FRETting and Formal Modelling: A Mechanical Lung Ventilator

Marie Farrell Matt Luckcuck Rosemary Monahan Conor Reynolds Oisín Sheridan

Department of Computer Science, The University of Manchester, Manchester, UK

School of Computer Science, University of Nottingham, Nottingham, UK

Department of Computer Science, Maynooth University/Hamilton Institute, Maynooth, Ireland

27th of June 2024

Introduction

Overview

- We describe a methodology that captures the requirements of the ABZ 2024 case study, the Mechanical Lung Ventilator, using the Formal Requirements Eliciation Tool (FRET)
- Our workflow uses the requirements, written in FRET's structured-natural requirements language FRETISH, to guide the development of a formal model in Event-B.
- Our goal was to examine how formalising the requirements could uncover problems in the requirements, thus improving the requirements set and helping with the construction of a system model

Mechanical Lung Ventilator: ABZ Case Study

Case Study Overview

- Many requirements in the documentation.
- Partitioned into
  - Functional Requirements (FUN),
  - Values and Ranges (PER),
  - Sensors and Interfaces (INT),
  - Alarm Requirements (SAV),
  - GUI Requirements (GUI),
  - Controller Requirements (CONT), and
  - Alarms (AL).
- Some requirements have 'child' requirements; for example, FUN6 is decomposed into FUN6_1–6.
- Requirements also reference others; for example, CONT4 refers to FUN6.

Mechanical Lung Ventilator: ABZ Case Study

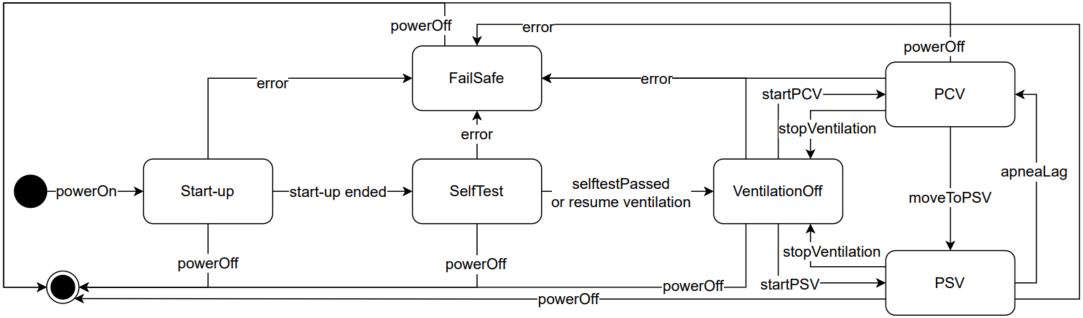
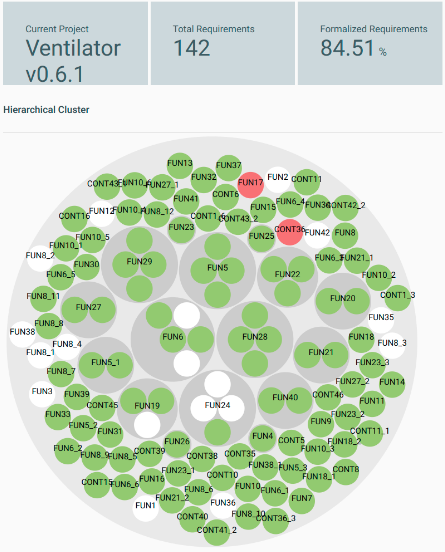


Figure 1: The controller state machine is labelled as Fig 4.1 in the case study documentation.

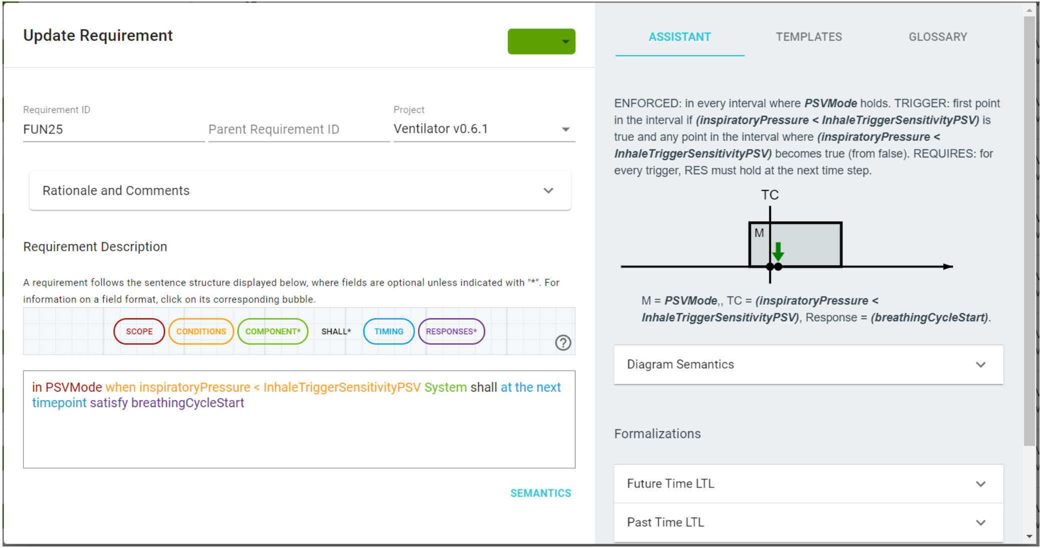
Formalisation with FRET

FRET

- An open source tool for requirements engineering developed by NASA
- Requirements are written in a structured natural-language called FRETish
- FRET provides automated translations from FRETish to CoCoSpec contracts, which can be verified with the Kind2 model checker, and Copilot runtime monitors
- Formalised requirements are indicated in green, those in white have not been formalised, and a red circle indicates invalid FRETish



The Formal Requirements Elicitation Tool (FRET)



Formalisation in FRETish - Methodology

Method

- We focused on the Functional and Controller requirements from the case study document. In total, we formalised 121 requirements in FRET, out of 142 total natural-language requirements in these categories.
- The formalisation was performed in stages, producing multiple versions of the requirements set:
  - v0.1 and v0.2 comprised the initial formalisation of the FUN requirements
  - v0.3, v0.3.1, and v0.4 included revisions to better align with the case study documentation where possible, and fix invalid variable names
  - v0.5 and v0.5.1 formalised the Controller requirements
  - v0.6 and v0.6.1 updated all requirements to use explicit timing conditions
- For traceability, we created FRETish requirements for all of the FUN and CONT requirements, even those that could not be formalised

Formalisation in FRETish - Examples

FUN.7	If the self-test fails, the user shall be warned that the system is out-of-service. In addition, any other operations shall be not allowed
	`in SelfTestMode if selfTestFail System` shall `at the next timepoint satisfy OutOfServiceWarning & FailSafeMode`
FUN.22	In PCV mode it shall be possible to initiate with the push of a single button a lung recruitment procedure, termed Recruitment Maneuver (RM)
	`in PCVMode when RMButton System` shall `at the next timepoint satisfy RM`
CONT.19	If the SelfTest fails, the controller shall not be able to proceed to ventilation
	`in SelfTestMode if SelfTestFail Controller` shall `until off satisfy !StandbyMode & !ventilating`
CONT.32	The inspiration phase lasts until the inspiration peak is reached but no later than the *max_insp_time_psv* is over. After that the expiration phase begins.
	`in PSVMode Controller` shall `until (P_insp >= MaxP_insp | inspClock >= inspiratoryTime) satisfy inspiratoryPhase`

Formalisation in FRETish - Metrics

scope-option	null = 49, in = 70, before = 1, after =1
condition-option	null = 51, trigger (regular) = 70
timing-option	null/eventually =22, until =6, always=34, after=5, for =4, next=50
parent-child	41 child requirements were assigned a parent requirement
Total Requirements	121 specified in FRETish, of 142 natural-language requirements

Fields

- FRET generates a Metric Temporal Logic (MTL) semantics for requirements using template keys
- Each template key is a tuple of: [*scope-option*, *condition-option*, *timing-option*]
- We used the `scope` field wherever the requirements explicitly mentioned a syetm mode

Formalisation in FRETish - Metrics

Timing

- Initially, we only included timing where it was explicitly mentioned in natural-language.
- On a second pass, we rechecked the timing conditions and added them explicitly.
- We usually used:
  - `always` when the requirement had no conditions,
  - `eventually` for events that would take an indeterminate amount of time (e.g. waiting for a process to finish or for user input), and
  - `at the next timepoint` for a response triggered by an event or button-press. We chose at the next timepoint instead of immediately to represent the time taken to react to the trigger and generate the response.

timing-option	null/eventually =22, until =6, always=34, after=5, for =4, next=50

Formalisation in FRETish - Analysis

Inconsistencies

- We encountered some cases where the Functional and Controller requirements didn't quite align, or where the language used wasn't entirely consistent.
- The mode that comes after the self test has passed and before the system moves to PCV or PSV mode is called "Standby Mode" in the FUN requirements, but is named "VentilationOff" in the CONT requirements.
- CONT24 and FUN22 refer to the Recruitment Maneuver. FUN22 says the maneuver should be initiated "with the push of a single button", which seemed to imply that the maneuver starts immediately when the button is pressed. However, CONT24 says that the maneuver should start at the end of an inspiration phase (if it has been set by the GUI).
- Formalising requirements in a structured language like FRETish helps to find cases like these where a requirement lacks important details.

Formalisation in FRETish - Analysis

Unformalised and Invalid Requirements

- The unformalised requirements often related to capabilities of the overall system, rather than specifiable behaviour
  - e.g. FUN.1: *"The system shall provide ventilation support for patients who require mechanical ventilation and weigh more than 40 kg (88 lbs). Rationale: ventilation of children and infants is more challenging"*,
- Similarly, there was no meaningful way to capture the *"Measured and displayed parameters"* requirements without a more detailed understanding of the sensors and GUI
- Some requirements were not written in a form that works in FRET. For example, CONT.36 simply reads: *"If the patient is in expiration phase:"*, and rely on its three child requirements to provide details

Modelling in Event-B

Modelling in Event-B

Overview

- Using the natural-language and FRETish requirements as a base, we constructed a model of the ventilator system in Event-B
- The structure of the initial model was based on the *"controller state machine"* diagram from the case study documentation.
- We then encoded the requirements into Event-B in different ways, depending on what they specified. Some requirements were easily represented in a context, others became part of the behavioural event specifications, and some became invariant specifications.

Event-B Model

```
MACHINE mac00
SEES ctx00
VARIABLES mode
INVARIANTS  typeof__mode: mode ∈ Mode

EVENTS
Initialisation
  then  act1: mode := PoweredOff

 Event PowerOn ≙
   when  grd0_1: mode = PoweredOff
   then  act0_1: mode := StartUp

  Event StartUpEnded ≙
    when  grd0_1: mode = StartUp
    then  act0_1: mode := SelfTest

  Event ResumeVentilation ≙
    when  grd0_1: mode = SelfTest
    then  act0_1: mode := VentilationOff

  Event SelfTestPassed ≙
    when  grd0_1: mode = SelfTest
    then  act0_1: mode := VentilationOff

  Event StartPCV ≙
    when  grd0_1: mode = VentilationOff
                ∨ mode = PSV
    then  act0_1: mode := PCV

  Event StartPSV ≙
    when  grd0_1: mode = VentilationOff
                ∨ mode = PCV
    then  act0_1: mode := PSV

  Event StopVentilation ≙
    when  grd0_1: mode = PCV
                ∨ mode = PSV
    then  act0_1: mode := VentilationOff

  Event MoveToPSV ≙
    when  grd0_1: mode = PCV
    then  act0_1: mode := PSV

  Event ApneaLag ≙
    when  grd0_1: mode = PSV
    then  act0_1: mode := PCV

  Event Error ≙
    when  grd0_1: mode ≠ PoweredOff
          grd0_2: mode ≠ Failsafe
    then  act0_1: mode := Failsafe

  Event PowerOff ≙
    when  grd0_1: mode ≠ PoweredOff
    then  act0_1: mode := PoweredOff
END
```

Event-B Model

```
CONTEXT ctx00
SETS  Mode
CONSTANTS
  Failsafe, PoweredOff, VentilationOff
  PCV, PSV, SelfTest, StartUp
AXIOMS
  axm0_1: partition(Mode, {StartUp},
          {SelfTest}, {VentilationOff},
          {PCV}, {PSV}, {Failsafe},
          {PoweredOff})
END
```

Context for the abstract machine, capturing FUN4/CONT1.

```
CONTEXT ctx01
  EXTENDS ctx00
  SETS ValveState, TestResult
  CONSTANTS
    ValveOpen, ValveClosed,
    TestPassed, TestFailed, TestSkipped
  AXIOMS
     axm1_1: partition(ValveState,
             {ValveOpen}, {ValveClosed})
     axm1_2: partition(TestResult,
             {TestPassed}, {TestFailed},
             {TestSkipped})
END
```

Extending context to capture necessary sets and constants related to the selftest process (FUN6_1–6).

Event-B Model

```
Event SelfTestPassedOrSkipped ≙
  REFINES  SelfTestPassed
  any timePoweredOff
    when
      grd0_1: mode = SelfTest
      grd1_1: testPowerSwitch ∈ {TestPassed, TestSkipped}
      grd1_2: testLeaks ∈ {TestPassed, TestSkipped}
      grd1_3: testFF12 ∈ {TestPassed, TestSkipped}
      grd1_4: testPS_EXP ∈ {TestPassed, TestSkipped}
      grd1_5: testOxygenSensor ∈ {TestPassed, TestSkipped}
      grd1_6: testAlarms ∈ {TestPassed, TestSkipped}
      grd1_7: timePoweredOff ∈ ℤ
      grd1_8: timePoweredOff ≤ 15 ∧ is_new_patient = FALSE
    then
      act0_1: mode := VentilationOff
      act1_1: in_valve := ValveClosed
      act1_2: out_valve := ValveOpen
END
```

`SelfTestPassedOrSkipped` event after the first refinement step. This captures requirements FUN10 and some of its children, along with FUN6.

- FUN6: *The system shall have a self-test procedure that ensures the system and its accessories are fully functional and the alarms work*
- FUN10.3: *If "Resume Ventilation" is selected, every step of the selftest procedure FUN.6 can be skipped or optionally rerun individually.*
- FUN10.4: *Once all self-test steps have been completed successfully, it shall be possible to proceed to the Standby Mode.*

Requirements in Event-B

This table outlines how various requirements were captured in the Event-B model

FRETish ID	Context(s)	Event(s)	Invariant(s)	Event-B File(s)
FUN4	✓	✓		mac00, ctx00
FUN5		✓		mac01
FUN5_3		✓	✓	mac01
FUN6	✓	✓		mac00, mac01, ctx01
FUN6_1–FUN6_6	✓	✓		mac01, ctx01
FUN7		✓		mac01
FUN10		✓		mac00
FUN10_1	✓	✓		mac01, ctx01
FUN10_3–FUN10_6		✓		mac01
FUN23		✓		mac01
FUN27		✓		mac01
CONT1	✓	✓		mac00, ctx00
CONT1_1		✓		mac01
CONT1_3			✓	mac01
CONT1_6			✓	mac01
CONT3		✓		mac00
CONT4		✓		mac00
CONT12		✓		mac00, mac01
CONT18	✓	✓		mac01, ctx01
CONT19		✓		mac01
CONT38			✓	mac01
CONT46		✓		mac01

Verification in Event-B

Proofs

- The Rodin Platform generates proof obligations for Event-B models, which can be discharged automatically or interactively
- We were able to discharge all 79 proof obligations generated by Rodin automatically
- Some requirements were verified by construction. For example, adherence to the controller state machine is obtained by constructing a model that evolves following the mode changes indicated by the diagram. Thus, we consider requirements referring to this sequence of states, e.g. FUN4 and CONT1, to be correct-by-construction.
- Other requirements are verified more directly, by inspecting the guard or action of the event that corresponds to the behaviour described by that requirement.

Discussion

- Expected many requirements to become machine invariants, but most basic requirements become machine functionality and are not formally verifiable properties of the machine.
- Clarification of apparently inconsistent requirements difficult without domain experts, sometimes unclear what is meant to happen.
- Wanted to capture functional and controller requirements (not GUI), but some functional requirements mix types: *"If the self-test mode fails, the user shall be warned that the system is out-of-service. In addition, any other operations shall be not allowed."*

Conclusion

Summary

- We used FRET and Event-B to formalise and model the requirements for the ABZ 2024 Mechanical Lung Ventilator case study
- We formalised the Functional and Controller requirements in FRETish, and described the methodology we followed
- We used these requirements to construct a model of the ventilator system in Event-B, and captured many of the requirements
- The FRET and Event-B artefacts are available at: `https://github.com/mariefarrell/abz2024` (link also included in the paper)