A Requirements-Driven Methodology: Formal Modelling and Verification of an Aircraft Engine Controller

Oisín Sheridan(✉), Rosemary Monahan, and Matt Luckcuck

Department of Computer Science/Hamilton Institute,
Maynooth University, Maynooth, Ireland
oisin.sheridan.2019@mumail.ie

Abstract. The formal verification of software systems often requires the integration of multiple tools and techniques. To ensure the accuracy of any verification done and to ensure the applicability of formal methods to industrial use cases, traceability must be maintained throughout the process so that it is clear what the requirements for the system are and how they are fulfilled. We propose a three-phase methodology for formal verification with the aim of ensuring traceability, built around the Formal Requirements Elicitation Tool (FRET). Our current case study applies this methodology to the use of FRET, Simulink and Event-B for the verification of the software controller for a civilian aircraft engine.

Keywords: Software verification · Formal methods · FRET · Event-B

1 Overview

Despite the wide applicability of formal methods in industrial applications, particularly safety-critical domains such as aerospace, offshore oil and gas, and the nuclear industry, uptake of formal techniques in industry has historically been slow. To remedy this, the VALU3S project[1] aims to evaluate the state-of-the-art verification and validation (V&V) methods and tools and their application to a number of use cases across different sectors.

We have been working on the elicitation of formal requirements for a software controller using the Formal Requirements Elicitation Tool (FRET), an open source tool developed by NASA that allows requirements to be encoded in a structured natural-language called FRETISH [1]. These requirements can be automatically translated into other formalisms, and the use of FRET reduces ambiguity and simplifies the verification process.

[1] The VALU3S project: https://valu3s.eu/.

This research was funded by the European Union's Horizon 2020 research and innovation programme under the VALU3S project (grant No 876852), and by Enterprise Ireland (grant No IR20200054). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Our example application is a software controller for a civilian aircraft engine; the model of the controller in Simulink was provided by our industrial partner on the VALU3S project. The controller is a representative example of a Full Authority Digital Engine Control (FADEC), which is a software system monitoring and controlling everything about the engine, including thrust control, fuel control, health monitoring of the engine, and so on. The controller's high-level objectives are that it should continue to control the engine and respect specified operating limits in the presence of various sensor faults, perturbations of system parameters, and other low-probability hazards.

In this research, we address two main research questions:

1. Can we accurately support traceability of formalised requirements in the implementation of (safety-)critical systems using a combination of formal and non-formal methods?
2. How can we reuse diverse V&V artefacts (proofs, models, etc.) to modularise and simplify the software verification process?

We are interested in the integration of multiple software V&V techniques, to provide a framework for reasoning about the correctness of concrete software implementations with respect to their abstract software models, to provide and evaluate practical tool support for software engineers. To this end, we propose a three-phase methodology for the verification of software systems.

2 Three-Phase Methodology

Our workflow takes requirements in natural-language and a Simulink diagram as input, and enables the formal verification of the system's design against the requirements. In the case of our current use case, these requirements and Simulink model have been provided by our industrial partner on the VALU3S project. Our approach is split into three distinct phases, shown in Fig. 1. First, in Phase 1 we elicit and formalise the natural language requirements using FRET. Then we move on to formal verification either supported (Phase 2A) or guided (Phase 2B) by FRET. The 'FRET-Supported' toolchain uses FRET's built-in translation function to produce contracts in the CoCoSpec language that can be incorporated into a Simulink diagram. The 'FRET-Guided' toolchain uses the formalised requirements to drive the (manual) translation into other formal methods as chosen by the verifier. Both verification phases can be applied in parallel. Finally, Phase 3 involves the assembly of a verification report to capture the verification results and traceability of

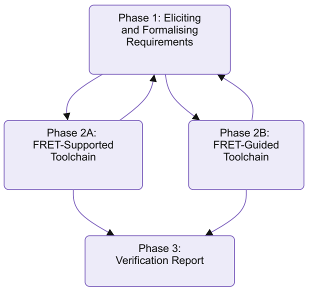
Phase 1: Eliciting and Formalising Requirements
Phase 2A: FRET-Supported Toolchain
Phase 2B: FRET-Guided Toolchain
Phase 3: Verification Report

Fig. 1. High-Level Flowchart of our Methodology. After Phase 1 is complete, Phases 2A and 2B can occur in parallel. Phases 2A and 2B can both highlight deficiencies in the requirements, prompting a return to Phase 1.

requirements. The methodology is presented in full in [2]. A report on our experience using FRET is presented in [3].

3 'FRET-Guided' Modelling

The current focus of this PhD is on Phase 2B of our methodology, 'FRET-Guided' verification in Event-B. A flowchart of this phase is shown in Fig. 2.

Rather than using a direct translation of the FRET requirements, the elicited semi-formal requirements and the Simulink model of the software controller are used to construct a formal model in the Event-B language, which can then be verified. Event-B is a set-theoretic modelling language that supports formal refinement [4]. Event-B has been used to verify models of cyber-physical systems, similar to our case study [5]. However, unlike this previous work, our goal is to model the behaviour of the entire engine control system, rather than using Event-B to model a particular self-contained algorithm. Work has also been done on using FRET as a basis for runtime monitoring with Copilot in [6], rather than theorem proving with Event-B.

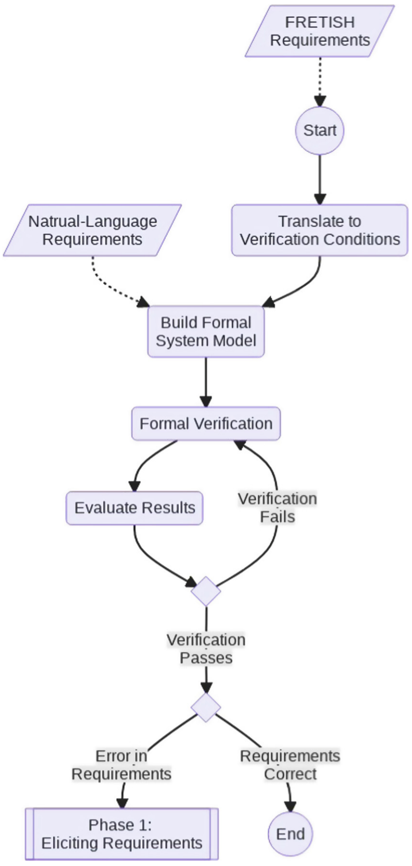

Fig. 2. Flowchart of Phase 2B: Verification guided by FRET requirements. The circular nodes are start and end points, the rectangular nodes are processes, the diamond nodes are decisions, and the rhomboid nodes are inputs or outputs.

Event-B offers an intuitive path to constructing a model of the Simulink diagram. We use a context to define variables external to the system we want to model; in this case, we have a composite input signal including operating limits for the engine (e.g. the shaft speed limit) and commands from the pilot (e.g. the desired thrust). Within the Event-B machine itself, we model the blocks from the system diagram as events where the guards are that the respective input variables have been set, and which then apply the specified function and set the variables representing their outputs. By using the Simulink block diagram as a basis and incorporating a consistent naming scheme to tie the events to their respective blocks, we can easily preserve traceability between the models.

Once the Simulink diagram has been adequately modelled, we can refine the Event-B model by incorporating the semi-formal FRETISH requirements as additional guards and invariants. If a conflict is found between the requirements and the existing model, we can return to the Simulink diagram to check whether this represents an error in the translation to Event-B or a failure of the diagram to meet the requirement in question. We may also find an error in the requirements themselves, prompting a return to the requirements elicitation phase.

4 Future Work

After modelling the system in Event-B, we will compare the verification process in Phase 2A and 2B of our methodology, investigating how both techniques can be utilised in parallel to reuse V&V artefacts to modularise and simplify the software verification process. We will also look into techniques to formally guarantee consistency in translation between models and ensure traceability in both directions, such as using institution theory to verify the translation or checking the Event-B model against the LTL specification using ProB.

Additionally, we are looking at ways to improve FRET with new functionality. Currently, FRET allows the user to define informal parent-child relationships between requirements, but we would like to expand this to support true formal refinement. This would be a great aid to both the requirements elicitation process and supporting traceability alongside other formalisms. We are also working on applying refactoring techniques to FRETISH requirements. Refactoring would minimise duplication of information across requirements, and so would streamline the elicitation process and remove opportunities for error. We discuss refactoring in full in [7].

References

1. Giannakopoulou, D., Pressburger, T., Mavridou, A., Schumann, J.: Generation of formal requirements from structured natural language. In: Madhavji, N., Pasquale, L., Ferrari, A., Gnesi, S. (eds.) REFSQ 2020. LNCS, vol. 12045, pp. 19–35. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44429-7_2
2. Luckcuck, M., Farrell, M., Sheridan, O., Monahan, R.: A methodology for developing a verifiable aircraft engine controller from formal requirements. In: IEEE Aerospace Conference (2022)
3. Farrell, M., Luckcuck, M., Sheridan, O., Monahan, R.: Fretting about requirements: formalised requirements for an aircraft engine controller. In: Gervasi, V., Vogelsang, A. (eds) Requirements Engineering: Foundation for Software Quality 2022. LNCS, vol. 13216, pp. 96–111. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-98464-9_9
4. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
5. Bourbouh, H., et al.: Integrating formal verification and assurance: an inspection rover case study. In: Dutle, A., Moscato, M.M., Titolo, L., Muñoz, C.A., Perez, I. (eds.) NFM 2021. LNCS, vol. 12673, pp. 53–71. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-76384-8_4

6. Perez, I., Mavridou, A., Pressburger, T., Goodloe, A., Giannakopoulou, D.: Automated Translation of Natural Language Requirements to Runtime Monitors. In: Fisman, D., Rosu, G. (eds) Tools and Algorithms for the Construction and Analysis of Systems 2022. LNCS, vol. 13243, pp. 387–395. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_21
7. Farrell, M., Luckcuck, M., Sheridan, O., Monahan, R.: Towards Refactoring FRETish Requirements. (2022). https://arxiv.org/abs/2201.04531. (to appear)