Book reviews

Modeling in Event-B – System and Software Engineering Jean-Raymond Abrial Cambridge University Press, May 2010 ISBN-10: 0521895561 doi:10.1017/S0956796812000081

The Event-B formal notation is a trimmed down descendant of B which has gained significant acclaim, particularly with the impressive tool support and industrial experience generated through EU projects Rodin (www.event-b.org, 2004–2007) and Deploy (www.deploy-project.eu, 2008–2012). Expectations were understandably high for this, the “Event-B Bible”, written by Jean-Raymond Abrial, inventor of Z and B.

As it turns out, this is a rich and accessible book, demonstrating both the strengths and weaknesses of the use of Event-B, and containing varied and valuable case studies as its core. It is written in a pleasant colloquial style, with changes in the vocal tempo and tone leaping off the pages.

An engaging introduction to the ideas of Event-B, and to formal development in general, is given in Chapter 2 which describes the case study “Controlling cars on a bridge”. Refinement as the gradual introduction of detail and determinacy is presented in a gentle way. Invariants and preconditions are strengthened incrementally in a convincing way as a consequence of intuitively sensible proof obligations. A strong case for doing proof alongside specification is made implicitly. The weaker side of the method in general shows through here already, though – more details on that below.

An overly sensitive cover-to-cover reader might not have made it as far as Chapter 2, however. The Prologue and to a lesser extent Chapter 1 are likely to turn readers off, through their many broad-brush statements: software engineering is in the first place mathematical; post-hoc verification is imposssible by definition; testing “always postpones any serious thinking”. Respected techniques such as model checking and abstract interpretation are also dismissed on the basis of shallow arguments. Formal methods, on the other hand, are often characterised by what they are *not* rather than by what they are.

I would also have liked to see a more substantive argument on the sufficiency of set-theoretic notation (naive set theory in the book; typed in the Rodin tools), rather than the terse “This is an error” and “This is nonsense” judgements for the alternatives.

My main methodological worry about this book (and to some extent Event-B in general) arises already from the prologue, in particular the following comment (page xiv): “From the point of view of modelling, it is important to understand that there are no fundamental differences between a human pressing a button, a motor starting or stopping, or a piece of software executing certain tasks.” If modelling stands by itself, that may be true; but where it is followed by refinement, like here, I have to disagree: some events model uncontrolled aspects of the system’s environment; some are part of the interface of the system to be developed, available to human users; and some are internal events that contribute to effecting the desired outcomes of the system without having been explicitly requested by an outside user. In my view, events with a different status like that should play a different role in refinement: external events, for example, should not be refined; internal events, on the other hand, can be refined very liberally as long as this does not impact on changes of observable behaviour. By not allowing for fundamental differences between events, Event-B ends up being too generous in allowing refinement of external events, and too strict on truly internal ones. For a further exploration and explanation of these and related issues, see two papers at the 2011 Refinement Workshop [1,3].

The informal motivations for imposing certain proof obligations – or not – are not always strong. From a formalist point of view, this could have been addressed by providing an exact definition of refinement, in particular what it establishes semantically, rather than a collection of proof obligations, with some of these considered optional. Rightly, however, refinement theorists are not taken as the book's primary audience. For everyone else, though, the informal explanations leave a lot to be desired.

Consider for example the motivation for non-divergence of new events (page 64): that this would stop the concrete instances of existing events from being postponed indefinitely. However, the refinement that follows this explanation ensures exactly that undesirable effect, not through divergence, but by making the guard of the new operation too strong. In detail: a traffic light is introduced, with new operations for the lights changing. These are potentially divergent, so light changes are constrained to only happening after a car has passed - the car passing being one of the "existing" events. This allows the scenario where one car enters, the light turns red, and then no car is allowed to enter any more until the light has turned green, which in turn will not happen until a car has exited again. This probably could have been prevented, either by using "anticipated events", or explicit requirements like property 16.4.5 on page 492 – but the main problem is that possible starvation of a particular event is not even noticed.

Chapters 3–4, 6–13 and 16–17 present more case studies. These form an interesting collection, ranging across a variety of application areas, and covering many relevant points of modelling and refinement. However, they are notably weaker in their motivation than Chapter 2, in particular at the methodological level. Most refinement steps are given a number but not a name, and as the book continues there is progressively less top-level information about the concerns addressed in individual refinements, nor about the rationales for the separation and sequencing of the concerns over these steps. Too often it is "now we do this" and "then we do that" without any further explanation. Several later chapters leave the final refinement steps as an exercise to the reader, without much guidance as to what these steps should achieve. However, presentations and Rodin scripts on the book's website http://www.event-b.org/abook.html help a bit in this respect.

Chapters 5, 14 and 15 are of a different nature. Chapter 5 gives a convincing overview of the Event-B notations and proof obligations. Chapter 14 presents the mathematical justification of the proof obligation rules used. This includes a fairly standard explanation of the one-to-one replacement of abstract operations by concrete ones, from refinement of traces. However, the justification for introducing new operations lacks in detail; a recent paper by Schneider, Treharne and Wehrheim [3] goes some way towards explaining this, including the Event-B concepts of "anticipated" and "convergent" labels on events, which get no mention at all in this chapter. Generally in this book bibliographic data is given chapter by chapter, and is of highly variable quality. For example, often publication information is lacking. Chapter 14 in particular contains no literature citations nor a bibliography, which makes it unnecessarily difficult to appreciate the Event-B refinement theory in the context of standard results.

Chapter 15 presents the application of Event-B to the development of sequential programs. Basic Event-B has a simpler structure than B, losing all program control structure; in this chapter, this structure is reintroduced in events obtained by "merging" simpler events using systematic rules. This alternative approach to "program refinement" does not appear to be fully crystalised yet; for a further exploration and critique of this method, see a recent paper by Hallerstede [2].

The final Chapter 18 lists some problems for readers to try out in Event-B and Rodin, including a substantive mathematical theorem.

The book is probably the closest to a reference manual for Event-B that currently exists, but its index could have been substantially extended. For example, "merging rules" is not included, nor is "feasibility" or its associated proof obligation "FIS".

Overall, this book contains a lot of valuable material, and due to the simplicity of the Event-B notation it should be a seriously considered for introductory courses on formal modelling with associated proof. As a basis for advanced study and research, it would be significantly better if it contained more conceptual clarity and methodological guidance on refinement, as well as a much more extensive index and bibliographic information. The case studies, however, still provide a rich source of example material for researchers.

References

Eerke Boiten (2011) *Perspicuity and granularity in refinement*, Refinement Workshop, EPTCS **55**, pp 155–165. DOI: 10.4204/EPTCS.55.10

Stefan Hallerstede (2009) *Proving quicksort correct in Event-B*, Refinement Workshop, ENTCS **259**, pp 47–65, DOI: 10.1016/j.entcs.2009.12.017

Steve Schneider, Helen Treharne & Heike Wehrheim (2011) *A CSP account of Event-B refinement*, Refinement Workshop, EPTCS **55**, pp 139–154. DOI: 10.4204/EPTCS.55.9

EERKE BOITEN
School of Computing, University of Kent, Canterbury, UK

Drawing Programs:
The Theory and Practice of Schematic Functional Programming, by Tom Addis and Jan Addis Springer, 2010, ISBN 978-1-84882-617-5, 379pp

doi:10.1017/S095679681200010X

The book presents the notion of schematic functional programming and demonstrates not only the concept but also how schematic functional programs can be processed, and how this approach can be used to develop small and even complicated programs.

A reader need not be familiar with the concept of functional programming, but on the other hand, it is probably expected that a reader is familiar with programming and programming languages. The development environment connected with schematic programming can be found on Internet and downloaded (but only for Windows 32bit systems) but it seems not to have been under active development for several years. Nevertheless, a reader can easily try the examples and follow the flow of the book.

Sometimes it is necessary to have broader knowledge to understand the examples presented, as they are built over problems from various areas of computer processing. Even when the book presents introduction to such areas (e.g. to Bayesian theory), it is not sufficient to fully understand the solution. Thus, a question comes to one's mind, whether such an introduction is necessary, as it is of little use for those who are familiar with the subject, but not adequate for those who do not know the subject. A reader familiar with these areas can go through the book quite easily and try to use the constructs and typical patterns of schematic programming, and, in such a way, learn how to use schematic programming and become familiar with it.

It is quite misleading to say that this is a functional language. It would be very helpful if the authors could give a clear explanation of how it is (and isn't) functional, so that a reader could recognize that it is an imperative language with stateful development and runtime environment, but that the schematic language itself exploits functions.

This also raises another issue – the theory behind the schematic programming is not presented at all. This includes no binding to any calculus, no presentation of any formal system that could be used as a formal basis of the schematic language or the underlying language. Thus, the 'theory' in the book title may be misleading.