A Logical Framework for Integrating Software Models via Refinement

Marie Farrell, Rosemary Monahan, and James F. Power

Maynooth University, Maynooth, Co. Kildare, Ireland
`mfarrell@cs.nuim.ie`

Abstract

We propose the development of an institution-based framework within which software models can be combined not only at different levels of abstraction but across multiple formalisms. Event-B is an industrial-strength formalism that supports refinement, we envisage that the construction of an institution for Event-B, $\mathcal{EVT}$, will not only increase the modularity of Event-B specifications but also provide a foundation for the interoperability of Event-B with other formalisms.

1 Introduction and Motivation

Modern software development focuses on model-driven engineering: the construction, maintenance and integration of software models, ranging from formal design documents through to program code. In the field of formal software development we can prove the correctness of a particular piece of software by reasoning logically about the system. Just as for non-formal development, it can be beneficial to model the aspects of a system using a variety of specialised formalisms to ensure different aspects of its correctness. In formal software engineering we can map between these levels of abstraction in a verifiable way through the process of refinement, which can take place within a single modelling language, or between languages at different levels of abstraction. The ideal scenario has been described as a "theory supermarket", in which a developer can shop for suitable theories with confidence that they will work together.

1.1 Event-B: a tool for Specification, Refinement and Verification

Event-B is an industrial-strength language for system-level modelling and verification that combines an event-based logic with basic set theory. In Event-B, *machines* model the dynamic parts of a system (variables, invariants and events) and *contexts* model the static parts of a system (carrier sets and axioms). An event is composed of a guard (predicate) and an action which is represented as a before-after predicate relating the new values of the variables to the old. Events can happen in any order once their guards evaluate to true, and the theorem provers check that each specified invariant is not violated by any event [3].

A key feature of Event-B is its support for formal refinement, which allows a developer to write an abstract specification of a system and gradually add complexity. The *Rodin Platform*, an integrated development environment for Event-B, ensures the safety of system specifications and refinement steps by generating appropriate proof-obligations, and then discharging these via support for various theorem provers [3]. Event-B has been used extensively in a number industrial projects, such as the Paris Métro Line 14, and is a relatively mature language. However, we believe there are two main areas where the Event-B language needs further improvement:

Modularity: Event-B lacks well-developed modularisation constructs and it is not easy to combine specifications in Event-B with those written in other formalisms. Some current approaches to modularisation in Event-B include two enhancements of the *Rodin* tool:

the decomposition plugin [6] and the modularity plugin [2]. Although both provide some degree of modularisation for Event-B they do not directly enhance the formalism itself, nor is it obvious how they will interact with other variants of Event-B.

Interoperability: Large software systems are often at such a level of complexity that no single formalisaton, encoding or abstraction of can aptly represent and reason about the whole system. This results in the system being modelled many times, often in separate formalisms, thus requiring proof repetition. For example, when developing software using Event-B, it is at least necessary to transform the final concrete specification into a different language to get an executable implementation. Current approaches to interoperability in Event-B consist of a range of *Rodin*-based plugins to translate to/from Event-B, such as the plugin for translating UML state machines into Event-B specifications [7], but these often lack a solid logical foundation.

In summary, the existing approaches to addressing modularity and interoperability issues in Event-B tend to be somewhat ad hoc, causing difficulties for interaction, proof sharing and maintainability. The goal of our research is to develop a set of modularisation constructs for Event-B that will be sufficiently generic, so that they are well understood (particularly in formal terms), and so that they can map easily to similar constructs in other formalisms.

1.2 Proposed Solution: An Institution for Event-B

Heterogeneity and interoperability are central goals in the field of formal methods. The theory of institutions has been identified as a mechanism by which these goals can be achieved. Once the syntax and semantics of a formal system have been defined in a uniform way, using some basic constructs from category theory, then a set of specification building operators can be defined allowing you to write, modularise and build up specifications in a formalism-independent manner [1]. Institutions have been defined for many logics and formalisms [5], including programming-related formalisms such as UML state machines, Hoare logic and CSP.

Devising an institution for the Event-B formalism involves specifying suitable categories and functors for the syntax and semantics of Event-B specifications, and verifying that they correctly meet the axiomatic requirements for institutional descriptions. By incorporating Event-B into the framework of institutions we will achieve three objectives. First, specifications in Event-B will be able to interact with formalisms already in the institutional framework. Second, Event-B will bring a new perspective to institutions, based on its well-developed refinement model. Finally, by integrating an industry-strength approach into this framework we will be much closer to establishing a solid, formal foundation to the theory of MDE.

The overall goal of this research is to fully integrate the process of formal refinement into the model-driven engineering approach so that models expressed in different modelling languages, formalisms and tools can be correctly combined within one formal framework. The resulting framework will support the sharing of refinement steps, and their associated proofs, between different modelling environments. The impact will be an improved software development process which allows the integration of software models, which focus on different aspects of the software, modelled at different levels of abstraction with increased modularity.

2 Providing Modularity and Interoperability for Event-B

In order to build an institution for Event-B, which we call $\mathcal{EVT}$, it is necessary to specify and verify a series of definitions (using category theory) for its syntax and semantics. The verification of our constructed institution is carried out by proving the satisfaction condition which states in formal terms the basic maxim of institutions, that "truth is invariant under change of notation". Full details are available from our website.[1]

Our description of $\mathcal{EVT}$ addresses the identified pitfalls in the Event-B language as follows:

Modularity: By defining $\mathcal{EVT}$ and carrying out the appropriate proofs, we gain access to an array of generic specification building operators [5]. These facilitate the combination (and, $+$, $\cup$), extension (then), hiding (hide via, reveal) and renaming via signature morphism (with) of specifications. Representing Event-B in this way provides us with a mechanism for combining and parameterising specifications. Most importantly, these constructs are formally defined, a crucial issue for a language used in formal modelling.

Interoperability: Institution comorphisms can be defined enabling us to move between different institutions, thus providing a mechanism by which a specification written over one institution can be represented as a specification over another. Devising meaningful institutions and corresponding morphisms to/from Event-B provides a mechanism for not only ensuring the safety of a particular specification but also, via morphisms, a platform for integration with other formalisms and logics.

Since refinement is a key feature of Event-B, it is vital that any semantics for Event-B provide constructs for dealing with this. The theory of institutions is equipped with a richer notion of refinement than that found in Event-B [5], and these can now be used in Event-B specifications. Another benefit of developing an institution-based specification for Event-B is that it provides a formal semantics for the language, something that has not been explicitly developed thus far.

In conclusion, we have successfully specified an institution for the Event-B formalism and proved the required properties to utilise the modularisation constructs. Our current task is that of implementation using the Heterogeneous Tool-Set, HETS, a framework for institution-based heterogeneous specifications [4]. A significant future challenge is the integration of proofs for Event-B, developed using the *Rodin Platform*, into the more general HETS environment.

References

[1] J. A. Goguen and R. M. Burstall. Institutions: abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.

[2] A. Iliasov, E. Troubitsyna, L. Laibinis, A. Romanovsky, K. Varpaaniemi, D. Ilic, and T. Latvala. Supporting reuse in Event-B development: Modularisation approach. In *Abstract State Machines, Alloy, B and Z*, volume 5977 of *LNCS*, pages 174–188. 2010.

[3] M. Jastram and P. M. Butler. *Rodin User's Handbook: Covers Rodin V.2.8.* CreateSpace Independent Publishing Platform, USA, 2014.

[4] T. Mossakowski, C. Maeder, and K. Lüttich. The heterogeneous tool set, HETS. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *LNCS*, pages 519–522, 2007.

[5] D. Sanella and A. Tarlecki. *Foundations of Algebraic Specification and Formal Software Development.* Springer, 2012.

[6] R. Silva and M. Butler. Shared event composition/decomposition in Event-B. In *Formal Methods for Components and Objects*, volume 6957 of *LNCS*, pages 122–141. 2012.

[7] C. Snook and M. Butler. UML-B: Formal modeling and design aided by UML. *ACM Trans. on Software Engineering and Methodology*, 15(1):92–122, 2006.

[1] http://www.cs.nuim.ie/ mfarrell/