The Rodin Formal Modelling Tool[1]

Michael Butler and Stefan Hallerstede
University of Southampton
United Kingdom
{*M.J.Butler, sth*}*@ecs.soton.ac.uk*

Abstract

We present a software tool, the Rodin tool, for formal modelling in Event-B. Event-B is a notation and method developed from the B-Method and is intended to be used with an incremental style of modelling. The idea of incremental modelling has been taken from programming: modern programming languages come with integrated development environments that make it easy to modify and improve programs. The Rodin tool provides such an environment for Event-B.
The two main characteristics of the Rodin tool are its ease of use and its extensibility. The tool focuses on modelling. It is easy to modify models and try out variations of a model. The tool can also be extended easily. This will make it possible to adapt the tool specific needs. So the tool can be adapted to fit into existing development processes instead demanding the opposite. We believe that these two characteristics are major points for industrial uptake.

Keywords: Event-B, formal modelling, modelling tool

1. INTRODUCTION

We consider modelling of software systems and more generally of complex systems to be an important development phase. We also believe that more complex models can only be written when the method of stepwise refinement [9] is used. Formal notation is indispensable in such a modelling activity. It provides the foundation on which building models can be carried out. Simply writing a formal text is insufficient, though, to achieve a model of high quality. The only serious way to analyse a model is to reason about it, proving in a mathematically rigorous way that all required properties are satisfied.

This short paper is organised as follows. In Section 2 we briefly describe Event-B. In Section 3 the Rodin tool for Event-B is presented. It is heavily inspired by modern programming tools. In Section 4 the need for extensibility and configurability is discussed and some extensions of the Rodin tool are presented. Section 5 gives an indication of the industrial relevance of Event-B and the Rodin tool.

2. THE EVENT-B METHOD AND NOTATION

Event-B [7] is a formalism and method for discrete systems modelling. It has been developed from the B-Method [1] using many ideas of Action Systems [8]. The semantics of an Event-B model is characterised by *proof obligations*. In fact, proof obligations have a two-fold purpose. On the one hand, they show that a model is sound with respect to some behavioural semantics. On the other hand, they serve to verify properties of the model. This goes so far that we only focus on the proof obligations and do not present a behavioural semantics at all. This approach permits us to use the same proof obligations for very different modelling domains, e.g., reactive, distributed and concurrent systems [6], sequential programs [3], electronic circuits [16], or mixed designs [2], not being constrained to semantics tailored to a particular domain. Event-B is a calculus for modelling that is independent of the various models of computation.

[1]This research was carried out as part of the EU research project IST 511599 RODIN (Rigorous Open Development Environment for Complex Systems) http://rodin.cs.ncl.ac.uk.

3. BORROWING IDEAS FROM PROGRAMMING TOOLS

In order for formal modelling to be used safely and effectively in engineering practice, good tool support is necessary. Present day integrated development environments used for programming do carry out many tasks automatically in the background, e.g. Eclipse [13, 15], and provide fast feedback when changes are made to a program text. In particular, there is no need for the user to start processes like compilation. A program is written and then run or debugged without compiling it. In the Rodin project (IST 511599 RODIN) a tool for Event-B [7], called *Rodin* [4], has been developed that applies these techniques used in programming to formal modelling. The Rodin tool is implement on top of the Eclipse platform [13]. Instead of compilation, we are interested in proof obligation generation and automatically discharging trivial proof obligations. Instead of running a program we reason about models or analyse them. A dedicated web page

http://www.event-b.org/

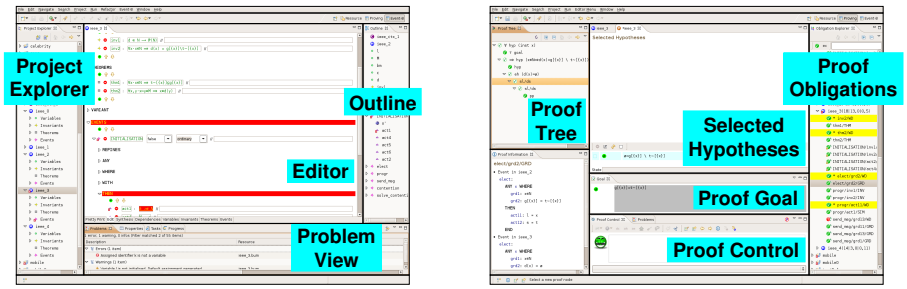
provides information on Event-B and the Rodin tool and download instructions for the tool.

Verification by proof is not restricted to modelling. It has a long tradition in programming methodology, too, e.g. [17]. Software tools that support formal verification methods in programming have been developed, e.g. [11]. We mention [11], in particular, because the Boogie architecture presented in that article provides characteristics similar to the Event-B tool. We quote two points from [11] about Boogie and present our view of them:

(1) “Design-Time Feedback”. The tool is very responsive and provides almost immediate feedback that easily relates to the program, (resp. model).

(2) “Distinct Proof Obligation Generation and Verification phases”. This allows decoupling the development of the programming (resp. modelling) method and prover technologies. It also allows the origin of a proof obligation to be traced easily. This is particularly important when proofs fail.

The third point in the list describing Boogie in [11] is “Abstract Interpretation and Verification Condition Generation”. The corresponding problem does not exist in the Event-B notation because it has been designed to be very close to the proof obligations by means of which we reason about Event-B. Technical difficulties encountered in Event-B stem more from the support of refinement and from the requirement that proof obligations appear transparent to the user. By transparency we mean that the user should look at the proof obligation as being part of the model. When a proof obligation cannot be proved, it should be almost obvious what needs to be changed in the model. When modelling, we usually do not simply represent some system in a formal notation but we learn about the system. This means we need support for the incremental approach taken to programming; the tool must be responsive to changes made to models. In programming this is essential during debugging and unit testing. When modelling we use formal proof instead of debugging and testing but need the same fast feedback. Figure 1 shows two default perspectives



(a) Modelling Perspective (b) Proving Perspective

FIGURE 1: Default Perspectives of Rodin

of the Rodin tool: 1a the default modelling perspective, 1b the proving perspective. Perspectives

are an Eclipse technique to create and store configurations of the user interface. The user can create own configurations that may better serve his needs.

4. EXTENSIBILITY AND CONFIGURABILITY OF RODIN

We take the view that no one tool can solve all development needs and that it is important to apply a range of tools in a complementary way in rigorous development. For example, it makes sense to apply model checking as a pre-filter, before applying a theorem prover to a proof obligation. Similarly the use of diagrammatic views (e.g., UML) of a formal model can aid with construction and validation. Many analysis tools, such as model checkers, theorem provers, translation tools (e.g., UML to B and code generators), have been developed, some of which are commercial products and some research tools. However a major drawback of these tools is that they tend to be closed and difficult to use together in an integrated way. They also tend to be difficult to extend for other interested parties and hinder collaboration. The Rodin tool greatly extends the state of the art in formal methods tools, allowing multiple parties to integrate their tools as plug-ins to support rigorous development methods. This is likely to have a significant impact on future research in formal methods tools and will encourage greater industrial uptake of these tools. Table 1 shows some plug-ins for the Rodin tool that are presently available.

Name	Description
B4free provers	**Provider:** ClearSy **Function:** Theorem provers **Web:** http://www.b4free.com/index.html
Brama	**Provider:** ClearSy **Function:** Animation of B models. The purpose is twofold: (1) experimentation with a model to observe states and transitions (2) Flash animation of Event-B models **Web:** http://www.brama.fr/index_en.html
UML-B	**Provider:** University of Southampton **Function:** UML-like graphical front-end for Event-B supporting class diagrams and state charts **Web:** http://users.ecs.soton.ac.uk/cfs/umlb.html
ProB	**Provider:** University of Düsseldorf **Function:** Animation and Model-checking of Event-B models; Counter-examples for false proof goals, in particular, proof obligations **Web:** http://www.stups.uni-duesseldorf.de/ProB/overview.php

TABLE 1: Some Available Plug-ins for Rodin

As well as supporting the combination of different complementary tools, openness and customisability is very important in that it will allow users to customise and adapt the basic tools to their particular needs. For example, a car manufacturer using Event-B to study the overall design of a car information system might be willing to plug some special tools able to help defining the corresponding documentation and maintenance package. Likewise, a rocket manufacturer using Event-B might be willing to plug a special tool for analysing and developing the failure detection part of its design.

5. INDUSTRIAL APPLICATIONS AND CASE STUDIES

The B-Method has a long history of successful industrial applications [10, 14]. Tools, such as, Atelier B [12], Click'n'Prove [5], and the B-Toolkit [18] are available and in use. The Rodin tool is still at an early stage but we have some experience with its use from case studies carried out in the Rodin project and porting models produced with previous tools (mostly Click'n'Prove) to Rodin.

The Rodin project included five industrial case studies that served to validate the tool set and helped with the elaboration of an appropriate methodology for using the tools. The case studies were lead by industrial partners of the Rodin project supported by the other partners. The case studies were as follows:

- a failure management system for an engine controller
- part of a platform for mobile Internet technology
- engineering of communications protocols
- an air-traffic display system
- an ambient campus application

Further details can be found in [20]. The air-traffic display system case study is also described in [19].

Some previous developments, e.g., [10] require special support for software development: means to express software modules and tools to refine certain modules automatically. Corresponding plug-ins have not yet been developed. There is no theoretical or practical reason why this could not be done. However, it will still require some time until a good set of stable plug-ins is available for this purpose. The core of the Rodin tool has intentionally not been geared towards a particular application in order to make very different uses possible, not just software development.

6. CONCLUSION

We believe that modelling will remain difficult. This does not mean, however, that it is impossible to develop a productive modelling tool. Programming is difficult, too. Still we have very efficient programming tools. But we also have many people who simply got used to the difficulties of programming. Hopefully, they will also get used to the difficulties of modelling when appropriate tools are available.

The Rodin tool provides a seamless integration between modelling and proving. This is important for the user to focus on the modelling task and not on switching between different tools. The purpose of modelling is not just to write a specification. It also serves to improve our understanding of the system being modelled. The Event-B tool tries to reflect this view by providing a lot of help for exploring a model and reasoning about it.

The tool is extensible and configurable because we cannot predict future uses of Event-B. The architecture has been designed to make this as easy as possible to invite anyone who needs a (formal) modelling tool to tailor it to his needs. We hope this will make it possible to employ the tool in very different development processes.

ACKNOWLEDGEMENTS

We would like to thank all members of the Rodin project who have contributed to the Rodin tool, especially, Jean-Raymond Abrial, Thai Son Hoang, Cliff Jones, Thierry Lecomte, Michael Leuschel, Farhad Mehta, Christophe Métayer, Colin Snook, François Terrier, and Laurent Voisin.

REFERENCES

[1] Jean-Raymond Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.
[2] Jean-Raymond Abrial. Event driven system construction, 1999.
[3] Jean-Raymond Abrial. Event based sequential program development: Application to constructing a pointer program. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, volume 2805 of *LNCS*, pages 51–74. Springer, 2003.

[4] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, and Laurent Voisin. An open extensible tool environment for Event-B. In Z. Liu and J. He, editors, *ICFEM 2006*, volume 4260, pages 588–605. Springer, 2006.
[5] Jean-Raymond Abrial and Dominique Cansell. Click'n'Prove: Interactive Proofs within Set Theory. In *Theorem Proving in Higher Order Logics*, volume 2758 of *LNCS*, pages 1–24, 2003.
[6] Jean-Raymond Abrial, Dominique Cansell, and Dominique Méry. A mechanically proved and incremental development of IEEE 1394 tree identify protocol. *Formal Aspects of Computing*, 14(3):215–227, 2003.
[7] Jean-Raymond Abrial and Stefan Hallerstede. Refinement, Decomposition and Instantiation of Discrete Models: Application to Event-B. *Fundamentae Informatica*, 77(1-2), 2007.
[8] Ralph-Johan Back. Refinement Calculus II: Parallel and Reactive Programs. In J. W. deBakker, W. P. deRoever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, volume 430 of *Lecture Notes in Computer Science*, pages 67–93, Mook, The Netherlands, May 1989. Springer-Verlag.
[9] Ralph-Johan Back and Joakim von Wright. *Refinement Calculus: A Systematic Introduction*. Graduate Texts in Computer Science. Springer-Verlag, 1998.
[10] Frédéric Badeau and Arnaud Amelot. Using B as a high level programming language in an industrial project: Roissy VAL. In Helen Treharne, Steve King, Martin Henson, and Steve Schneider, editors, *ZB 2005*, volume 3455 of *LNCS*, pages 334–354, 2005.
[11] Mike Barnett, Bor-Yuh Chang, Robert DeLine, Bart Jacobs, and K. Rustan M. Leino. Boogie: A Modular Reusable Verifier for Object-Oriented Programs. In *FMCO 2005*, volume LNCS. Springer-Verlag, 2005. to appear.
[12] Clearsy. Atelier B tool homepage. http://www.atelierb.societe.com/.
[13] Eclipse. Eclipse platform homepage. http://www.eclipse.org/.
[14] Didier Essamé and Daniel Dollé. B in large-scale projects: The canarsie line CBTC experience. In Jacques Julliand and Olga Kouchnarenko, editors, *B*, volume 4355 of *Lecture Notes in Computer Science*, pages 252–254. Springer, 2007.
[15] Erich Gamma and Kent Beck. *Contributing to Eclipse*. Addison Wesley, 2003.
[16] Stefan Hallerstede. Parallel hardware design in B. In Didier Bert, Jonathan P. Bowen, Steve King, and Marina A. Waldén, editors, *ZB*, volume 2651 of *LNCS*, pages 101–102. Springer, 2003.
[17] James C. King. A new approach to program testing. In *Proceedings of the international conference on Reliable software*, pages 228–233, New York, NY, USA, 1975. ACM Press.
[18] B-Core(UK) Ltd. B-Toolkit homepage. http://www.b-core.com/btoolkit.html.
[19] Abdolbaghi Rezazadeh, Neil Evans, and Michael Butler. Redevelopment of an Industrial Case Study Using Event-B and Rodin. In *BCS-FACS Christmas 2007 Meeting*, 2007.
[20] RODIN. Deliverable D18: Intermediate report on case study developments. http://rodin.cs.ncl.ac.uk/D18.pdf.