ProB2-UI: A Java-Based User Interface for ProB

Jens Bendisposto(✉), David Geleßus(✉), Yumiko Jansing, Michael Leuschel(✉), Antonia Pütz, Fabian Vu(✉), and Michelle Werth

Institut für Informatik, Universität Düsseldorf, Universitätsstr. 1, 40225 Düsseldorf, Germany
{bendisposto,dagel101,leuschel,fabian.vu}@hhu.de

Abstract. ProB2-UI is a modern JavaFX-based user interface for the animator, constraint solver, and model checker ProB. We present the main features of the tool, especially compared to ProB's previous user interfaces and other available tools for B, Event-B, and other formalisms. We also present some of ProB2-UI's history as well as its uses in the industry since its release in 2019.

1 Introduction and Motivation

This paper presents ProB2-UI, a JavaFX-based user interface for the animator, constraint solver, and model checker ProB. The core of ProB is written in SICStus Prolog and supports formalisms such as B, Event-B, Z, TLA$^+$ and Alloy. Initially, ProB had a Tcl/Tk interface, first presented in 2003 [28], but with roots dating back to around 2000. Later the command-line interface probcli was developed, which is still being heavily used for testing, data validation, and batch verification. For example, probcli is used for data validation of railway systems, see [7,27] or Sect. 4 of [3].

Around 2005 we started to integrate ProB into the Rodin tool for Event-B, requiring integration with Java and the Eclipse user interface. For this purpose, an interface was added to probcli, allowing Java code to control ProB by sending commands over a socket. This resulted in the first version of ProB for Rodin. Its user interface was more intuitive, appealing and modern than ProB for Tcl/Tk, but did only provide a limited set of features (e.g., it had no state space visualisation or projection features).

At that moment we had to decide whether to fully focus on Rodin and Eclipse, or keep ProB and its user interface independent of it. In the end, we decided to develop a new lightweight Java API allowing end-users to customize ProB independent of Eclipse. This resulted in the development of the ProB2 Java API, which was available in 2014 (cf. Sect. 2 of [20]). After several unfruitful attempts at developing a new user interface, we started to work on a JavaFX user interface in 2016. The first stable version 1.0.0 was released in 2019. ProB2-UI has been used within our team for a variety of case studies, e.g., for an automotive case study [29]. It has also already been used for several industrial applications

A. Lluch Lafuente and A. Mavridou (Eds.): FMICS 2021, LNCS 12863, pp. 193–201, 2021.
https://doi.org/10.1007/978-3-030-85248-1_12

(many confidential). In particular, ProB2-UI was used successfully during the demonstration of the ETCS hybrid level 3 concepts [13,14] at runtime. Here, model traces were captured during field tests and could be replayed along with a visualisation plugin for fine-grained retrospective analysis. Another use was made for modelling key properties of a CBTC zone controller in [6].

2 Features of ProB2-UI

Figure 1 shows the main window of ProB2-UI. On the right-hand side, you can see the *project view*. A project contains a group of (related) models along with preference settings and configurations for validation and verification tasks. This lets the user easily re-check these tasks, e.g., after modifying a model, without having to re-enter the parameters. The project information is stored in a project file; it contains a list of models and tasks with their status, a list of preference settings along with visualisation, simulation, and trace files. An overview of the status of all verification tasks is shown in the "status" tab of the project view.

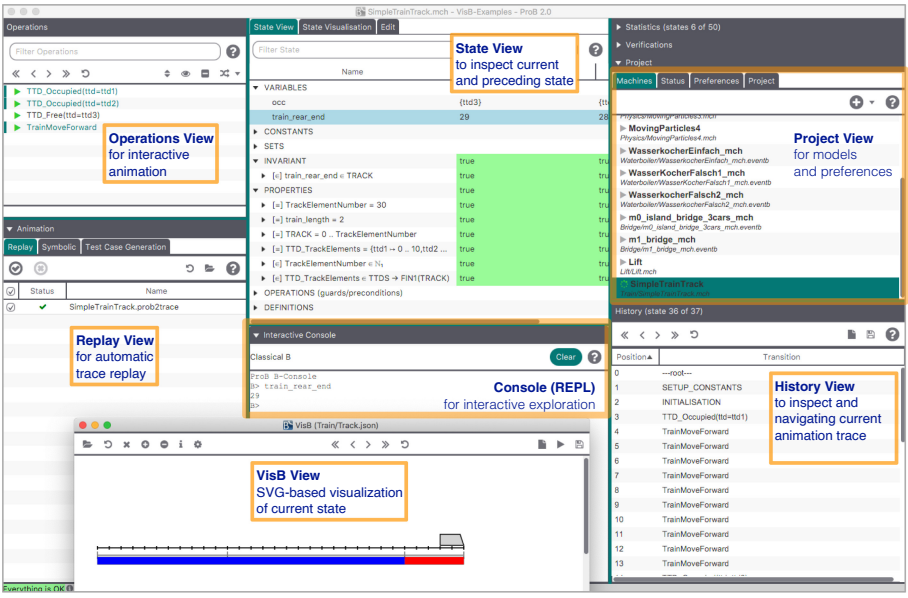

Fig. 1. Main window of ProB2-UI

Animation. After a model has been loaded, the user can animate the model. In contrast to simulation, an animator allows the user to interactively select the execution steps. This is done in the *operations view*, where ProB has precomputed all enabled operations (up to a user-provided limit). By clicking on an operation, the current state of the model is changed, which can be seen in the state view. It is also possible to enter some of the operation's parameters and

an optional post-condition to execute an operation. This is for example useful when not all operations were precomputed by ProB.

The *state view* displays the current and previous state of the model, highlighting differences. In addition to variables and constants, the view also shows predicates such as the properties (aka axioms), invariants, or the guards of operations. Complex expressions and predicates can be expanded to inspect the values of individual sub-expressions/predicates.

During the interaction with the model, a trace of all executed operations and visited states is maintained. One can navigate through this trace in the *history view*, e.g., to return to a previous state and then execute an alternate operation. The history can be saved to a file. The saved trace files are shown in the *replay view* on the left-hand side, where they can be replayed by a simple click. These trace files contain the exact parameter and variable values for each animation step, which allows replaying the trace exactly even if some operations are not deterministic.

Visualisation. ProB2-UI provides various ways to *visualise* the currently animated model. For example, it is possible to visualise the state space, the machine hierarchy, event hierarchy, formula trees etc. Figure 2 shows a state space projection [25] of the model from [29] on the right-hand side.

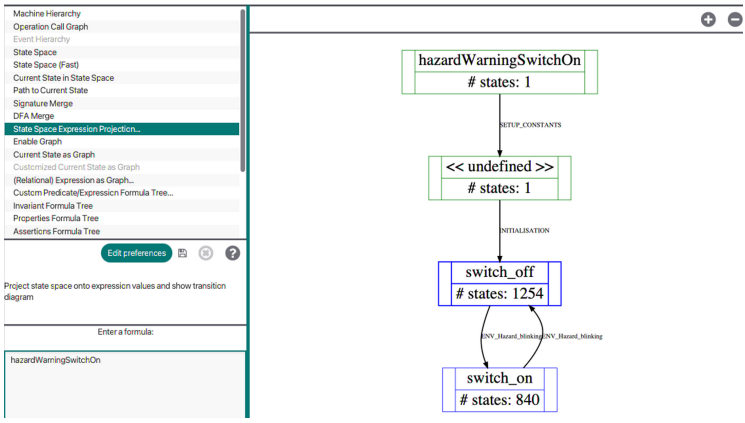

Fig. 2. Graph visualisation window of ProB2-UI with projected state space

The visualisation tool VisB [39] can be used to create interactive visualisations using SVG images and a glue file. It was previously a ProB2-UI plugin, but is now available in the regular *VisB view*, as shown at the bottom of Fig. 1. In the view, the user can see a visual rendering of the current state and execute operations by clicking on visual elements (as stipulated in the glue file). A very recent feature is the export of an animation trace as a stand-alone HTML file containing the visualisations. This HTML file can, e.g., be sent via email to domain experts and they can inspect the trace with its states in a regular browser, without any need for ProB2-UI.

Using ProB2-UI's plugin support, custom visualisations can be implemented using Java and JavaFX. Developing such a visualisation requires more effort and knowledge than VisB, but offers greater flexibility via arbitrary GUI elements and code. Such a visualisation plugin was developed as part of a demonstration of the ETCS Hybrid Level 3 concept [13,14], and was instrumental in domain experts understanding the model and discovering issues. More figures and screenshots of ProB2-UI can be seen in [13,14] (Fig. 3).

Figure 3 of [6] shows another ProB2-UI visualisation plugin for an Alstom CBTC system. The plugin mechanism was also used to provide custom views for a domain-specific extension of B for data validation [15].

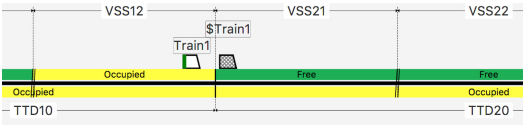

Fig. 3. Figure 3 from [13] with physical train and train image

Verification. As explained before, a modeller can save verification tasks for each model, which is particularly useful during development. It is possible to re-check all verification tasks on an updated model.

ProB2-UI supports various model checking techniques, including exhaustive model checking [28], LTL model checking [32], and symbolic model checking [22] (Fig. 4).

Regarding explicit-state model checking, one can choose different search strategies, e.g., breadth-first, depth-first, and mixed. Furthermore, one can define the properties to be checked in each state. These include the invariant, deadlock-freedom, assertions, well-definedness, and preconditions. It is also possible to provide a goal predicate to specify desired states, or limit the number of states explored or the time spent. In the case that an error or a goal is found, the counterexample is shown in the history view of the animator.

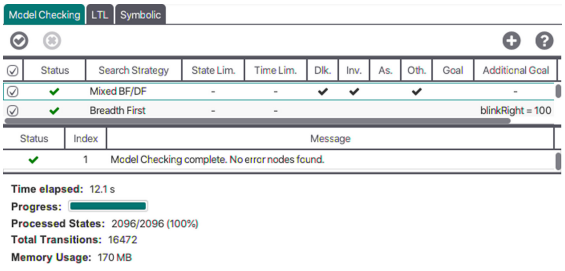

Fig. 4. Model checking view

LTL model checking can be used to verify temporal properties of a model. Here, the user can define LTL formulas combining state predicates with the standard LTL operators. In addition, operators of Past-LTL are available, as well as operators for fairness, deadlock, determinacy, and enabledness of operations. Finally, it is possible to declare LTL patterns, which

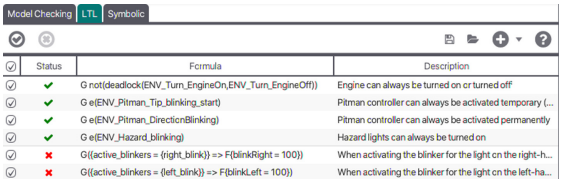

Fig. 5. LTL model checking view

can be reused in multiple formulas. Similar to explicit-state model checking, counterexamples are also stored and can be displayed in the history view.

Simulation. Recently, ProB2-UI has been extended by a simulator called SimB [38]. It is based on lightweight annotation files that specify how events activate each other along with timing, priorities, and probabilistic annotations as well as possible start and end conditions. The user can then perform simulations, where the model is simulated live in real-time. In combination with the other views, in particular the VisB view, one can see how the model's state changes in real-time. It is also possible to perform Monte Carlo simulations in accelerated time. Based on the simulations, statistical techniques such as hypothesis testing and estimation can then be applied to validate probabilistic and timing properties. Every single simulation can be saved as a *timed trace*, or be replayed in real-time afterwards (Fig. 5).

Other Features. In addition to the features presented so far, ProB2-UI also provides an editor to modify the models in the project. Furthermore, there are three consoles: a B console in which formulas in B or Event-B can be evaluated, a Groovy console in which one has access to the objects of the ProB2 API, and a Prolog console to inspect debugging or performance messages of the ProB kernel. In addition to high-level languages such as B, Event-B, Z, TLA+, Alloy or CSP, ProB also supports XTL files, which contain a raw Prolog encoding of the transition system to be checked. Here, the modeller has to implement the interface to ProB consisting of the predicates `start/1` (calculating the initial states), `trans/3` (calculating transitions and resulting states outgoing from every state), and `prop/2` (calculating the properties of every state). We have used it for teaching, e.g., by encoding the rules of chess in Prolog and illustrate game-playing algorithms. It is also possible to provide a Prolog interpreter for another specification language and extend ProB and ProB2-UI in this way. For example, we have such interpreters for Promela [40], SMV, and Lustre [37] models.

3 Related Work

In this section, we compare ProB2-UI with other formal methods tools that come with a graphical user interface. The reader may also consult various recent studies [8–10,35] about tools and also their usability.

Atelier-B [5] and Rodin [1] have a strong focus on the refinement-based software development process of the B method. They both provide a project or workspace concept, with a convenient way for managing and discharging proof obligations. As mentioned, ProB can also be run within [1], but with a much-reduced feature set compared to ProB2-UI. It is also possible to start ProB2-UI from [1] for a given model. From Atelier-B it is also possible to start ProB Tcl/Tk for a model, or use ProB as an alternate prover. At the moment, ProB2-UI is complementary to Atelier-B and Rodin: ProB2-UI focuses on validation and

model checking, Atelier-B and Rodin on proof and proof obligations.[1] Other animation tools for B, such as Brama [34], AnimB [30] or JeB [41] provide convenient visualisation features, but lack many of the features of ProB2-UI.

The TLA Toolbox is an IDE for the modelling language TLA+ which is widely used for distributed systems. Both the TLC model checker [42] and the TLA+ proof system are integrated into the TLA Toolbox [23]. In practice, most users will use the TLC features, where similar to ProB2-UI, one can save various model checking configurations. Counterexample traces can be inspected and a REPL has been added recently to the toolbox. However, at the moment, there are no interactive animation or visualisation features available. Here, a ProB plugin for the TLA toolbox was implemented to support animation and visualisation [12], but it is now superseded by ProB2-UI which can also open TLA+ files.

Overture [26] is an Eclipse-based IDE for the VDM formal method with a large feature set, particularly as far as simulation is concerned. The Maestro [36] INTO-CPS tool is an evolution targeting cyber-physical systems, containing unique features such as three-dimensional rendering of systems. PVSio-Web [31] for PVS is a tool particularly well-suited for domain-specific visualisations, e.g., for medical user interfaces. Many other formal specification languages and tools come with powerful user interfaces, such as Spin [16] or extensions thereof [33] for Promela, UPPAAL [2] and PRISM [24] for (probabilistic) timed automata, Alcoa [18] for Alloy [17], NuSMV [4] for SMV, or FDR4 [11] for CSP.

4 Conclusion

As presented above, ProB2-UI has some special features not available in its predecessor ProB Tcl/Tk. Still, there are some features of ProB Tcl/Tk that are not yet implemented in ProB2-UI, such as the LTSmin [19] integration [21]. Indeed, we have learned that one should not underestimate the time for a complete rewrite. Catching up with a still-evolving feature set of an existing tool is hard. Furthermore, it was not easy to achieve a performance as good as ProB Tcl/Tk. In initial versions of ProB2-UI, both memory usages and runtimes were considerably higher, and it took a while to identify and correct the bottlenecks. On the one hand, the Tcl interface of SICStus Prolog is relatively limited, only supporting simple values such as atoms and numbers, and lists thereof. This actually forced us to write an efficient UI API design from the start in ProB Tcl/Tk (e.g., not sending formulas back and forth between Tcl and Prolog). On the other hand, the Tcl interface of SICStus Prolog has less latency than the communication via sockets in ProB2. This meant that we had to group requests in ProB2 in order to improve the performance.

Two recent surveys [10,35] have investigated the formal methods tools used in the railway sector and have found the B-Method tools (in particular AtelierB and ProB) to be the most mature and widely used. We hope that ProB2-UI provides another step forward, in the form of a feature-rich and intuitive user

[1] See Sect. 5 of [9]: "Atelier-B and ProB are the right choice for top-down development of mainly monolithic systems, with complementary verification capabilities".

interface, which helps the modeller develop, verify, validate, debug, and understand formal models.

Our tool's homepage with download links and a video presentation is available at: https://prob.hhu.de/w/index.php/ProB2-UI.

Acknowledgements. We want to thank Christoph Heinzen who has elaborated the plugin mechanism in his Master's thesis. Many more persons were involved in the implementation of ProB2-UI, notably Dominik Hansen, Jessica Petrasch, Daniel Plagge, and Sebastian Stock. Thanks also to Olga Iudina for the Russian translations and anonymous referees for their useful corrections and suggestions. ProB2-UI is currently being extended within the DFG funded project IVOIRE.

References

1. Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. Int. J. Softw. Tools Technol. Transf. **12**(6), 447–466 (2010)
2. Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Yi, W.: UPPAAL — a tool suite for automatic verification of real-time systems. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) HS 1995. LNCS, vol. 1066, pp. 232–243. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0020949
3. Butler, M., et al.: The first twenty-five years of industrial use of the B-Method. In: ter Beek, M.H., Ničković, D. (eds.) FMICS 2020. LNCS, vol. 12327, pp. 189–209. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58298-2_8
4. Cimatti, A., et al.: NuSMV 2: an opensource tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45657-0_29
5. ClearSy, A.B.: User and Reference Manuals. Aix-en-Provence, France (2016). http://www.atelierb.eu/
6. Comptier, M., Leuschel, M., Mejia, L.-F., Perez, J.M., Mutz, M.: Property-based modelling and validation of a CBTC zone controller in Event-B. In: Collart-Dutilleul, S., Lecomte, T., Romanovsky, A. (eds.) RSSRail 2019. LNCS, vol. 11495, pp. 202–212. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-18744-6_13
7. Falampin, J., Le-Dang, H., Leuschel, M., Mokrani, M., Plagge, D.: Improving railway data validation with ProB. In: Romanovsky, A., Thomas, M. (eds.) Industrial Deployment of System Engineering Methods, pp. 27–43. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-33170-1_4
8. Ferrari, A., Mazzanti, F., Basile, D.: Systematic evaluation and usability analysis of formal tools for system design. CoRR, abs/2101.11303 (2021)
9. Ferrari, A., Mazzanti, F., Basile, D., ter Beek, M.H., Fantechi, A.: Comparing formal tools for system design: a judgment study. In: Rothermel, G., Bae, D. (eds.) ICSE 2020: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June–19 July, 2020, pp. 62–74. ACM (2020)
10. Ferrari, A., et al.: Survey on formal methods and tools in railways: the ASTRail approach. In: Collart-Dutilleul, S., Lecomte, T., Romanovsky, A. (eds.) RSSRail 2019. LNCS, vol. 11495, pp. 226–241. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-18744-6_15

11. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.W.: FDR3 — a modern refinement checker for CSP. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 187–201. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_13
12. Hansen, D., Bendisposto, J., Leuschel, M.: Integrating ProB into the TLA Toolbox. In: TLA Workshop (2014)
13. Hansen, D., et al.: Validation and real-life demonstration of ETCS hybrid level 3 principles using a formal B model. Int. J. Softw. Tools Technol. Transf. **22**(3), 315–332 (2020)
14. Hansen, D., et al.: Using a formal B model at runtime in a demonstration of the ETCS hybrid level 3 concept with real trains. Proceedings ABZ **2018**, 292–306 (2018)
15. Hansen, D., Schneider, D., Leuschel, M.: Using B and ProB for data validation projects. In: Butler, M., Schewe, K.-D., Mashkoor, A., Biro, M. (eds.) ABZ 2016. LNCS, vol. 9675, pp. 167–182. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33600-8_10
16. Holzmann, G.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley Professional, 1st edition (2011)
17. Jackson, D.: Alloy: a lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol. **11**, 256–290 (2002)
18. Jackson, D., Schechter, I., Shlyakhter., I.: Alcoa: the alloy constraint analyzer. In: Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium, pp. 730–733 (2000)
19. Kant, G., Laarman, A., Meijer, J., van de Pol, J., Blom, S., van Dijk, T.: LTSmin: high-performance language-independent model checking. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 692–707. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_61
20. Körner, P., Bendisposto, J., Dunkelau, J., Krings, S., Leuschel, M.: Embedding high-level formal specifications into applications. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) FM 2019. LNCS, vol. 11800, pp. 519–535. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30942-8_31
21. Körner, P., Leuschel, M., Meijer, J.: State-of-the-Art model checking for B and Event-B using ProB and LTSmin. In: Furia, C.A., Winter, K. (eds.) IFM 2018. LNCS, vol. 11023, pp. 275–295. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98938-9_16
22. Krings, S.: Towards infinite-state symbolic model checking for B and Event-B. Ph.D. thesis, Heinrich Heine Universität Düsseldorf, August 2017
23. Kuppe, M.A., Lamport, L., Ricketts, D.: The TLA$^+$ toolbox. Electron. Proc. Theoret. Comput. Sci. **310**, 50–62 (2019)
24. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: probabilistic symbolic model checker. In: Field, T., Harrison, P.G., Bradley, J., Harder, U. (eds.) TOOLS 2002. LNCS, vol. 2324, pp. 200–204. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46029-2_13
25. Ladenberger, L., Leuschel, M.: Mastering the visualization of larger state spaces with projection diagrams. In: Butler, M., Conchon, S., Zaïdi, F. (eds.) ICFEM 2015. LNCS, vol. 9407, pp. 153–169. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25423-4_10
26. Larsen, P., Battle, N., Ferreira, M., Fitzgerald, J., Lausdahl, K., Verhoef, M.: The overture initiative: integrating tools for VDM. ACM SIGSOFT Softw. Eng. Not. **35**, 1–6 (2010)

27. Lecomte, T., Burdy, L., Leuschel, M. :Formally checking large data sets in the railways. CoRR, abs/1210.6815. Proceedings of DS-Event-B 2012, Kyoto (2012)
28. Leuschel, M., Butler, M.: ProB: a model checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805. Springer, Heidelberg (2003). https://doi.org/10.1007/b13229
29. Leuschel, M., Mutz, M., Werth, M.: Modelling and validating an automotive system in classical B and Event-B. In: Raschke, A., Méry, D., Houdek, F. (eds.) ABZ 2020. LNCS, vol. 12071, pp. 335–350. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-48077-6_27
30. Métayer, C.: AnimB 0.1.1 (2010). http://wiki.event-b.org/index.php/AnimB
31. Oladimeji, P., Masci, P., Curzon, P., Thimbleby, H.: PVSio-web: a tool for rapid prototyping device user interfaces in PVS. In: Proceedings FMIS, vol. 69 (2013)
32. Plagge, D., Leuschel, M.: Seven at a stroke: LTL model checking for high-level specifications in B, Z, CSP, and more. Int. J. Softw. Tools Technol. Trans. **12**, 9–21 (2007)
33. Ruys, T.C.: Xspin/Project - integrated validation management for Xspin. In: Dams, D., Gerth, R., Leue, S., Massink, M. (eds.) SPIN 1999. LNCS, vol. 1680, pp. 108–119. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48234-2_8
34. Servat, T.: BRAMA: a new graphic animation tool for B models. In: Julliand, J., Kouchnarenko, O. (eds.) B 2007. LNCS, vol. 4355, pp. 274–276. Springer, Heidelberg (2006). https://doi.org/10.1007/11955757_28
35. ter Beek, M.H., et al.: adopting formal methods in an industrial setting: the railways case. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) FM 2019. LNCS, vol. 11800, pp. 762–772. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30942-8_46
36. Thule, C., Lausdahl, K., Gomes, C., Meisl, G., Larsen, P.G.: Maestro: the INTO-CPS co-simulation framework. Simul. Model. Pract. Theory **92**, 45–61 (2019)
37. Vu, F.: Simulation and verification of reactive systems in Lustre with ProB. Master's thesis, Heinrich Heine Universität Düsseldorf, June 2020
38. Vu, F., Leuschel, M., Mashkoor, A.: Validation of formal models by timed probabilistic simulation. In: Raschke, A., Méry, D. (eds.) ABZ 2021. LNCS, vol. 12709, pp. 81–96. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77543-8_6
39. Werth, M., Leuschel, M.: VisB: a lightweight tool to visualize formal models with SVG graphics. In: Raschke, A., Méry, D., Houdek, F. (eds.) ABZ 2020. LNCS, vol. 12071, pp. 260–265. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-48077-6_21
40. Winter, D.: Validating promela models with the ProB model chcker. Master's thesis, Institut für Informatik, Universität Düsseldorf (2008)
41. Yang, F., Jacquot, J., Souquières, J.: JeB: safe simulation of Event-B models in JavaScript. In: Proceedings APSEC, vol. 1, pp. 571–576. IEEE (2013)
42. Yu, Y., Manolios, P., Lamport, L.: Model checking TLA$^+$ specifications. In: Pierre, Laurence, Kropf, Thomas (eds.) CHARME 1999. LNCS, vol. 1703, pp. 54–66. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48153-2_6