Developing the UML-B modelling tools

Colin Snook[0000−0002−0210−0983], Michael Butler[0000−0003−4642−5373], Thai Son Hoang[0000−0003−4095−0732], Asieh Salehi Fathabadi[0000−0002−0508−3066], and Dana Dghaym[0000−0002−2196−2749]

ECS, University of Southampton, Southampton, U.K.
{cfs, m.j.butler, t.s.hoang, A.Salehi-Fathabadi, D.Dghaym }@soton.ac.uk

Abstract. UML-B is a UML-like diagrammatic front end for the Event-B formal modelling language. We have been developing UML-B for over 20 years and it has gone through several iterations, each with significant changes of approach. The first version was an adaptation of a UML tool, the second generated a complete Event-B project, the third contributed parts of an Event-B model, and the fourth (currently under development) provides a human usable text persistence. Here we outline the reasons for these different developments and summarise the lessons learnt.

1 Introduction and Motivation

Towards the end of the last century it was widely recognised that formal modelling is beneficial in reducing specification errors, but despite various arguments regarding the cost benefits of early error detection, it was difficult to dispel the view that they were costly to achieve and required 'special' engineers or mathematicians. We investigated these beliefs through empirical experiments and interviews with industry experts. The experiments [23] established that formal specifications are no more difficult to understand than computer programs of equivalent complexity. However, when interviewed, industry exponents of formal methods warned that it is the choice of useful abstractions that is difficult and requires experience [22]. Abstraction is something of an art and often counter to the nature of engineers used to looking for solutions. Finding abstractions that are amenable to verification tools adds another complication which can only be mitigated by experience and expertise.

We postulated that a visual modelling tool would aid engineers in exploring and choosing different abstractions. This theory was grounded in 'The Cognitive Dimensions of Notations Framework' [5] which provides a "common vocabulary for discussing many factors in notation, UI or programming language design". (In the following, the terms from the framework are shown in italics). Using this framework, we postulated that, for systems modelling, we need *abstractions* for a *close mapping* to the problem domain, but this requires *premature commitment* (early decisions) which makes specification more difficult especially when compounded by *viscosity* (the effort needed to change the specification) which can be high in a large textual specification with many inter-dependencies. The UML-B

diagrams help by increasing the *visibility* of chosen abstractions through visualisation and reducing viscosity. The reason the diagrams are efficient is because a single diagram entity represents many lines of formal specification text compared to a textual specification. A translation tool then converts the diagram into a textual form for formal verification and validation. This iterative *progressive evaluation* alleviates the difficulty of making premature commitments. A more detailed usability assessment of UML-B using cognitive dimensions is discussed in [17].

The B-method [1] is a method of software development using the formal modelling language, B which is based on set theory and first order predicate logic. It supports the concept of abstraction and incremental refinement with verification by proof. Event-B [2] is a formal modelling language for modelling discrete systems. Event-B was developed from the B-method and hence also supports abstraction and incremental refinement with verification by proof. We chose to use B, and later Event-B, as our underlying formal specification language because they provide a notion of formal refinement with strong tool support for verification using theorem provers as well as model checking and animation tools.

We chose to use the UML (Unified Modelling Language) [18] as the basis for our diagrammatic modelling because it was already fairly widespread and therefore familiar within industry. Event-B models are based on set theory which involves collections of instances and their relationships. This has a natural visualisation as an entity-relationship diagram which can be represented using UML class diagrams. Behaviour in Event-B is modelled as events that fire spontaneously when their guards are true and alter the variables using actions that are treated as a set of simultaneous parallel substitutions. Here there are some important differences between Event-B events and UML state-chart transitions. However, a state-machine representation, similar in structure to UML statecharts, is useful for representing the behaviour of Event-B models. Hence we developed the UML-B diagrammatic modelling tools [19,20] and have been supporting and developing them for over 20 years during which time we have enjoyed many collaborations with various industry sectors. Our current research work, industrial case studies and tool installations are shown on our UML-B website [12].

2 History of UML-B

Driven by experience gained through industrial collaboration, UML-B has been developed over the last 22 years, going through several distinct and fundamentally different versions. This section gives a history of the development of UML-B and the motivation for changing to a new approach in each case.

2.1 Version 1 - Extending standard UML

The initial concept of UML-B (in 2000) was to translate from UML into the B formal notation. (This was before Event-B and Rodin existed). Hence the first

version of UML-B [20] was based on the IBM Rational Rose UML tool. Rational Rose provided a visual basic scripting facility for the user to add tooling features to enhance the diagrams. UML-B was implemented as a script that traversed the UML diagrams and output a B model as a text file. The UML-B model was constructed as a standard UML class diagram but with some restrictions and additional properties added as UML stereotypes. Invariants, could be added to classes and guards and actions could be added to class methods, in order to fully specify the behaviour of the model. The notation used for these textual annotations was derived from the target notation, B, but with support for automatic quantification over instances of a class or parameterisation of the contextual class instance ('self'). Here we may have been able to use OCL for the constraint language and possibly, in a declarative style, for actions. However, this would have entailed more work to invent a translation and caused more separation between the specification and the verification languages. For this reason we took the easier route of basing our constrain/action language on Event-B rather than OCL.

The generated B file was then imported into the B-Core tool [4] for formal analysis. Unfortunately, the Rational Rose tool was a Windows-based application, whereas the B-Core tool was only available for Linux operating systems. Therefore the user had to switch to a different operating system in order to analyse the formal model.

2.2 Version 2 - UML-B: Like UML but different

In 2004 the Rodin project [3,15] was started with the aim of developing a new extensible formal modelling platform to support the new Event-B notation for systems modelling. It includes Event-B editors, static checking tools and mathematical theorem provers for verification of the models. This gave an opportunity to greatly improve UML-B and a new version was developed with a different concept from the first version.

- We no longer tried to bend UML to our purpose but instead, developed our own diagrammatic modelling notation borrowing ideas from UML only when they fitted.
- We had an integrated extensible modelling platform based on Eclipse [8,7] which greatly improved the workflow from source model to verification results.
- The Event-B notation was aimed at systems level modelling and so UML-B followed suit. The concept of UML-B was always more aligned to systems level rather than software development, hence Event-B was a better fit for our purposes.

This version of UML-B [19] generated an entire Event-B project from a UML-B project. Hence all modelling had to be done in UML-B since anything the user did to the Event-B model would be overwritten the next time the UML-B was translated. More and more features were added to UML-B in order to support

different modelling use cases. The action and constraint notation for invariants, guards and actions was continued in this version and developed further by adding new features. Class diagrams and state-machines were supported, but both deviated from their UML counterparts in order to provide a better correspondence with the target formalism. It should be noted that through our industrial collaborations we were gradually appreciating the significance of the very different semantics between UML statecharts and UML-B state-machines. An example of this was that users tended to attach the same event to two transitions of the same state-machine expecting one of them to fire depending on which state was active. However, in UML-B this creates two transitions that must fire together and hence never do so (since both sources can never be active at the same time). Therefore we referred to UML-B as being 'UML-like' from this point on and took care to prepare users for the differences.

The UML-B modelling language used the Eclipse Modelling Framework [24] (EMF) where a meta-model is constructed to define the abstract syntax of a modelling language and the EMF tools then generate Java code that can load model instances of that language and serialise (persist) them. The default format for model serialisation is XMI (an XML based notation for model interchange), but this can be overridden with any user-defined serialisation format. For this version of UML-B, the default XMI format was used for serialisation. EMF is a very useful basis for defining modelling notations and we have continued to use it for all our future version of UML-B as well as any other model tooling that we have developed. We used the Graphical Modelling Framework (GMF) [14] to develop the concrete diagram syntax, editors and tooling.

Although this version of UML-B was quite popular with industrial users that were relatively new to formal modelling, a significant portion of users were already familiar with Event-B and would prefer to have the full flexibility of working in Event-B and using the diagram notations more selectively.

2.3 Version 3 - iUML-B: Extending Event-B

In 2008, The 'Deploy' project [13] was started as a follow on from the Rodin project with the aim of promoting the use of the Rodin platform, and its associated plug-ins such as UML-B, in industry. During this project a new version of UML-B was developed that could work alongside Event-B, rather than overwrite the Event-B models all the time.

Since the new iUML-B needed to be an extension of Event-B rather than a separate language, a new EMF meta-model was needed. An Event-B text editor (Camille) was also developed by Heinrich Heine University in Dusseldorf and since both needed an EMF meta-model for Event-B, researchers at Dusseldorf and Southampton, as well as University of Newcastle, worked together to produce a common EMF based framework and meta-model for Event-B [21] which could be used as the basis for future tools. The iUML-B meta-model then extends the Event-B meta-model to support class diagrams and state-machines using a generic extension mechanism built into the meta-model. The iUML-B model

was serialised (i.e. saved/persisted) within a single extension element within the Rodin Event-B model.

In this version of UML-B, the diagrams still generate Event-B elements but not the complete Event-B model. Some parts of the Event-B model are expected to already exist and the diagrams *elaborate* them by providing further details. For example a UML-B class no longer generates the data item (set, constant or variable) that models the set of instances, but it can generate invariants that constrain the set of instances. Similarly, attributes and associations, 'elaborate' existing data elements by generating invariants about their type (being a relation between the containing class instances and the attribute/association type). Class methods and state-machine transitions 'elaborate' events that already exist in the Event-B and contribute extra parameters, guards and actions. This strategy of elaboration allows the modeller to retain control over the Event-B model and choose which parts to model in Event-B and which parts to model diagrammatically in UML-B. (For expediency, the UML-B diagram editor provides an option to create the elaborated elements if they do not already exist in the Event-B).

However, a disadvantage of diagrammatic models is that it becomes more difficult to get a quick overview of all the details in the model. In a textual syntax all of the details of the model are visible in the same view, even if they are complicated to interpret, whereas in a diagram, it is cumbersome to show everything on the canvas. Hence in UML-B certain model details are given in the associated contextual properties view which only becomes visible when the appropriate model element is selected. This led to some users asking for a human readable text persistence for UML-B. Other advantages of a human readable text persistence are that it may be easier to compare different versions of models (provided order is maintained) and to copy and paste sections of models. (Note that the default persistence is XMI (a variant of XML) which is ASCII text, but designed for machine loading and therefore difficult to read).

2.4 Version 4 - xUML-B: A human usable text persistence

The Camille text editor for Event-B was very popular but still serialised models using the Rodin XML-based format. Another problem with Camille was that it is very difficult to extend a concrete syntax. Hence extensions to the Event-B modelling language (e.g. UML-B) were difficult to accommodate. To obtain an extensible and true human usable text serialisation for Event-B we developed a new 'front-end' for Event-B using XText [6,16] which we call 'CamilleX' [9]. Due to the difficulty of extending Rodin models, CamilleX models are written in a separate human readable text file. Hence the source models are separate from the Rodin models which are automatically re-generated for verification purposes when the CamilleX model is saved. Regeneration is efficient since the translations are very fast and the Rodin verification builders are designed to find and reuse existing proofs wherever possible. Further discussion on the development of CamilleX is given in [10]

However, this meant that the UML-B models can no longer be persisted inside the Rodin models. Hence we are now developing an alternative persistence scheme for iUML-B so that its models are stored separately from the elaborated Rodin models. The new UML-B persistence is also based on XText so that we have a human readable persistence for UML-B. We call this version xUML-B.

3 Conclusions

The main lessons we have learnt from our experiences of developing UML-B are.

- Heavily featured semi-formal modelling languages such as UML are difficult to use for precise formally verifiable specification. While UML covers a wide range of users needs it doesn't support the precise mathematical semantics needed for proof. UML can be specialised through profiles and stereotypes, but users are confused if familiar features are not used or represent different semantics. Therefore, it is better not to try to translate UML but to invent a new notation that is better suited to the target formalism.
- A downside of making a new notation similar to a well-known existing one such as UML, is that users may be confused when the model does not behave as they are used to. An example of this is the difference between UML-B state-machines and UML statechart 'run to completion' semantics.
- Model edition, checking and verification needs to be highly integrated so that changes can be quickly assessed.
- While there are many users that are attracted to a self contained diagrammatic notation, experienced users want the flexibility to choose between diagrammatic and textual representations for different parts of a model.
- Even when diagrams are used, users express a strong desire for a human usable textual persistence which helps with maintenance activities such as version comparison and copy and paste as well as enabling a quick oversight of the content

A common reaction to UML-B is to question the decision not to translate standard UML. There is of course a desire not to proliferate new languages unnecessarily. As we have already discussed, the UML semantics is not easily used for representing Event-B semantics. For example, we have extensively researched ways to reconcile run to completion semantics (used in UML statecharts) with Event-B style refinement [11]. An alternative approach would be to develop a new formalised theory of refinement for UML and provide new theorem provers to support it. However, we believe this would be extremely difficult simply because great care was taken to achieve tractable refinement and proof in Event-B by keeping to a simple and appropriate semantics.

Acknowledgements

This work is supported by the HiClass project (113213), which is part of the ATI Programme, a joint Government and industry investment to maintain and grow the UK's competitive position in civil aerospace design and manufacture.

References

1. J-R. Abrial. *The B Book - Assigning programs to meanings.* Cambridge University Press, 1996.
2. J-R. Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, 2010.
3. J-R Abrial, M. Butler, S. Hallerstede, T.S. Hoang, F. Mehta, and L. Voisin. Rodin: An open toolset for modelling and reasoning in Event-B. *Software Tools for Technology Transfer*, 12(6):447–466, 2010.
4. B-Core(UK). *B-Toolkit User's Manual, Release 3.2.* Oxford, UK, 1996.
5. A. Blackwell and T. Green. Chapter 5 - notational systems—the cognitive dimensions of notations framework. In J.M. Carroll, editor, *HCI Models, Theories, and Frameworks*, Interactive Technologies, pages 103–133. Morgan Kaufmann, San Francisco, 2003.
6. M. Eysholdt and H. Behrens. Xtext: Implement Your Language Faster Than the Quick and Dirty Way. In *OOPSLA*, pages 307–309. ACM, 2010.
7. The Eclipse Foundation. The Eclipse Project Website. http://www.eclipse.org, 2009. Accessed Sept. 2022.
8. E. Gamma and K. Beck. *Contributing to Eclipse: Principles, Patterns, and Plugins.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2003.
9. T. S. Hoang and D. Dghaym. Event-B and Rodin Documentation Wiki: CamilleX. http://wiki.event-b.org/index.php/CamilleX, 2018. Accessed Sept. 2022.
10. T.S. Hoang, C. Snook, D. Dghaym, A. Salehi Fathabadi, and M. Butler. Building an extensible textual framework for the rodin platform. *F-IDE 2022, Lecture Notes in Computer Science (to be published)*, 2022.
11. K. Morris, C. Snook, T. S. Hoang, G. Hulette, R. Armstrong, and M. Butler. Formal verification and validation of run-to-completion style state charts using event-b. *Innovations in Systems and Software Engineering*, Mar 2022.
12. The University of Southampton. The UML-B website. https://uml-b.org/, 2021. Accessed Sept. 2022.
13. The Deploy Project. The deploy project website. http://www.deploy-project.eu/, 2008. Accessed Sept. 2022.
14. The Graphical Modelling Project. The GMP project website. https://www.eclipse.org/modeling/gmp/, 2010. Accessed Sept. 2022.
15. The Rodin Project. Rigorous open development environment for complex systems. http://rodin.cs.ncl.ac.uk/, 2004. Accessed Sept. 2022.
16. The XText Project. The XText project website. https://www.eclipse.org/Xtext/, 2020. Accessed Sept. 2022.
17. R. Razali, C. Snook, M. Poppleton, and P. Garratt. Usability assessment of a UML-based formal modeling method using a cognitive dimensions framework. *Human Technology*, 4(1):26–46, May 2008.
18. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley, Reading, MA., 1998.
19. M.Y. Said, M. Butler, and C. Snook. Language and tool support for class and state machine refinement in UML-B. In A. Cavalcanti and D. Dams, editors, *FM 2009: Formal Methods*, pages 579–595, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
20. C. Snook and M. Butler. UML-B: formal modelling and design aided by UML. *ACM Transactions on Software Engineering and Methodology*, 15(1):92–122, 2006.

21. C. Snook, F. Fritz, and A. Iliasov. Event-B and Rodin Documentation Wiki: EMF Framework for Event-B. http://wiki.event-b.org/index.php/EMF_framework_for_Event-B, 2009. Accessed Sept. 2022.
22. C. Snook and R. Harrison. Practitioners' views on the use of formal methods: an industrial survey by structured interview. *Information and Software Technology*, 43(4):275–283, 2001.
23. C. Snook and R. Harrison. Experimental comparison of the comprehensibility of a z specification and its implementation in java. *Information and Software Technology*, 46(14):955–971, 2004.
24. D. Steinberg, F. Budinsky, M. Paternostro, and Ed Merks. *Eclipse Modeling Framework*. The Eclipse Series. Addison-Wesley Professional, 2nd edition, 2008.