Masoumeh Parsa | Marina Waldén | Colin Snook

An Overview of Formal Specification Languages and Tools Supporting Visualisation of System Development

Turku Centre *for* Computer Science

TUCS Technical Report
No 1127, December 2014

An Overview of Formal Specification Languages and Tools Supporting Visualisation of System Development

Masoumeh Parsa
Åbo Akademi University, Department of Computer Science
Joukahaisenkatu 3-5A, 20520 Turku, Finland
mparsa@abo.fi
Marina Waldén
Åbo Akademi University, Department of Computer Science
Joukahaisenkatu 3-5 A, 20520 Turku, Finland
mwalden@abo.fi
Colin Snook
Southampton University, Electronics and Computer Science
Southampton, United Kingdom, SO17 1BJ
cfs@ecs.soton.ac.uk

TUCS Technical Report
No 1127, December 2014

Abstract

A formal development approach is required for high-quality development of safety critical systems. In order to make the formal development process more feasible, tool support is needed. Furthermore, integrating visualization means to the process makes it easier to communicate the model and to observe its behavior at an early development stage. These efforts are, however, not always enough to make the formal development process feasible for industrial use. Therefore, to make the process smoother and more flexible for industrial applications, it also needs to be parallelized. In this report we give an overview of the state-of-the-art of formal specification languages that are used in industrial settings. Tools supporting these methods in combination with different visualization means to facilitate the formal development are also presented. Parallelizing the formal development process by dividing a model into multiple abstractions that capture different aspects of the model is suggested as an approach to make the process more flexible.

Keywords: Formal Methods, Tool Support, Visualisation, Abstractions

TUCS Laboratory
RITES – Resilient IT Infrastructures
Distributed Systems Laboratory
Integrated Design of Quality Systems group

The work has been done within the ADVICeS-project (No.: 266373) funded by the Academy of Finland.

1 Introduction

Different approaches have been proposed for verification from the primitive way of proving the system manually by pen and paper to a more comprehensive approach of using a framework for supporting the formal languages and proofs. The behavior of a software system cannot be captured by a unique mathematical theory. Similar to the programming languages that were designed for different purposes, various formal languages and tools were developed for different kinds of systems. Our vision in this survey is not to discover a perfect tool for formal methods, but to compare the features of existing tools with respect to different application areas and development from specification to implementation. By discovering tools for visualising the formal development process we aim at facilitating the formal development process. Since UML [12] is widely accepted in industry for designing systems, we study the previous attempts on the integration of UML and formal methods to get an overview of the strengths and weaknesses of the previous approaches. We will suggest a novel design approach as an improvement of the formal development. Formalizing the specification of the system allows a correct-by-construction design, while animating the specification lets the users observe the behavior of a system at an early development stage.

Different specification languages are available in terms of the characteristic of the system. We are mainly interested in formalisms that have been applied in industry. We focus mainly on model-based specifications, where the system can be described by a set of states and a finite number of state transitions that can be applied non-deterministically for each state. The B-Method [5] as well as VDM [10] and Z [36] are of our prime interest and are three specification languages based on the set theory. Alloy [25] can also be mentioned extending a declarative language similar to Z for providing a fully automatic analyzer. Modeling languages for real-time systems, such as UPPAAL [8, 9], are used for formalizing the systems that must fulfill one or more physical properties, e.g., time, temperature or altitude.

One approach of constructing the implementation from specification is refinement where a program is constructed stepwise, starting from an abstract model while each step preserves the correctness of the previous ones. The refinement approach can be seen as a software development process which simplifies the system specification and reduces the proof complexity of proof tools. If program A is refined by program B, then every behavior accepted by B should be accepted by A as well. The refinement in VDM, Z and B is based on the posit-and-prove approach that lets the user define his or her own refinement and then use proof tools to check the correctness of the refinement. Since the refined models are tightly dependent on the more abstract models, a mistake in one level can cause an inconsistency in the whole model. Hence, tools supporting the refinement process are vital for the system development. Moreover, studies have been performed to give guidance and to assist the refinement process [23].

In this paper we mainly focus on VDM, Z as well as B, and describe how they are integrated with design means to improve the software development. This survey is aimed at giving an overview of formal method techniques and point out how they could be more tightly connected to software development in industry.

2 Formal specification languages

In this section, we give an overview of the state-of-the-art specification languages that have been used in industry to some extent.

VDM (Vienna Development Method) [10] is a specification language for modeling real-time and concurrent systems. VDM-SL [35], the basic form of VDM, supports modeling of software systems by defining an abstract model and then deriving a more concrete model which is close to implementation including states and a set of operation which are defined as pre- and postconditions. VDM++ [17] is an extension on VDM-SL to support specification of object-oriented models and concurrent extension. As an example of industrial use of VDM, we can mention the secure smart cards. The structure of VDM is heavily keyword oriented.

Z [36] is a formal specification language based on Zermelo-Fraenkel set theory and the first order predicate logic. In contrast to the keyword orientation in VDM, Z uses boxes and schemas for a better demonstration of the specification. Common criticism against Z includes the syntax which is not close to the programming languages and hence not executable. Later, ZRC [14] was defined as a theory of refinement for Z to support well-defined formal techniques for developing programs.

Refinement from abstract level to implementation is a main key also in the B-Method [5][1]. The B-Method is a formal language for specifying software systems. One of the well-known industrial uses of the B-Method is the braking system for the Paris metro. The B-Method is one of the few formal methods that covers all stages of software development, from requirements (specification) and design (refinement) to implementation and code generation. The method uses abstract machines including states and operations. Moreover, it supports code generation to the languages C, Java and Ada. Event-B [4][2] was designed based on the B-Method to allow development of distributed systems. Operations were substituted by events that have no parameters. At most one event is executed non-deterministically at a time. The refinement in Event-B is more flexible compared to classical B, Z and VDM since adding new events, merging or splitting events are allowed. The B-Method and Z allow splitting a model into hierarchical sub-models. However, model decomposition was introduced only in Event-B to reduce the complexity of modeling big systems by dividing them into smaller sub-models.

Alloy [25] is a formal language inspired by Z for modeling and analyzing software abstractions. Alloy is simpler and more flexible than the above

mentioned specification languages. This is achieved by the cost of being less expressive. Alloy is strictly first order logic for accelerating the analyzability. It translates the model to a large boolean expression and analyzes the boolean expression automatically by a SAT solver. Alloy is mostly used for systems with complex structural states such as name servers, access control and cryptography.

Circus [43] is a formal language based on Z and CSP for stepwise development of concurrent and reactive systems. The language supports the behavioral aspects of modeling such as parallelism and choice which is difficult to define in Z, VDM-SL and B. CSP is based on process algebra and can be used to define the behavioral aspects of the system, while Z focuses on defining the data aspects.

3 Formal method tools

In this section we introduce the state-of-the-art formal tools for modeling and verification in the previously mentioned specification languages. In addition, various tools are available to assist the modeling steps by animation and execution. We focus more on the tools supporting B, due to their extensive use also in industry.

Atelier-B [19] was developed for designing and verifying models in the B-Method used in industrial B-Software projects. Systems can be refined from abstract machine to implementation. Atelier-B supports both a graphical and a command language interface. Theorem provers guarantee the correctness of the system by discharging the proof obligations. Also B-Toolkit [30] was introduced as a set of tools for software development in the B-Method. In B-Toolkit the file management and documentation together with the syntax and type checking are automated. BART [37] is an automatic refinement tool for B components and can be extended to support Event-B models as well. It is a stand alone tool but can also be launched in Atelier-B. BART can refine the abstract variables, operations and initialization of B components. Rules are used to determine the elements that need to be refined.

Rodin [3] is an Eclipse-based framework for developing Event-B specifications. It also verifies the Event-B models by generating and discharging the proof obligations and supports the refinement process of the models. Since Rodin is an eclipse-based platform, it is extensible by other plug-ins. Figure 1 shows the Rodin editor with two events from a model of an aircraft landing gear system [41]. The model is a case study that was proposed by Bionel and Wiels [11] and was modeled in the Rodin platform by Wen and Abrial (amongst others) [1].

Various plug-ins are available to provide a more flexible environment for Event-B development such as for example ProB and the model decomposition plug-in. ProB [28] includes both a standalone animator and an Eclipse plugin for animating and model checking of B and Event-B models. Invari-

[1] http://www.lab205.org/case-landing

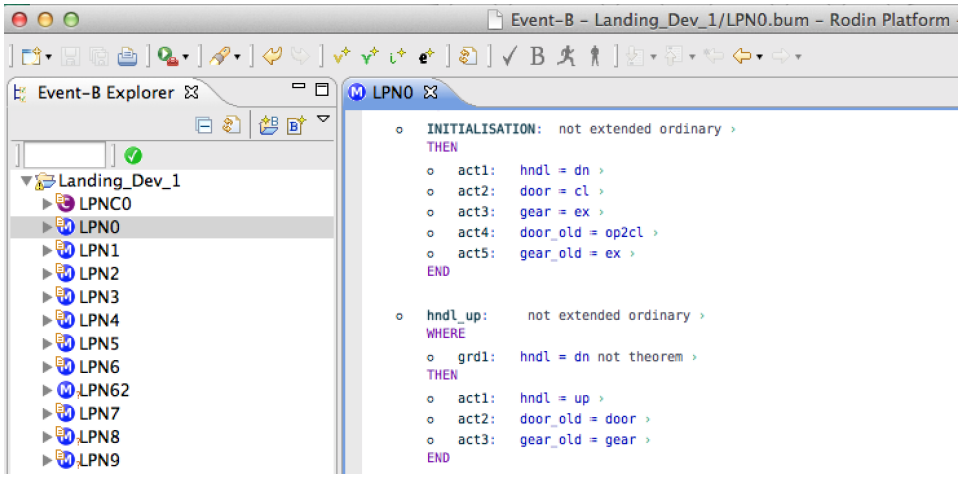

Figure 1: The Rodin editor

ant violations can be detected by finding a counterexample in the model. Moreover, ProB can be used in automated refinement checking. The refinement error detection in ProB discovers inconsistent behaviors and guard weakening. Figure 2 illustrates an example of an Event-B model [41] of the aircraft landing gear case study [11] in the ProB environment. A model decomposition plug-in is available for breaking a model into sub-models in Event-B which is classified into two approaches: shared variable decomposition and shared event decomposition. Depending on the preference and the system, the user can optimize the decomposition style.

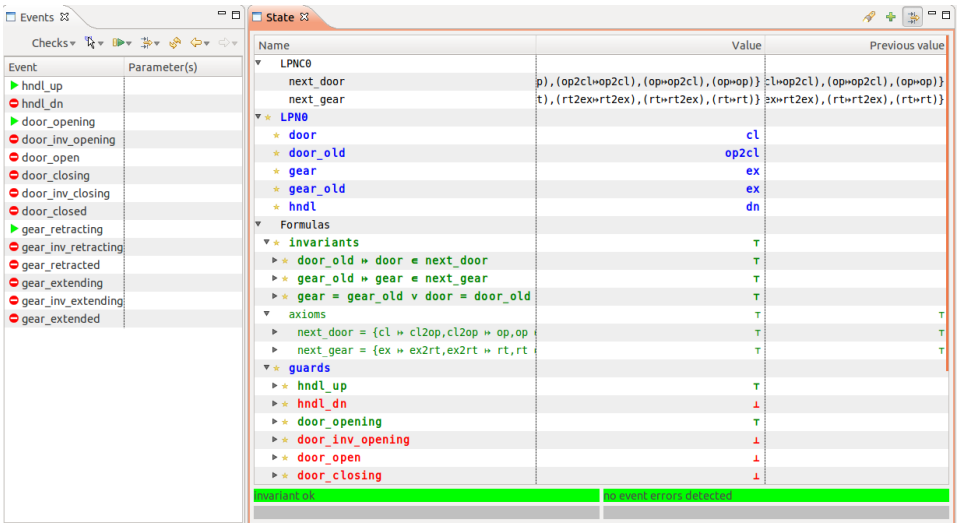

Figure 2: The state of invariants and variables in ProB

VDM-Tools [21] is a formal modeling tool for VDM specification language which supports modeling the software systems along with the code generation from VDM specification to Java and C++. Overture tool [33]

was designed afterwards with the same functionality, but based on Eclipse [32] (as the Rodin tool).

The community Z tool (CZT) [31] is a group of tools based on a java framework for supporting Z formal language with some support for Z extensions such as the Circus language. Both a stand alone version on the command line and a plug-in version for the Eclipse environment are available for the Z tool. The environment of CZT is suitable for specialists in Z, since it is not so intuitive. Jaza animator integrates with the Z tool CZT for validating Z models by animation and testing. Moreover, ZRC-Refine [22] is an interactive tool for supporting ZRC refinement.

Alloy Analyzer automatically analyses properties of Alloy models by bounding a model and converting it to a boolean logic formula. Then it uses a SAT solver to find a satisfied model. When Alloy Analyzer finds a false assertion, it generates a counter example. In this sense it is different from theorem provers. It visualizes the sample that is satisfied. Also Alloy Analyzer is available both as command line and as API versions to be used in other tools. Alloy Analyzer is not a verification tool and can be compared to ProB, but is more dominant.

CRefine [24] is a JAVA based framework that was developed to support the Circus language. The tool is more practical for educational purposes, but can also be appropriate for applications of industrial scale.

4 Integration of formal tools and visualization

In order to facilitate the formal development visualization means are needed in addition to the modelling and verification tools. One way of visualizing the design is to specify the requirements with for example UML [12] or other graphical languages. Another approach is animating the model to illustrate its behavior in a visual language.

4.1 Integrating the semi-formal and formal languages

UML was developed to represent graphical models of systems in software engineering. Various diagrams were defined to cover all aspects of a system such as package diagrams, class diagrams and state diagrams. Since UML has been used in specifying the requirement of the systems in many industrial projects, it has been considered as a front-end of various formal method languages. Other graphical interfaces were developed as well for other systems that will be mentioned at the end of this section.

UML-B [40] is a graphical editor for the B-Method and Event-B models including package diagrams, class diagrams and state charts. Class diagrams can be used to define data entities and their relationships, while packages represent the B components for covering a group of class diagrams. The UML-B model is then translatable to an Event-B model for verification. iUML-B [39] is a newer integrated version of UML-B with more editing features that make it possible to extend the model in both the UML diagram

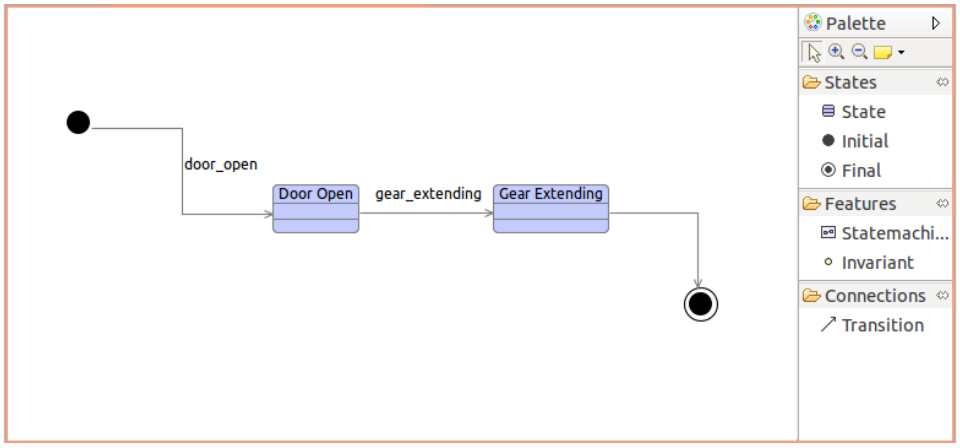

Figure 3: A state diagram in iUML-B

and Event-B synchronously. iUML-B includes both state-machines and class diagrams. The behavior of Event-B models can be visualized via UML-B state machines which also supports nested state machines that captures the refinement behavior. Figure 3 illustrates a part of the aircraft landing gear case study [11] in iUML-B. The `door_open` and `gear_extending` are events that are shown in the state machine as transitions.

VDMtools and Overture both support the transformation between UML class diagram and VDM++. While VDMtools is integrated with UML version 1.4, Overture is an Eclipse based plugin which links VDM++ with UML 2 class and sequence diagrams [27]. Also the sequence diagram can be used as a test sequence description that can be executed.

UML+Z [16] is a framework based on UML and Z for building, analyzing and refining models including state and class diagrams. The target group of this tool is developers with UML background knowledge, but not familiar with Z notations. However, a Z specialist is required to write Z operation specifications and invariants which cannot be expressed diagrammatically.

UML2Alloy [6] is a tool for transforming a subset of UML class diagrams and OCL constraints into the Alloy language by defining a mapping between the UML meta model and the Alloy meta model. It applies to different domains such as, for example, Agile manufacturing, security and access control. Alloy also integrates with UML to ensure the requirements of the system as a part of the software process.

UPPAAL is a model checker for realtime systems based on theory of timed automata including a graphical interface and a model checking engine. The tool guarantees that the system behaves in an intended way. UPPAAL uses state machine diagrams for validation. However the usage of state machines is quite different from Event-B state machines, since it is used for reachability, safety and liveness checks.

CODA (Co-Design Architecture) [13] is a modeling framework for embedded systems as an extension to Event-B and UML-B. Component be-

havior can be specified in CODA by assistance of UML-B and Event-B. CODA provides a diagrammatic notation for modelling the asynchronous communication between components using clock synchronised operations and timeouts. State-machines are used to constrain the enablement of component operations. There are two approaches for verifying the refinement in CODA: either by proving the correctness of the renement in each step, or by revealing the unseen problems that require re-modeling by model checking.

Simulink [15] is another graphical framework based on Matlab for modeling embedded systems and verifying the correctness of them. It also supports code generation from the model. Libraries of blocks are defined in Simulink that makes it easier to use as well as gives a possibility to define ones own libraries of blocks. There is a tool used to verify the correctness of Simulink diagrams by translating them to Z called ClawZ. Stateflow is an extension of Simulink to design statemachines and flowcharts for reactive systems.

4.2 Integrating formal methods and animation

Animation is integrated with formal tools to provide a better understanding of the system behavior and assuring the requirements. It can also be used to present a model to a customer, since understanding the model in formal languages can be difficult for those who were not involved in developing it. ProB was already described in Section 3.

BRAMA [38] is an Eclipse-based plug-in tool together with flash, which animates the graphical models of the system including guards, events and properties for both B- and Event-B models. BRAMA links a model specified in Rodin, Atelier B or B4free to the graphical representation and animation of the model using flash tools. Moreover, the user can interact with the models by various buttons. However, the animation should be created by a modeler. After creating the animation, the model can be exported and saved on a CD.

Anim B, an Eclipse-based plug-in, is also an animator and model checker for Event-B that allows the user to animate the whole model including the refinements. B Motion Studio [26] is developed for domain specific visualization.

4.3 Assisting the formal method verification with invariant discovery

An invariant is a property which is preserved at each point of a program. Invariants can be considered as one of the critical points of program verification, since it is needed for automatic theorem proving and testing. Different approaches have been proposed and various tools have been developed to automate the invariant discovery. Static invariant discovery focuses on the program texts and consider all possible executions. However, dynamic invariant detection is another techniques based on dynamic test execution traces. An example of a tool for dynamic invariant detection is Diakon [18]

which can detect invariants in Java, C and perl programs. HR [29] uses the invariant discovery approach to automate the discovery of properties in Event-B systems. HR generates invariants from the ProB animation traces. HRemo, an extension to HR, is an invariant discovery for Event-B that is especially used to find gluing invariants. Gluing invariants are the invariants that relate the states of the refined model to its abstraction.

Clousot [20] is another tool released by Microsoft for finding loop invariants automatically based on abstract interpretation, as well as pre- and postconditions extracted from the code. It is used for .Net languages (C#, F#, VB, etc) and can be installed on Visual Studio framework.

5 Parallelizing the refinement models by a layering approach

Visualisation of the refinement process makes it easier to communicate the system developement process. We have here chosen to focus on Event-B with visualisation help from UML, since it is a formalism that has good tool support. Moreover, it has been proved useful for industrial applications. However, in order to make the refinement process more smooth and flexible, it need to be parallelized, as well.

Applying superposition refinement using a layering approach for the development process is a perspective for parallelizing the formal development process of a system by dividing the requirements into different levels. Back proposed an idea of constructing software by adding each feature as a new layer on top of the abstract level while preserving the previous layers [7]. Developing systems in the B-Method using layers has also been investigated earlier by Waldén [42].

Even if the development process from specification to implementation in the different layers could be performed simultaneously, the layers are usually developed in a sequential manner due to dependencies between the layers. However, for a more flexible development in industrial settings a parallel approach would be more advantageous. Hence, we consider abstractions that can be developed in parallel [34]. Each abstraction can be considered as an aspect that represent a new feature of the system in the same way as a layer. To ease the validation of the model, we use UML state machines for the Event-B models. The requirements of the system are captured in UML statechart diagrams. The statechart diagrams can then be translated to Event-B machines. The system development may proceed in UML and Event-B in an integrated manner using iUML-B [39]. In our approach we propose multiple abstractions that can be refined in parallel. The abstractions are identified by capturing different properties of the system, e.g, safety temporal properties. These abstractions are then refined separately in a stepwise manner. To have a complete model, we merge the abstractions at some point of the development process. The merged version can be proved to be a refinement of each of the merged abstractions.

Yeganefard et al. [44] proposed a method for decomposing the requirements into sub-problems; their pattern for defining the sub-problems is, however, different from ours. Hoang et al. [41] also introduced composition of different abstractions. Their work inspired our contribution to incorporate multi-abstraction merging into a useable flexible development process.

As a case study [34], we have modeled part of the landing gear system case study which was proposed by Boniol and Weils [11]. For the case study we propose the abstractions in Figure 4; moving gear and door, analogical switch, general electro-valve, as well as anomalies. In each abstraction, the relationships to other abstractions are modeled via Boolean variables. The *moving gear* abstraction represents the extension and the retraction of the landing gear system, as well as opening and closing of the door. In the refinement steps the handle controlling the moving of the gear and the lights indicating the door states are introduced. The *analogical switch* abstraction models the hydraulic part controlling the system. The switch should be closed to enable the moving gear. Hence, these two abstractions merge naturally. The *general electro valve* controls the hydraulic pressure for opening and closing the doors, as well as extending and retracting the gear when the analogical switch is closed. More details on the pressure and its control are added in the refinement steps. All the failure modes concerning anomalies in the behaviour of the doors and the electro-valves are modeled in the *anomaly* abstraction. Figure 4 gives an overview of the abstractions of the landing gear case study, where each abstraction can be seen as a parallel layer representing an aspect of the system. The nature of these abstractions are such that they can be developed in parallel independently of each other in order to make the system development more flexible.

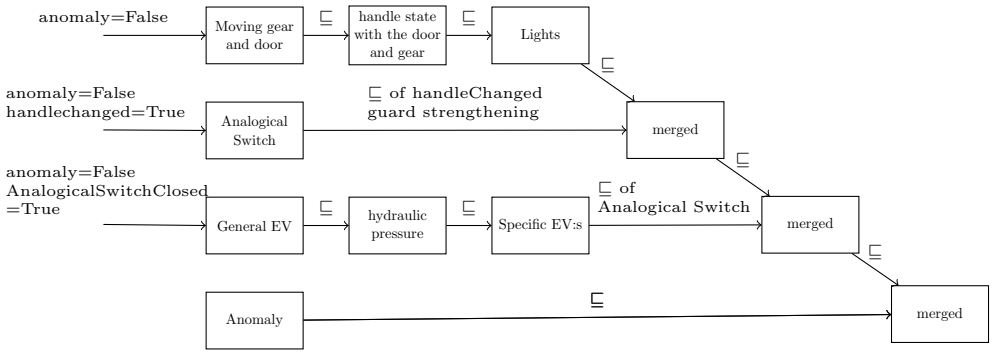

Figure 4: Proposed abstractions of the landing gear system

6 Conclusion

In this paper, we have presented an overview of the state-of-the-art formal specification languages that are used in industrial settings, as well as available tools for supporting these languages. Moreover, we presented various approaches of integrating the semi-formal languages such as UML to the

formal languages for representing the requirements. Animating the model is a complementary approach to ease the use of formal methods by observing the behavior of the model during the execution. Since Event-B specification language is mainly used for the development of safety critical systems and provides a good tool support for the development of a model, it was our primary choice for the case study. By dividing a model into multiple abstractions we can decrease the complexity of the model, and parallelize the development process. At least one property is captured in each abstraction. In this way the development process becomes more flexible.

In our future work, we would like to extend our model to improve the merging process and to cover more refinement levels to better discover advantages and drawbacks of our approach. Defining new transformation rules and incorporating the existing rules into the tool support for merging the abstractions is another challenge in our future studies.

References

[1] Jean-Raymond Abrial. The B Book - Assigning Programs to Meanings, 1997.

[2] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[3] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *Software Tools for Technology Transfer*, 12(6):447–466, November 2010.

[4] Jean-Raymond Abrial and Thai Son Hoang. Using design patterns in formal methods: An Event-B approach. In *ICTAC*, pages 1–2, 2008.

[5] Jean-Raymond Abrial, Matthew K. O. Lee, David Neilson, P. N. Scharbach, and Ib Holm Sørensen. The B-Method. In *VDM Europe (2)*, pages 398–405, 1991.

[6] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. Uml2alloy: A challenging model transformation. In *In: ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems*, pages 436–450. Springer, 2007.

[7] Ralph-Johan Back. Software construction by stepwise feature introduction. In Didier Bert, Jonathan P. Bowen, Martin C. Henson, and Ken Robinson, editors, *ZB*, volume 2272 of *Lecture Notes in Computer Science*, pages 162–183. Springer, 2002.

[8] Gerd Behrmann, Re David, and Kim G. Larsen. A tutorial on UPPAAL. In *SFM-RT 2004*, pages 200–236. Springer, 2004.

[9] Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL - A Tool Suite for Automatic Verification of Real-Time Systems. In *Hybrid Systems*, pages 232–243, 1995.

[10] Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, London, UK, 1978. Springer-Verlag.

[11] Frederic Boniol and Virginie Wiels. The landing gear system case study. In *ABZ 2014: The Landing Gear Case Study*, pages 1–18. Springer, 2014.

[12] G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language - a Reference Manual*. Addison-Wesley, 1998.

[13] Michael Butler, John Colley, Andrew Edmunds, Colin F. Snook, Neil Evans, Neil Grant, and Helen Marshall. Modelling and refinement in CODA. In John Derrick, Eerke A. Boiten, and Steve Reeves, editors, *Refine*, volume 115 of *EPTCS*, pages 36–51, 2013.

[14] Ana Cavalcanti and Jim Woodcock. ZRC - A Refinement Calculus for Z. *Formal Aspects of Computing*, 10(3):267–289, 1998.

[15] Chunqing Chen, Jin Song Dong, and Jun Sun. A formal framework for modeling and validating Simulink diagrams. *Formal Aspects of Computing*, 21(5):451–483, 2009.

[16] S. Dupuy, Y. Ledru, and M. Chabre-Peccoud. An Overview of RoZ: A Tool for Integrating UML and Z Specifications. In B. Wangler and L. Bergman, editors, *CaiSE 2000*, Lecture Notes in Computer Science, pages 417–430. Springer-Verlag, 2000.

[17] Eugène Dürr and Jan van Katwijk. VDM++, A Formal Specification Language for OO Designs. In Georg Heeg, Boris Magnusson, and Bertrand Meyer, editors, *TOOLS (7)*, pages 63–77. Prentice Hall, 1992.

[18] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1–3):35–45, December 2007.

[19] STERIA (F). Atelier B, Manuel Utilisateur, 1998.

[20] Manuel Fähndrich and Francesco Logozzo. Static contract checking with abstract interpretation. In *Proceedings of the 2010 International Conference on Formal Verification of Object-oriented Software (FoVeOOS'10)*, pages 10–30, Berlin, Heidelberg, 2011. Springer-Verlag.

[21] John Fitzgerald, Peter Gorm Larsen, and Shin Sahara. VDMTools: Advances in Support for Formal Modeling in VDM. *SIGPLAN Not.*, 43(2):3–11, February 2008.

[22] Angela Freitas, Carla Nascimento, and Ana Cavalcanti. A Refinement Tool for Z. In Jin Song Dong and Jim Woodcock, editors, *ICFEM*, volume 2885 of *Lecture Notes in Computer Science*, pages 396–415. Springer, 2003.

[23] Gudmund Grov, Andrew Ireland, and Maria Teresa Llano. Refinement plans for informed formal design. In John Derrick, John Fitzgerald, Stefania Gnesi, Sarfraz Khurshid, Michael Leuschel, Steve Reeves, and Elvinia Riccobene, editors, *ABZ 2012*, volume 7316 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2012.

[24] A. C. Gurgel, C. G. de Castro, and M. V. M. Oliveira. Tool Support for the Circus Refinement Calculus. In *ABZ 2008*, volume **5238** of *Lecture Notes in Computer Science*, page 349. Elsevier B. V., 2008.

[25] Daniel Jackson. Alloy: A lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology*, 11(2):256–290, April 2002.

[26] Lukas Ladenberger, Jens Bendisposto, and Michael Leuschel. Visualising Event-B Models with B-Motion Studio. In María Alpuente, Byron Cook, and Christophe Joubert, editors, *FMICS 2009*, volume 5825 of *Lecture Notes in Computer Science*, pages 202–204. Springer, 2009.

[27] Kenneth Lausdahl, Hans Kristian Agerlund Lintrup, and Peter Gorm Larsen. Connecting UML and VDM++ with Open Tool Support. In Ana Cavalcanti and Dennis Dams, editors, *FM 2009*, volume 5850 of *Lecture Notes in Computer Science*, pages 563–578. Springer, 2009.

[28] Michael Leuschel and Michael Butler. ProB: A Model Checker for B. In *FME 2003*, volume 2805 of *Lecture Notes in Computer Science*, pages 855–874. 2003.

[29] Maria T. Llano, Andrew Ireland, and Alison Pease. Discovery of Invariants through Autimated Theory Formation. *Formal Aspects of Computing*, (26):203 – 249, 2014.

[30] B-Core(UK) Ltd. B-Toolkit user manual, 1996.

[31] Petra Malik and Mark Utting. CZT: A Framework for Z Tools. In *ZB 2005*, Lecture Notes in Computer Science, pages 65–84. Springer, 2005.

[32] Bill Moore, David Dean, Anna Gerber, Gunnar Wagenknecht, and Philippe Vanderheyden. Eclipse development using the graphical editing framework and the eclipse modeling framework. Technical report, IBM, January 2004.

[33] J. P. Nielsen and J. K. Hansen. Development of an overture/VDM++ tool set for Eclipse. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Lyngby (DK), 2005.

[34] Masoumeh Parsa, Colin Snook, Marta Olszewska, and Marina Walden. Parallel Development of Event-B Systems with Agile Methods. In *Proc. of NWPT'2014*, Halmstad, Sweden, October 2014.

[35] Nico Plat and Peter Gorm Larsen. An Overview of the ISO/VDM-SL Standard, August 1992.

[36] Ben Potter, David Till, and Jane Sinclair. *An Introduction to Formal Specification and Z*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1996.

[37] Antoine Requet. BART: A tool for automatic refinement. In Egon Borger, Michael J. Butler, Jonathan P. Bowen, and Paul Boca, editors, *ABZ 2008*, volume 5238 of *Lecture Notes in Computer Science*, page 345. Springer, 2008.

[38] Thierry Servat. BRAMA: A New Graphic Animation Tool for B Models. In *B 2007*, number 4355 in Lecture Notes in Computer Science, pages 274–276. Springer-Verlag, 2006.

[39] Colin Snook. iUML-B. http://wiki.event-b.org/index.php/IUML-B, 2013.

[40] Colin Snook and Michael Butler. UML-B: A Plug-in for the Event-B Tool Set. In *Proceedings of the 1st International Conference on Abstract State Machines, B and Z (ABZ 2008)*, pages 344–344. Springer-Verlag, 2008.

[41] Wen Su and Jean-Raymond Abrial. Aircraft landing gear system: Approaches with Event-B to the modeling of an industrial system. In *ABZ 2014 Case Study Track (CCIS 433)*, pages 19–35. Springer International Publishing, 2014.

[42] M. Walden. Layering Distributed Algorithms within the B-Method. In *In (Bert, D.): Second International B Conference (B'98)*, volume 1393 of *Lecture Notes in Computer Science*, pages 243–260. Springer-Verlag, 1998.

[43] Jim Woodcock and Ana Cavalcanti. The semantics of Circus. In Didier Bert, Jonathan P. Bowen, Martin C. Henson, and Ken Robinson, editors, *ZB*, volume 2272 of *Lecture Notes in Computer Science*, pages 184–203. Springer, 2002.

[44] Sanaz Yeganefard and Michael Butler. Problem decomposition and sub-model reconciliation of control systems in Event-B. In *IEEE 14th International Conference on Information Reuse and Integration (IRI 2013)*, pages 528–535. IEEE, August 2013.

Turku
Centre *for*
Computer
Science

Joukahaisenkatu 3-5 A, 20520 TURKU, Finland | www.tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics

Turku School of Economics
- Institute of Information Systems Sciences

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

ISBN 978-952-12-3160-5
ISSN 1239-1891