Formal Meta Engineering Event-B: Extension and Reasoning The EB4EB Framework

Peter Riviere

Formal Meta Engineering Event-B: Extension and Reasoning The *EB4EB* Framework

Peter Riviere

IRIT/INPT-ENSEEIHT
2 rue Charles Camichel Toulouse 31000, France
peter.riviere@toulouse-inp.fr

1 Context

State-based Formal methods have been used to design and verify the development of complex software systems for a long time. Such methods are underpinned with solid mathematical concepts. Event-B [1] belongs to this family of methods. It advocates a correct-by-construction approach to model a complex system. It is based on set theory and first-order logic. It comes with a powerful integrated development environment called Rodin [9].

The use of formal methods must satisfy the needs of the end user by allowing for scalability, portability, expressiveness, and modularity, among other things. Many key features are currently supported by the Event-B language either in the core modelling language or through specific plugins (e.g. composition plug-in [10], Theory plug-in [2, 4], code generation plug-in [8, 6]). The Theory Plug-in [2, 4] extends Event-B to allow for the definition of new data types, theories, and operators in order to enhance the expressiveness of the formalism. For example, to handle the continuous behavior of a hybrid system for designing a safe controller, domain specific features, related to continuous mathematics [5], have been developed in the form of theories.

The Event-B language requires advanced modelling and reasoning concepts in order to capture the notion of model, proof obligation (PO) and proof process. Currently, verifying interesting properties such as deadlock freeness, event scheduling, liveness, etc., requires ad hoc modelling by the designer. Establishing these properties is based on the use of automatic and/or interactive proof systems and/or model checkers.

Due to a lack of access and explicit manipulation of Event-B concepts, it is quite impossible to express a generic property on these concepts in a theory within a generic definition. Indeed, there is no mechanism in Event-B allowing a designer to define, at a higher order level, additional reusable POs.

The above mentioned issue has been addressed by the development of several plug-ins as Rodin tools. Examples are machine compositions and decomposition, event scheduling, code generation, and translation that have been developed using Eclipse. There is a lack of evidence to guarantee the functional correctness of such developed tools. For example, how can we assert that the machine composition and decomposition plug-in behave correctly.

2 Motivation and Objectives

The above mentioned plug-ins enrich Event-B by introducing either new data types (e.g. using the theory plug-in [2, 4]) or by externally defined specific programs that manipulate Event-B models (e.g. composition/decomposition plug-in). None of the above approaches allow to manipulate Event-B models as first order objects. A typical example is the case of deadlock freeness. Three possible options are currently possible: either the developer writes explicitly this PO in the form of a theorem to be proved in the development, or by writing an external program in the PO Generator (to generate this theorem as a PO), or in the form of a plug-in (generating an Event-B machine with the PO as theorem). Both approaches are error prone (written programs are not certified to be correct) or ad hoc (no reuse, properties need to be written for each specific model analysis).

Offering the capability to manipulate Event-B concepts (models, states, transitions, invariants, variants, guards, POs, etc.) as first order objects will allow the developer to express properties on these objects. For example, deadlock freeness PO can be expressed at machine level if guards and invariants can be manipulated in the modelling language. Such a manipulation is possible if the Event-B theory, as defined in the Event-B book [1] associated to Event-B, is formalised in Event-B itself (reflexive modeling).

So, the objective of our PhD thesis work consists in improving modelling and reasoning capabilities of Event-B through the development and formalisation of a theory of Event-B in Event-B. Indeed, we propose to develop the *EB4EB* framework grounded on a set of theories defining data types for Event-B concepts, operators manipulating these concepts and a set of proved theorems precising their semantic properties. In addition, we propose to build other theories to introduce other Event-B models domain specific analyses in the form of properties expressed on the Event-B concepts that describe POs on the analysed models.

Note that the soundness of this framework for Event-B extension and reasoning developed in Event-B shall preserve the core logical foundation of original Event-B models. The main objectives of our work are summarised as follows:

- Analyse and identify the fundamental Event-B concepts and properties that define the notion of machine. Then, using a context or a theory, formalise these concepts (modelling in the small).
- Analyse and identify the Event-B refinement operation for events, data and machine, and then formalise it as a context or theory (modelling in the large).
- Deploy the proposed approach for enhancing reasoning mechanism like deadlock freeness, reachability, etc.
- Introduce new modeling and reasoning mechanisms to handle domain specific analyses of Event-B models. For example, continuous behaviour, human machine interaction, and so on.
- Deploy the proposed approach for analysing and certifying the existing plug-ins, such as composition/decomposition, code generation, etc.
- Allow the capability to Import/Export Event-B models as First Order Logic formulas in other proof tools, as these models become expressed as instances of Event-B theories.

3 Proposed Approach

3.1 Overview of the approach

An *Event-B system model* consists of *context, machine* that focuses on formal modelling to describe system behaviour using refinement approach. Additional theories may be required to axiomatise new definitions and data-types in either contexts or *theory* components.

Our approach focuses on the development of Event-B theory axiomatising Event-B concepts. We propose a set of datatypes, operators, and theorems to specify the Event-B concepts, their relationships and other additional attributes and properties related to these concepts. The obtained meta-theory serves to design a system model as instances of this meta-theory. This instantiation generates a set of new POs.

3.2 Modelling and Instantiation mechanism

Once the theory for Event-B concepts is designed, two main approaches to instantiate it are envisioned, namely deep modelling and shallow modelling.

- **Deep modelling**. All the concepts of an Event-B model to be analysed, variables, events, guards, invariants, substitutions, etc., are defined as instances of the developed meta-theory. An Event-B model is represented as an Event-B context, and POs are described as either theorems or well-definedness POs. Higher order logic and set theory are used to express all the definitions. It can be used as an entry point to the design of an import/export system between other proof assistants.
- **Shallow modelling.** In this case, for each Event-B model to be analysed, we define another Event-B model consisting of a context instantiating the theory with the concepts of the analysed model and a refinement of an abstract generic machine composed of two events *init* and *progress* capturing generic behaviours. This machine is refined to introduce the events of the analysed Event-B model. This method contributes to reduce the proof effort as the POs associated each event become simulation proofs. In the same spirit as TLA$^+$[7], the concepts of *init* and *progress* events are identified similar to *init* and *next* events of TLA$^+$.

These two instantiation mechanisms extend the modeling and reasoning capabilities of Event-B language itself as it makes it possible to define additional theorems encoding other POs (e.g. deadlock freeness). It is important to note that these two instantiation mechanisms are distinct and play an important role in the refinement process. The modeling tool Rodin equipped with proving tools will be used to support all the theories and models. To check the correctness of the developed models, all the generated POs related to well-defined conditions, theorems and properties must be successfully discharged. Instantiation also generates some new POs that must be discharged before any further development. In addition, the developed theory of Event-B can be used for analysing and verifying the core functionalities of existing Rodin plugins.

4 Future Work

The development of Event-B theory is currently in progress. In addition to the developments of the necessary theories, we intend to develop complex case studies to demonstrate the expressiveness and scalability of both deep and shallow mechanisms. Other planned work includes checking the correctness of existing plug-ins like composition/decomposition, code generation, etc., by describing, at the level of the Event-B theory, the operation they encode. Moreover, we plan to deploy the proposed approach for enhancing the reasoning mechanism, such as deadlock freeness, reachability etc. In addition, the proposed approach will be implemented with the theory plug-in and context instantiation developed in the context of the EBRP project. Our long term future work includes to import/export the Event-B theory as well as Event-B models in other proof assistants, through Dedukti [3].

Acknowledgements. This study was undertaken as part of the EBRP (Enhancing EventB and RODIN: EventB-RODIN-Plus) project. We are very grateful to EBRP project members for their valuable discussion and feedback.

References

1. Abrial, J.R.: Modeling in Event-B: system and software engineering. Cambridge University Press (2010)
2. Abrial, J.R., Butler, M., Hallerstede, S., Leuschel, M., Schmalz, M., Voisin, L.: Proposals for mathematical extensions for Event-B. Tech. Rep. (2009)
3. Boespflug, M., Carbonneaux, Q., Hermant, O., Saillard, R.: Dedukti: A Universal Proof Checker. In: Journées communes LTP - LAC. Orléans, France (Oct 2012), https://hal-mines-paristech.archives-ouvertes.fr/hal-01537578
4. Butler, M., Maamria, I.: Mathematical extension in Event-B through the rodin theory component (2010)
5. Dupont, G., Yamine Aït Ameur, Pantel, M., Singh, N.K.: Formally verified architecture patterns of hybrid systems using proof and refinement with event-b. In: Raschke, A., Méry, D., Houdek, F. (eds.) Rigorous State-Based Methods - 7th International Conference, ABZ. LNCS, vol. 12071, pp. 169–185. Springer (2020)
6. Fürst, A., Hoang, T.S., Basin, D., Desai, K., Sato, N., Miyazaki, K.: Code Generation for Event-B. In: Albert, E., Sekerinski, E. (eds.) Integrated Formal Methods. pp. 323–338. Springer, Cham (2014)
7. Lamport, L.: Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc. (2002)
8. Méry, D., Singh, N.K.: Automatic code generation from Event-B models. In: Proceedings of the 2011 Symposium on Information and Communication Technology, SoICT 2011. pp. 179–188 (2011)
9. Rodin sourceforge, https://sourceforge.net/projects/rodin-b-sharp/
10. Silva, R., Butler, M.: Shared Event Composition/Decomposition in Event-B. In: Aichernig, B.K., de Boer, F.S., Bonsangue, M.M. (eds.) Formal Methods for Components and Objects. pp. 122–141. Springer (2012)