Event-B at Work: some Lessons Learnt from an Application to a Robot Anti-Collision Function

Arnaud Dieumegard, Ning Ge, Eric Jenn

Event-B at Work: some Lessons Learnt from an Application to a Robot Anti-Collision Function

Arnaud Dieumegard[1], Ning Ge[1, 2], Eric Jenn[1]

[1]IRT Saint-Exupéry
118 Route de Narbonne, 31432 Toulouse, France
<firstname.lastname>@irt-saintexupery.com

[2]Systerel Toulouse
La Maison des Lois, 2 impasse Michel Labrousse, 31036 Toulouse, France
<firstname.lastname>@systerel.fr

Abstract. The technical and academic aspects of the Event-B method, and the abstract description of its application in industrial contexts are the subjects of numerous publications. In this paper, we describe the experience of development engineers non familiar with Event-B to getting to grips with this method. We describe in details how we used the formalism, the refinement method, and its supporting toolset to develop the simple anti-collision function embedded in a small rolling robot. We show how the model has been developed from a set of high-level requirements and refined down to the software specification. For each phase of the development, we explain how we used the method, expose the encountered difficulties, and draw some practical lessons from this experiment.

Keywords: Formal refinement, software verification, formal verification, anti-collision, Event-B.

1 Introduction

The practical implementation details and the difficulties encountered during the application of the Event-B method by "typical industrial engineers" are usually not widely discussed. Therefore, in the current publication, we share the method *we* have used, the difficulties *we* have encountered, and some lessons *we* have learnt when applying this method to develop one particular function of our small rolling robot [1].

It is worth noting that even though this development was tightly driven by considerations about aeronautical certification, the question of compliance with ARPs [2] or DOs [3]–[5] objectives using Event-B is not directly addressed here.

The paper is organized as follows. Section 2 outlines our development process. Section 3 introduces our case study: the anti-collision function of a small rover. Section 4 details the elaboration of the software requirements using formal refinement. Section 5 covers related works. We conclude in section 6.

2 Formal Refinement in an Industrial Development Process

Our experiment focuses on the following development activities: (i) formalization of the system specification, (ii) definition of a *refinement strategy*, (iii) application of the refinement strategy to elaborate a set of high-level software requirements compliant with the initial specification. Subsequent software production activities are not detailed and are the subject of an ongoing publication [6]. Other activities such as integration or testing are not addressed.

The development process starts with a set of informal requirements expressed in a natural language. In order to optimize the modelling and validation effort, the initial set of requirements is decomposed into disjoint subsets, the processing of which is realized sequentially. Processing a subset of the requirements involves several phases: *formalization,* where requirements are translated into Event-B constructs; *validation,* where these constructs are validated against the initial user specification; *refinement,* where these constructs are made more concrete; *verification,* where the correctness of these constructs is proved. This process stops when (i) all subsets have been processed and (ii) the set of modelling elements allocated to software is completely defined. The overall development process is depicted on **Fig. 1**.

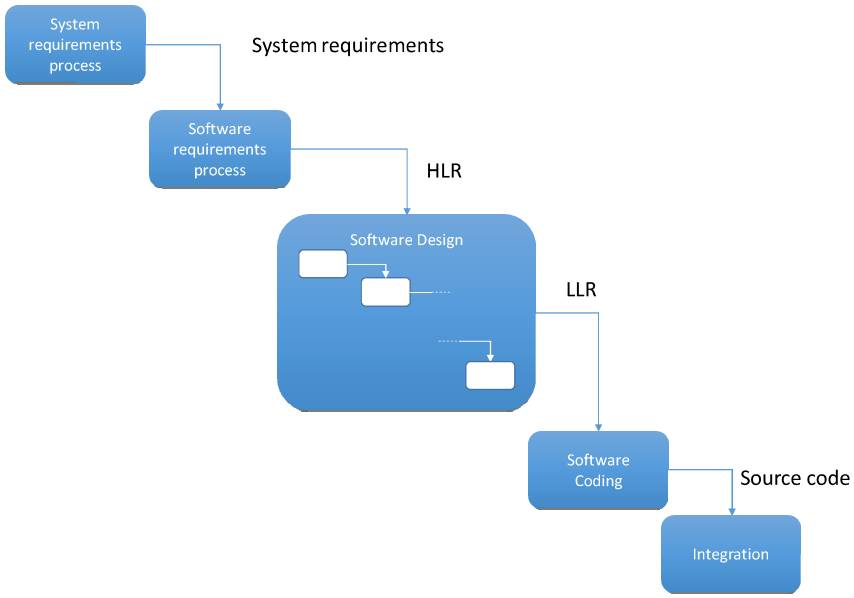

Fig. 1. Overall development process

With respect to a typical development process in the aeronautical domain, this part of the overall process covers part of the *system-level specification* and *design activity* (as per ARP4754 [2]) and part of the *software requirement activity* (as per DO-178C [3]).

In our case, we consider the last refinement of the Event-B model to carry high-level requirements (HLR), i.e., "software requirements developed from analysis of system requirements, safety-related requirements, and system architecture" (DO-178C). The software code will be implemented from those HLR; this part of the process is described in [6].

3 The Case Study

3.1 The TwIRTee Rover and the ARP Function

TwIRTee is the three-wheeled robot (or "rover") used as the demonstrator of the INGEQUIP project conducted at the *Institut de Recherche Technologique of Toulouse (IRT Saint-Exupéry)*. It is used to evaluate new methods and tools in the domain of hardware/software co-design [1], virtual integration, and application of formal methods for the development of equipment [7]. TwIRTee's architecture, software, and hardware components are representative of aeronautical, spatial and automotive systems.
A rover performs a sequence of *missions* (❶ on **Fig. 2**) defined by a start time and an ordered set of *waypoints* to be passed-by. Missions are planned *off-line* and transmitted to the rover by a supervision station (❷). To go from the first waypoint to the last, the rover moves on a track materialized by a dark line on the ground. In a more abstract way, a complete mission can be modelled by a *path* in a *graph* where *nodes* represent waypoints, and *edges* represent parts of the track joining two waypoints.

A rover shares the tracks with several identical rovers. In order to prevent collisions, each of them embeds a protection function (or ARP) which purpose is to maintain some specified spatial (❸) and temporal separation (❹) between them. On **Fig. 2**, temporal separations are represented by light green and light red areas superimposed on the map: basically, rover 2 (resp. rover 1) shall never enter the light green (resp. light red).

In our implementation, the ARP essentially acts by *reducing the rover speed* and, in some specific cases, by performing a simple *avoidance trajectory.* To take the appropriate action, the ARP exploit the following information: the map, the position of all other rovers transmitted by a centralized supervision station (❺), and its own position.

For this paper, we rely on a simplified version of the ARP function where some specification elements such as the rovers positions, speeds, decelerations, etc. are represented as discrete values (no use of Real or Floating Point data). Interested readers can refer to another study [9] conducted on this same function but covering different formal modelling aspects.

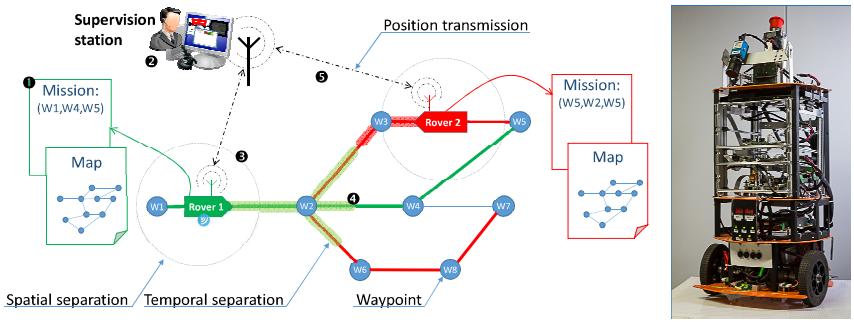

Fig. 2. System overview

3.2 Rodin and Event-B

Event-B [10] is a method to develop systems according to a correct-by-construction approach. It is the system level modelling evolution of the B-method [11] successfully applied in real-size industrial applications [12]. The Event-B method constructs a correct model of a system via a series of refinements of its specification. The correction of a refinement is ensured by proving automatically or manually a set of proof obligations generated from the model.

The Rodin Platform[1] is an Eclipse-based IDE for Event-B that provides effective support for refinement and mathematical proof. The platform is open source, based on the Eclipse framework. Its development started in 2004 during the RODIN project, and continued within the DEPLOY and ADVANCE projects. The community is still active regarding the development. The extensibility of the platform through the use of plugins is of great interest as it allows to rely among others on (i) analysis tools for verification (SMT solvers, model checkers) or validation (animators, simulators generators) of the models and the refinements, (ii) traceability facilities for link with requirement documents, (iii) code generation tooling, (iv) automated refinements methods easing the refinement work.

4 From System-level Requirements to High-level Requirements

In our process, the latest refinement of the Event-B model represents software HLR. As already studied in [10,13], the development of a refinement strategy is the entry point for the definition of Event-B models. It improves the understanding of the requirements by the designer and the robustness of the development process by providing an intermediate formalization phase between requirements and design. Refinement strategy application produces Event-B refinements.

[1] http://www.event-b.org/

4.1 Building a Refinement Strategy

Our refinement strategy is based on Abrial [10], Butler et al [13] and Wen Su et al [14]. The work started with a thorough analysis of the requirements to identify the variables used in the system and classify them as either **uncontrolled** (environment), or **controlled** (system) or **commanded** (operator). Requirements are classified according to the same three categories. The main role of the ARP function is to ensure the absence of collision between rovers by controlling the deceleration of the rover. The controlled variable *deceleration* of the control function is chosen as the first element of focus in the requirements document for the elaboration of the refinement strategy.

Requirements Layering

The refinement strategy defines the processing order of the requirements. This order is determined from the dependencies between variables and, consequently, between requirements. In our case study, we identified the *deceleration* feature as dependent of the occurrence of *conflicts* and *emergency braking*. As a first abstraction, *conflicts* might occur at any time and so might *emergency braking*. Our initial layer of refinement was thus only composed of these three variables.

From this entry point, the next requirements layers are produced by gradually introducing new features such as: *fleet of rovers*, *distances between rovers*, *inhibition by operator* etc. Each feature is attached to a subset of the initial requirements. As some requirements are linked to multiple features, they are attached to multiple layers and their implementation is gradually completed along with the refinement of layers.

Complementary to the previous *horizontal* refinements, *vertical* data refinements are also performed. For instance, the values of the deceleration variable, initially constrained by a simple range in the early refinement, become later constrained by axioms specifying the semantics of deceleration. Similarly, the calculus of the distance between rovers that was simply defined as a value in a range is refined as a shortest path function.

Lessons Learnt

Building a consistent, adequate and applicable refinement strategy is the first step towards the correct understanding of the system and contributes to the correct modelling of the system. If requirement classification is a rather systematic activity, their layering (or sequencing) is more difficult. Layering starts with the identification of an entry point from which the activity starts. Layering may be driven by the identification of the minimal subset of features that ensures the capability to simulate and validate the model at each layer.

4.2 Formalization of Requirements

Formalization starts with the definition of Event-B contexts containing sets, constant variables and constant relations, the definition domain of which are specified as axioms. Then machines are detailed with variables and relations with their definition domain specified as invariants. Variables require the setting of their initial value in the special INITIALIZATION event. Variables shall be used in events specifying the condition under which their value changes (guards) and how their value changes (actions). Event

execution modifies the state of the system. Properties expected to be verified by the system shall be added as invariants of the machine and shall hold in every event.

Producing Event-B models from informal specification can be done using multiple approaches. A first approach relies on modelling the states of the system as sets. In that interpretation, state changes are represented by the "movement" of elements from one set to another. This approach has been used for instance in an alternative modelling of our use case in [9] where the study goal was on time and the data refinements relied on the use of real values.

Our modelling approach, depicted in **Fig. 3**, is inspired from [10]. The function is first abstracted as a hierarchic cyclic state machine comprising two states: the first one updates the state of the environment of the system and the second updates the state of the system itself (i.e., performs the function under design). Transition from one state to the other is triggered by dedicated events (`arp_state_env_start` and `arp_state_fun_start`) updating a state variable `arp_state`. Sub state machines are triggered depending on activation variables (`[mm|fm|cm|em]_activated`). This approach provides a clear separation between the environment and the system under design, exposes the execution cycle, and so facilitates the production of the executable code from the model. Unfortunately, exposing the execution cycle of the function may also introduce implementation details too early in the refinement process.

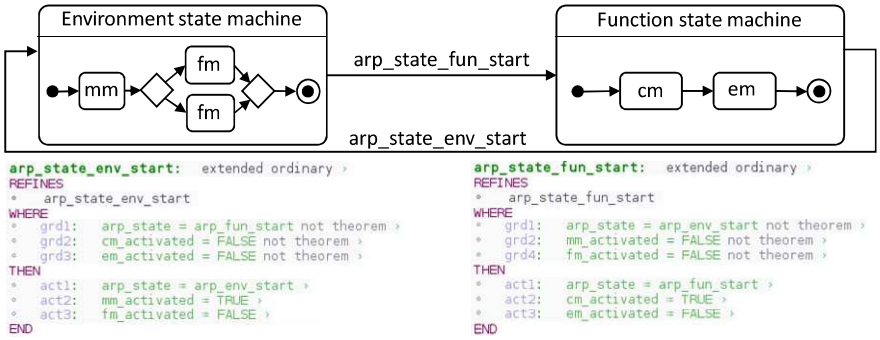

Fig. 3. Event-B model as a circuit

Lessons Learnt

Modelling the system using our approach does suffer from some serious limitations. We assume that all other rovers in the environment do implement the same ARP function as the one under design. For our implementation, this assumption was added as a new environment requirement. Such assumption was not necessary in the alternative modelling approach as every rover in the system was explicitly modelled and each of them implements the same ARP behaviour. Our modelling approach yields an advantage regarding the formal verification: as we do not model all the rovers, a level of universal quantification in the model is removed.

Vertical data refinements produce detailed specifications for variables and for functions. These specifications may be purely declarative or imperative. In the first case, implementation is provided outside of the Event-B world; in the second case, Event-B is used to "code" the function. In our use case, for instance, an imperative model of the

simple “deceleration function” could be easily designed in Event-B. However, this would be much more tedious for the “shortest path function”. Thus we have favoured a pure declarative approach in Event-B, leaving the implementation details to programming languages.

The choice of the “set-oriented” or “finite-state-machine-oriented” modelling approach has an impact on efficiency. The use of sets increases abstraction and reduces the modelling effort, but it increases the implementation work. Reciprocally, using finite state machine approach is less abstract, less compact, more difficult to write, but simplifies the implementation. Additionally, this approach also facilitates the automatic discharging of POs but at the price of adding invariants to propagate the values of variables changed in sub states to the final state of the state machine. Note also that the nature of the variables and the system under design are likely to favor one or the other modelling approaches.

Finally, it is worth noting that *writing* Event-B models does not require more *knowledge* than writing software. While using first order logic and set theory is a shift from classical software engineering methods, this belongs to the mathematical background of any engineers. However, writing Event-B model requires a strong capability of *abstraction* and a capability to describe without being able to execute…

4.3 Verification of Refinements

Verification of formal refinements in the Event-B methodology relies on the discharging of automatically generated proof obligations (PO). POs can be automatically discharged using predicate provers embedded in the Rodin toolset. Plugins have been developed to leverage the increasing capabilities of SMT solvers such as Alt-Ergo[2], Z3[3], CV4[4], or others. Formal verification is conducted in parallel with formal refinement: as soon as any element is added in an Event-B model, PO are generated and potentially discharged automatically. In some way, this can be related to the automatic syntactic verifications performed by current IDEs.

Refinement Verification in Practice

The number of generated POs increases with the size of the model. Even with automatic verification provided by embedded PP and SMT solvers, some POs remain to be proved “manually”. Hopefully, the proof plug-ins in Rodin are easy to use and very intuitive for the users, and thus is of great help when manual proofs are required.

Unfortunately, diagnose *why* some PO fails to be discharged manually or automatically remains difficult. The reason may be that the property simply does not hold, or that either the automatic prover or the user is not able to carry out the proof. In the latter case, reasons may be the limited capabilities of the human or mechanical prover, or missing lemmas. Discriminating the various situations is very hard and may require a significant (but hard to estimate) effort.

---

2 http://alt-ergo.lri.fr/

3 https://github.com/Z3Prover/z3

4 http://cvc4.cs.nyu.edu/web/

Rodin embedded prover can be adapted through the definition/modification (with a graphical interface) of profiles. Profiles customization finds its interest in case dependent models as it provides tactics adapted to specific goals to be proved. We relied on profiles customization in our use case in order to add tactics such as "domain rewriting" that were of great help for the automation of the proof work.

Part of the proof work was additionally assisted by adding "helper" invariants. This was unfortunately not enough to fully automate the formal verification, as about 1% of the proofs remained to be done by hand (a total of 2442 POs including 15 proven by hand). Part of the remaining proofs relate to the use of non-linear arithmetic for which automatic provers are not really efficient. We dealt with these proofs by adding theorems adapted to the proof goals and by performing their proof by hand. The necessary work was not complex but is time consuming due to the manual search for missing theorems.

Lessons Learnt

Formal verification is the most time consuming activity in the refinements process. This work is complex and requires experience and specific skills when automatic proof fails to discharge all POs. Worse, the effort to complete a proof is difficult to estimate. This problem is made even more critical due to the fact that no guidance can be provided to complete a proof.

On the other side, proofs performed fully automatically and immediately may cover other difficulties. Hence, our first proofs were performed in no time due to contradictory axioms/invariants/guards. Unfortunately, avoiding such inconsistencies is difficult and detection cannot be done automatically. So we relied on the voluntary insertion of inconsistent axioms/invariants/guards to check for the consistency of the other axioms/invariants/guards.

After a relatively short training on the Event-B methodology, formalism and proof techniques, it appears to us that modelling systems and proving them using the Rodin toolset is a task that is accessible to engineers with some background in mathematical logics. However, the time needed for the modelling and verification of a system remains difficult to estimate. Worse, the effect of a simple model modification on the proof effort (especially, manual) is difficult to estimate. We really miss appropriate modeling guidance.

4.4 Validation of Formal Requirements

Ideally, the set of requirements is consistent and complete at each refinement level. In reality, it is very likely that some requirements have been ignored, misunderstood, or badly transcoded. As the rework of an Event-B model is fairly expensive, it shall be validated as early and often as possible.

Executing the model has been identified by Event-B experts as the only mean to achieve validation [15,16]. The production of simulators has been the subject of many works [17]–[19] and tools have been developed for this purpose.

Simulator-Based Validation

In our experiment, we relied on ProB [20] complemented by B-Motion [21] and JeB [22] as validation tools. The last two additionally provide means to graphically represent the execution of the model: this greatly improves stakeholders' ability to validate the Event-B models.

During the phase of requirement analysis, we developed a simulator including movement dynamics of the rovers on a map using ScicosLab[5] as depicted in **Fig. 4**. The purpose of the simulator was to validate our understanding of the specification. Such simulator also has the interesting effect of producing simulation scenarios that can be used as test vectors fed to the Event-B simulators [19]. Simulations relying on such values directly contribute to the validation of Event-B models as they rely on pre-validated sets of values. Integration of third party simulators and produced values can be technically done relying on FMI (Functional Model Interface) and the related plugin developed for integration in the Rodin platform [17].

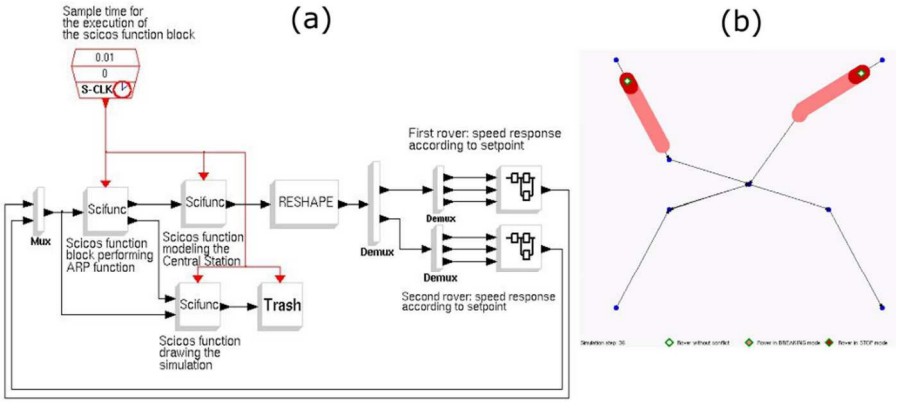

Fig. 4. ScicosLab simulator with graphical display (b) and underlying model (a)

Developing Event-B simulators is easy, especially during the first steps of refinement. However, generating actual input vectors for the simulation can be quite tedious and complex when the variables or constants are specified using non-deterministic expressions.

We relied on JeB for the generation of a web-based simulator and for the generation of values for constants. JeB provides an automatic translation of Event-B models to an executable JavaScript implementation. It is then possible to provide JavaScript functions computing the values for constants (resp. variables and parameters). Such functions produce values that are pretty-printed using Event-B notation. These values can then be used in the original Event-B model making JeB a very handy tool for the production of test vectors for complex data (relations values as sets of pairs etc…). Computed values correction is formally verified using PP and SMT solvers when they are injected in the Event-B model. In our ARP function we produced values of the refined function for the calculus of the deceleration to be applied by the rover using JeB.

[5] http://www.scicoslab.org/

In control systems, *liveness properties* or correctness properties such as *deadlock freeness* shall be verified to ensure the responsiveness of the system. Simulation can be used to obtain a first level of confidence on the absence of deadlocks, before resorting to formal proof. Deadlock freeness theorems can be generated using dedicated Rodin plugins, but depending on the model size, their verification may become very challenging. Verifying these properties can also be done using model checking. But this approach suffers from the classical limitations of model checkers. In our experiment, we used a translation to another formalism and toolset (HLL and S3, see [6]) after introducing a scheduling sequence of events to the system under design to tackle more efficiently and automatically the verification of those properties.

Lessons Learnt

Validating a formal model with respect to a set of informal requirements is a difficult task. Hopefully, the Event-B environment provides a set of very helpful animation tools. Animation allows stakeholders to *see* the behavior of the formal model and validate it. Furthermore, it allows to assess reachability and liveness properties that are difficult (and sometime impossible) to express directly on the Event-B model and to formally verify these properties using model checking. However, as for any test-based approach, confidence on the validation depends on the coverage of the validation scenarios.

4.5 Model Review

The review activity in a classical development process aims at ensuring the correct implementation of requirements as code or the correct refinement of requirements, to detect inconsistencies and misinterpreted requirements, and enforce the use of development standard (e.g., code writing standards). Here, we consider three specific goals: ensure a correct encoding of the designer's intent, reduce the verification effort, and support traceability.

Ensure Correct Encoding of Designer Intent

The correct encoding of the designer intent is ensured by the validity, correctness, consistency and completeness of the formal model with respect to the requirements. We provide here multiple elements supporting this goal.

Introduction of verification lemmas is a starting point advocated in many publications to assess the consistency of an Event-B model. As already stated, success in proving obviously false theorems/invariants/guards put in contexts/machine/events allows to detect inconsistencies in contexts/machine initialisation/event guards and parameters definitions.

Additional automated tooling for checking expressions could also help in our verification process, as an example, checking if bounded logic variables are used in quantified constructs or writing implications in the body of existentially quantified expressions might raise a warning for the designer.

A *proofreading approach* to model review could also be applied to Event-B models by having a reviewer to rewrite chosen guards and invariants using natural language.

The reviewer would then check if the natural language expressions are indeed correct rewritings of the associated requirements. The opposite approach could also be done and would be safer (reviewer to write the natural language expression of the guard using FOL) but less straightforward for engineers. Proofreading should be focused on complex guards and invariants that are more likely to contain errors and on invariants stating key properties of the system under design.

Minimize Verification Effort

Verification is one of the most expensive activities in the development of embedded critical systems. Minimizing verification efforts is thus of primary interest.

To facilitate the (possibly automatic) verification process, we have to add additional lemmas to the model. Those lemmas were explicitly identified as "helper" lemmas, so as to ease the work of assessing the correction of the model. After several modifications of the model, some of those lemmas became unnecessary and were removed from the model to lighten the verification. It is worth noting that some tautologies were kept in the model even though they did not bring additional information as they appeared to be very helpful to support "case splitting" and simplify the automatic proof.

The verification effort obviously strongly depends on the ability for the verifier to understand the model. One way to achieve this goal relies on the compliance to a set of well-defined modelling rules compiled in a "modelling standard", in a way similar to what is usually done for software coding. Many rules for code writing such as MISRA-C[23] can be applied to the writing of logical expressions: avoid deep nesting, avoid too long lines of code, line breaks position according to operators, indentation consistency, parenthesizing consistency, avoid having two operator of different precedence at the same level of indentation. Verification effort can also be strongly reduced by an appropriate organization of the models. For instance, in our experiment, we applied the following rule about model elements ordering: "the order of declaration of constants, variables or parameters should match the order of appearance of their respective definition (axioms, invariants, guards)".

It is obvious but worth noting that adding comments in the model significantly contributes to a better understanding of the intent of the designer and of the structure and choices made during the design process. Comments shall be of help and not state obvious information.

Existing tooling may also simplify the models and thus impact its understandability. For instance, the "theory" plugin provides the capability to factorize properties or expressions of the model and thus simplifies the writing (and, later, the understanding) of complex Event-B models.

We have provided here a few examples of good practices for the writing of an Event-B model to produce more readable, reviewable and thus understandable models. There exists many works and standards used in the industry to ensure such properties for code but to our knowledge there is a minimal work done on applying this to logical specification. We plan on tackling these with more details on a dedicated publication.

Traceability

Aeronautics certifications require to trace each design elements to some requirement. The corresponding certification objective is "High-level requirements are traceable to system requirements" (DO178 Annex A, table A-3, objective 6). In our experiment, ensuring traceability during the refinement process first relied on making explicit the mapping between the elements in the informal specification and Event-B constructs. At high level, naming conventions allowed us to link each refinement layer defined by the refinement strategy to its corresponding Event-B machine and context. Newly introduced model element (constant, axiom, variable, invariant, event, guard and action) were commented with the name of the requirement to which it was linked. If an element could not be linked to a requirement, it was marked as "derived" and the corresponding derived requirement was added to the specification.

We decided to use this approach to keep the traceability artefacts visible at all time. An alternative solution would be to rely on the traceability plugin integrated in the Rodin platform that is based on the requirement management toolset of the Eclipse platform[6]. This solution would simplify the traceability review process and avoid cluttering of the models. Unfortunately, it was not available for the version of Rodin we used in our experiment (such integration is planned to be provided at the time of writing).

Lessons Learnt

We advocate that code review can be applied to Event-B models and may help in (i) demonstrating the correct encoding of the intent of the designer in the formal model; and (ii) minimizing the verification effort by adopting appropriate modelling patterns.

Model review against a well-defined modelling standard is a simple and efficient means to enhance the quality of the model and reduce the number of errors. The benefits of such activity strongly overcome its cost. Hence, it shall be an integral part of the Event-B models development process. We believe that the complexity of such a review activity is affordable for software engineers with basic mathematical knowledge.

Additionally, generating appropriate documentation from Event-B models would also greatly simplify the review work. Indeed, the way of displaying models in the Rodin environment is not really adapted to a proper review activity. For instance, a categorization of model elements and comments according to their purpose / role (traceability, design choices, model element meaning, general information …) would greatly help the review process.

Our approach to deal with traceability was applicable to our use case because of the granularity of our requirements. Tracing more abstract requirements to specific model elements would be difficult to manage and verify that way. Relying on an intermediate level of (semi-)formal requirements as advocated in [24] on the use of the "extended problem frame" approach would be more generalizable.

---

[6] http://www.eclipse.org/rmf/

5 Related Works

Research projects have produced a large literature on the methodology and tools around the use of Event-B for system modelling. Projects such as DEPLOY, for instance, have provided some very valuable results on the application of Event-B on industrial use cases [24]. In this work, they rely on the "extended problem frames" approach as an intermediate formalism between informal requirements and Event-B models to further formalize relations between requirements elements and thus simplify the formalization work. Model validation is tackled in their approach using traceability and animation through the use of ProB. To assess deadlock freeness, they rely exclusively on ProB.

A complete approach for the design and conception of a pacemaker system [25] and an adaptive cruise control has been developed by Singh [19]. Formalization of requirements is done through the extraction of modes and variables and introduction of refinement charts [25]. Event-B models are then produced, verified and validated [26]. The whole process is also confronted to a potential use in a software certification environment [27].

Our work on the analysis and formalization of requirements does not provide additional elements compared to previously presented state of the art applications. We advocate on relying on animation technologies to improve the understanding of simulation results by stakeholders by providing graphical simulators generated using B-Motion and/or JeB. Simulation input data may be produced through the use of simulators generators like JeB. We propose to additionally rely on a transformation of Event-B models to HLL for verification and validation. A similar approach is advocated in the FORMOSE[7] project relying on UPPAAL [28]. We propose an additional review process to complement validation relying on software review techniques ensuring a better detection of conception errors and misunderstanding of the specification during Event-B models design.

6 Conclusion

This work focuses on the application of the Event-B method on part of the process followed during an industrial development. We give some lessons and proposed some of the simple practices that we applied during this experiment. Relying on the Event-B methodology for the development of systems provides a framework for the formalization of textual requirements. This is strengthening the traditional error prone formalization step of a software development process. Formal modelling, verification and validation of Event-B models at an early stage provide a very valuable and fast feedback on the correction of requirements.

One important conclusion of our experiment resides in the very fact that we – "standard" software engineers – were able to apply the method on a non-trivial problem in a very reasonable time. This is in particular due to the great maturity of the toolset and the efficiency of the underlying provers. However, this positive conclusion is certainly

[7] http://formose.lacl.fr/

largely due to the natural adequacy of our problem to the method. An additional conclusion of our experiment is that classical verification and validation activities shall be complemented by review activities. They strongly contribute to reduce the number of errors and more generally to enhance the quality of the model.

Before moving to a large scale industrial application, some very important questions remain to be answered: What is the actual usage domain of the methodology, considering the constraints imposed by the capability of the automatic verification means? How robust is the method to a change in the requirements? What are the good modeling practices to enhance this robustness and to reduce the verification effort? Definitely, it is necessary to evaluate the method on different types of systems to detect weak and strong points for its application.

This work will be pursued to answer these questions, and more specifically to address the applicability of the Event-B method in a DO-178C compliant development process. Additional tooling may be necessary for this purpose.

7 REFERENCES

[1] P. Cuenot, E. Jenn, E. Faure, N. Broueilh, and E. Rouland, "An Experiment on Exploiting Virtual Platforms for the Development of Embedded Equipments," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.

[2] SAE, *SAE ARP4754 Certification Considerations for Highly-Integrated Or Complex Aircraft Systems*. Warrendale, USA: Society of Automotive Engineers (SAE), 1996.

[3] RTCA, *DO-178C, Software Considerations in Airborne Systems and Equipment Certification*. Special Committee 205 of RTCA, 2011.

[4] RTCA, "DO-333 Formal Methods Supplement to DO-178C and DO-278A," RTCA & EUROCAE, Dec. 2011.

[5] RTCA, "DO-331 Model-Based Development and Verification Supplement to DO-178C and DO-278A," RTCA & EUROCAE, Dec. 2011.

[6] N. Ge, A. Dieumegard, E. Jenn, and L. Voisin, "From Event-B to Verified C via HLL," Oct-2016.

[7] M. Clabaut, N. Ge, N. Breton, E. Jenn, R. Delmas, and Y. Fonteneau, "Industrial Grade Model Checking Use Cases, Constraints, Tools and Applications," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, Toulouse, France, 2016.

[8] N. Ge, E. Jenn, N. Breton, and Y. Fonteneau, "Formal Verification of a Rover Anti-collision System," in *Critical Systems: Formal Methods and Automated Verification*, M. H. ter Beek, S. Gnesi, and A. Knapp, Eds. Springer International Publishing, 2016, pp. 171–188.

[9] N. K. Singh, Y. Ait-Ameur, M. Pantel, A. Dieumegard, and E. Jenn, "Stepwise Formal Modeling and Verification of Self-Adaptive systems with Event-B. The Automatic Rover Protection case study," presented at the ICECCS, 2016.

[10] J.-R. Abrial, *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.

[11] J.-R. Abrial, *The B-book: Assigning Programs to Meanings*. New York, NY, USA: Cambridge University Press, 1996.
[12] J.-L. Boulanger, *Formal methods applied to complex systems: implementation of the B Method*. 2014.
[13] M. Butler, "Towards a cookbook for modelling and refinement of control problems," 2009.
[14] W. Su, J.-R. Abrial, R. Huang, and H. Zhu, "From requirements to development: methodology and example," in *Formal Methods and Software Engineering*, Springer, 2011, pp. 437–455.
[15] A. Mashkoor, J.-P. Jacquot, and J. Souquières, "Transformation heuristics for formal requirements validation by animation," in *2nd International Workshop on the Certification of Safety-Critical Software Controlled Systems-SafeCert 2009*, 2009.
[16] S. Hallerstede, M. Leuschel, and D. Plagge, "Refinement-animation for Event-B—towards a method of validation," in *International Conference on Abstract State Machines, Alloy, B and Z*, 2010, pp. 287–301.
[17] V. Savicks, M. Butler, J. Colley, and J. Bendisposto, "Rodin multi-simulation plug-in," presented at the 5th Rodin User and Developer Workshop, Toulouse, France, 2014.
[18] F. Yang, "A Simulation Framework for the Validation of Event-B Specifications," Université de Lorraine, 2013.
[19] N. K. Singh, "Reliability and safety of critical device software systems," Ecole Centrale de Nantes, 2011.
[20] M. Leuschel and M. Butler, "ProB: A model checker for B," in *International Symposium of Formal Methods Europe*, 2003, pp. 855–874.
[21] L. Ladenberger, J. Bendisposto, and M. Leuschel, "Visualising event-B models with B-motion studio," in *International Workshop on Formal Methods for Industrial Critical Systems*, 2009, pp. 202–204.
[22] F. Yang, J.-P. Jacquot, and J. Souquières, "JeB: safe simulation of Event-B models in javascript," in *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, 2013, vol. 1, pp. 571–576.
[23] MIRA Ltd, "MISRA-C:2004 Guidelines for the use of the C language in Critical Systems." .
[24] L. Petre, K. Sere, and L. Tsiopoulos, "Deploy Methods: Final Report," D44, Apr. 2012.
[25] D. Méry and N. K. Singh, "Formal Specification of Medical Systems by Proof-Based Refinement," *ACM Trans Embed Comput Syst*, vol. 12, no. 1, p. 15:1–15:25, Jan. 2013.
[26] D. Méry and N. K. Singh, "Real-Time Animation for Formal Specification," in *Complex Systems Design & Management 2010*, Paris, France, 2010, pp. 49–60.
[27] D. Méry and N. K. Singh, "Trustable Formal Specification for Software Certification," in *4th International Symposium On Leveraging Applications of Formal Methods - ISOLA 2010*, Heraklion, Crete, Greece, 2010, vol. 6416, pp. 312–326.
[28] G. Behrmann *et al.*, "UPPAAL 4.0," in *Third International Conference on the Quantitative Evaluation of Systems - (QEST'06)*, 2006, pp. 125–126.