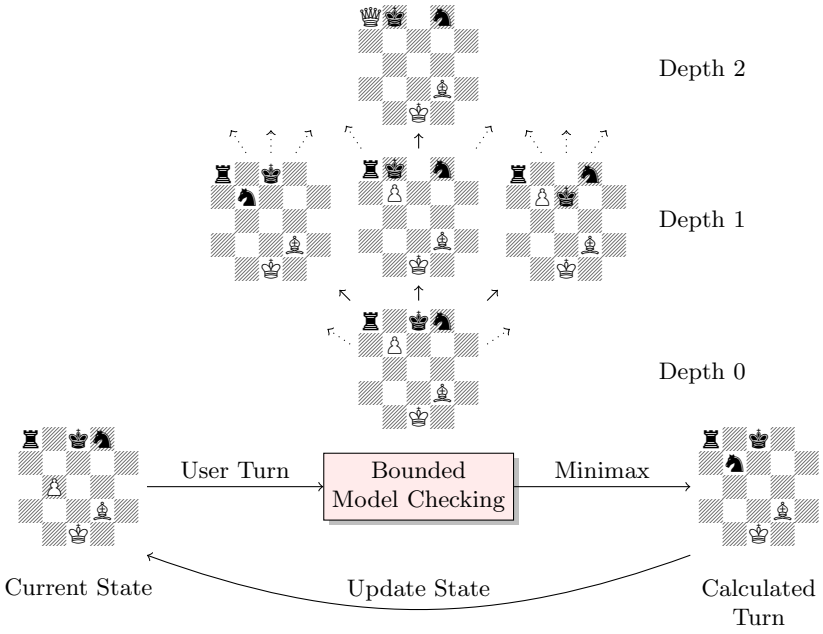

Fig. 3: Architecture of Chess Based on a Formal Model

state space from a given board position can be generated in reasonable time, which results in the AI being a rather weak opponent.

3.3 Real Time Animation: ETCS Hybrid Level 3 Concept

We also used ProB 2.0 in an industrial project, that became a case study for ABZ 2018 later on [8], as well as its successor project. Unfortunately, we are not allowed to disclose the B model and application itself.

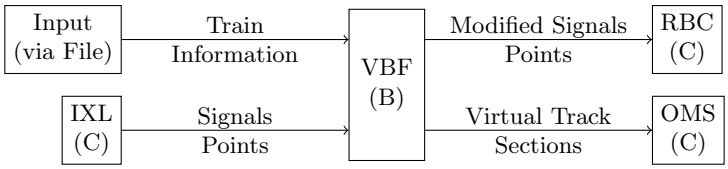

Fig. 4: Architecture of the HL3 Prototype

The main idea is that by using a Virtual Block Function (VBF), the capacity of existing railroad networks can be increased as track sections can be subdivided even without adding sensors or other hardware to the tracks. The main part of the VBF was written as a B model.

Real, existing components running in separate processes or even (virtual) machines are able to feed information into the model (by tailing logfiles and calling wrapper interface functions implemented in C) in order to drive the formal model. The resulting tool was used in order to control real signals and trains on a test track. A video of the demonstration can be found under `https://www.youtube.com/watch?v=FjKnugbmrP4`, where the GUI implemented via ProB 2.0 is seen starting at 2:30.

In our naive approach, data structures grow rather large and repeated communication of all variables and constants results in an unacceptable performance degradation, since the calculated results should be able to reach the components in a few to hundreds of milliseconds. With a few performance enhancements, e.g. pre-filtering data structures for possible changes and removing constants from the communicated data, the application performs well enough on a regular notebook computer.

3.4 ProB as a Constraint Solver: PlüS

PlüS is an application for planning university timetables [11,12] and is available under `https://plues.github.io/`. The goal is to show that, starting from the current timetables, it is possible for students to finish their studies in legal standard time for all courses or combinations of major and minor subjects. If a course or a combination is found to be infeasible, the smallest conflicting set of classes and timeframes should be provided such that it can be fixed manually. Complete re-generation of timetables is avoided due to informal agreements, e.g., lecturers prefer given timeslots or are unavailable on certain days.

A database stores information about all courses, e.g., for which subject they can be attributed, whether they are mandatory or if other courses are required. From this database, a B model is compiled. This is included in another B machine that allows checking feasibility of a subject, move lectures etc. from one time-slot to another and to calculate the unsatisfiable core if applicable.

On top of this functionality, a UI in JavaFX is implemented. Using the trace of the model, i.e., considering movement of sessions to different timeslots, timetables can be generated as PDF files. The interaction with ProB is absolutely hidden from the user.

4 Conclusion

In this paper, we presented ProB 2.0 which offers a Java API to the ProB model checker. ProB 2.0 renders it possible to write applications that interact with a formal model at runtime, offering declarative programming, rapid prototyping and easy debugging.

The main concern for real-life applications, as already stated by [9], is performance. Because computers grew a lot faster since the early 90s, it is an option that is viable to explore. Yet, applications written in traditional imperative, functional or even logical programming languages are by orders of magnitudes

faster because they have to work at lower levels of abstraction. For many time-critical applications, however, this is not the way to go yet. As long as mediocre performance suffices or is not a concern at all (e.g., if data sets are rather small), utilising formal models allows us to quickly write usable applications that also can make use of the eco-system associated with formal methods, e.g., generating and discharging proof obligations, or model checking.

Acknowledgement. Most of the presented work is the work of other people. Christoph Heinzen is the original author of the presented Pac-Man application and Philip Höfges wrote the chess model, AI and GUI. Additionally, we want to thank the many people who were involved in the development of both ProB and ProB 2.0, the Slot Tool and the ETCS Hybrid Level 3 case study.

References

1. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 1st edition, 2010.
2. J.-R. Abrial, M. K. Lee, D. Neilson, P. Scharbach, and I. H. Sørensen. The B-method. In *Proceedings VDM*, volume 552 of *LNCS*. Springer, 1991.
3. R. Back. On correct refinement of programs. *Journal of Computer and System Sciences*, 23(1):49–68, 1981.
4. R.-J. Back and J. Wright. *Refinement calculus: a systematic introduction*. Springer Science & Business Media, 2012.
5. D. Cansell and D. Méry. Foundations of the B method. *Computing and informatics*, 22(3-4):221–256, 2012.
6. M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In *Programming Languages: Implementations, Logics, and Programs*, volume 1292, pages 191–206. Springer, 1997.
7. M. Carlsson, J. Widen, J. Andersson, S. Andersson, K. Boortz, H. Nilsson, and T. Sjöland. *SICStus Prolog user's manual*, volume 3. Swedish Institute of Computer Science Kista, Sweden, 1988.
8. D. Hansen, M. Leuschel, D. Schneider, S. Krings, P. Körner, T. Naulin, N. Nayeri, and F. Skowron. Using a Formal B Model at Runtime in a Demonstration of the ETCS Hybrid Level 3 Concept with Real Trains. In M. Butler, A. Raschke, T. S. Hoang, and K. Reichl, editors, *Proceedings ABZ 2018*, volume 10817 of *LNCS*, pages 292–306. Springer, 2018.
9. I. J. Hayes and C. B. Jones. Specifications are not (necessarily) executable. *Software Engineering Journal*, 4(6):330–339, 1989.
10. M. Leuschel and M. Butler. ProB: A model checker for B. In *Proceedings FME*, volume 2805 of *LNCS*. Springer, 2003.
11. D. Schneider. *Constraint Modelling and Data Validation Using Formal Specification Languages*. PhD thesis, Heinrich-Heine-Universität Düsseldorf, 2017.
12. D. Schneider, M. Leuschel, and T. Witt. Model-based problem solving for university timetable validation and improvement. *Formal Aspects of Computing*, pages 545–569, 2018.

Change Impact Analysis in Refinement-based Formal Specification

Shinnosuke Saruwatari

The University of Tokyo, Japan
s-saruwatari@nii.ac.jp

Models developed with formal method language are rigorous but there is a problem that it is costly to develop them. If the formal model we want is a variant of a model that is already developed, it is natural that we want to reuse it. We developed a method for extracting information that is useful for reusing already existing models from them.

One problem with reusing models is that we need to comprehend the characteristic of refinements. This is because we can implement specifications in arbitrary order to some extent and from the view of verification and another reuse, it is important to modify models in a way that suits their characteristics of refinements.

Another problem is that we need to confirm the consistency of specifications between different abstraction levels. Comprehending restrictions like guards and invariants becomes difficult according to the size of the model, especially when we have little knowledge of the model.

The main reason for these problems is that predicates in formal models have implicit and complex relationships between before and after refinements. Our method focuses on extracting information about these relationships and information that show us how we should deal with a formal model. The method we propose here labels guards with their relationships between before and after refinements. We also stick labels that present information that can be used as clues for modifying models on each action.

There are two types of label in this method, label on guards and label on actions, and we deal them in a different way since they contain a different type of information. The former provides information on how the model was constructed through stepwise refinements and the latter represents how difficult their actions are to be modified and give us hints for modifying them without losing consistency.

To label each predicate, our method first uses SMT solver. The SMT solver analyzes guards whether their restrictions become stricter than before refinement and actions whether there are any other values that do not break the consistency of the model. Then, each predicate is labeled according to the result of SMT solver with the consideration of the structure of the model.

There are four labels for guards and actions each. For label on guards, there are *Limited*, *Divided*, *Replaced*, and *Unchanged*. *Limited* shows that the restriction of guard becomes stricter through refinement. *Divided* shows that the situation when its event can operate is divided into several different situations

through refinement. *Replaced* shows that the situation when its event can operate does not change through refinement but has some variables replaced with other variables. *Unchanged* shows that the guard does not change through refinement. For label on actions, there are *Suggested*, *Restricted*, *Adapted*, and *Untouched*. *Suggested* shows that there are other values that can be substituted in the action without changing other predicates in its event. *Restricted* shows that some changes within the event are needed to modify the action. *Adapted* shows that the action cannot be modified unless invariants are changed.

We have conducted the experiment with subjects of Smart Grid model and Bridge model. We examined whether the labels support understanding the characteristic of stepwise modeling and modifying models without breaking the consistency of models.

For the first question, whether the labels support understanding the characteristic of stepwise modeling, we focused on labels labeled on guards most in each model. Smart Grid model has *Divided* label most and can be considered that events in the abstract model were comprehensive of various cases and divided into specific cases through refinements. For Bridge model, however, has *Limited* label most and can be considered that most events are already prepared in abstract model and the model's refinements are mainly focused on depicting specifications in detail.

For the second question, whether the labels support modifying models without breaking the consistency of models, we focused on the ratio of labels on actions in each model. Smart Grid model has 6 actions that have *Suggested* label, which means that they can take other values for substitution. On the other hand, the Bridge model has only 2 actions that have *Suggested* label. However, there are 10 actions labeled with *Restricted* label and this shows that these actions can be modified if other actions or guards are modified together.

From the experiment, we confirmed that the labels we proposed are able to extract information of refinement characteristics and modifiable predicates that are useful for modifying models.