Maynooth University
National University of Ireland Maynooth

Paths to Heterogenous Verification

Rosemary Monahan

Principles of Programming Research Group, Department of Computer Science,
Maynooth University

IFIP WG 1.9/2.15 Verified Software
Fourteenth meeting, Paris, France,
1-2 July 2025

Background

- Principles of Programming Research Group - Software Verification
- PACT Research Group - Computational Thinking

https://www.maynoothuniversity.ie/faculty-science-engineering/our-people/rosemary-monahan

Some Recent Projects:

- MAIVV: Modular AI Verification and Visualisation (2021 - 2025, SFI)
- VerifAI: Writing formal requirements using AI (2024 - 2026, SFI ADAPT)
- VALU3S: Verification and Validation of Automated Systems' Safety and Security (2020-2023, EI and H2020 ESCEL)
- Cyclone: A new specification language for verifying/testing graph-based structures.
- A Model Checker for Python (2022 - 2026, SFI CRT Foundations of Data Science)
- A Constructive Framework for Software Specification and Refinement (2012-2023, 2 x IRC)

Paths to Hetrogenous Verification

- Formal Requirements Elicitation via FRET
- Modelling in Event-B
- Deductive Verification with Dafny
- Model Checking with Kind2/NuSMV
- Runtime Monitoring
- Workflows and Toolchains for Verification
- Heterogeneous Verification

Formalising Requirements

Overview of our work in VALU3S Project

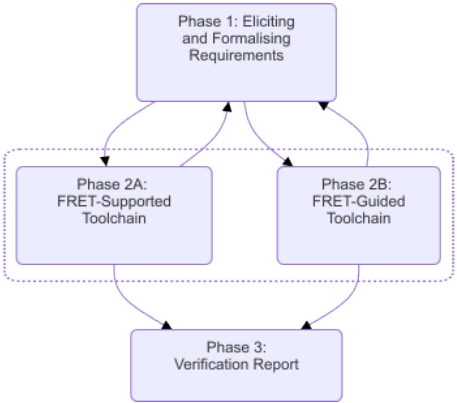
- Use Case 5 from VALU3S Project[a]
  - Aircraft engine's software controller provided by Collins Aerospace
- Formalising requirements in NASA's Formal Requirements Elicitation Tool (FRET)[b]
  - Requirements Elicitation
  - Creating parent-child requirements
- Support for refactoring of requirements in Mu-FRET[c]
- Modelling in Event-B[d]
- Toolchain to support model checking of formalised requirements
- `https://repo.valu3s.eu/`

[a] VALU3S: `https://valu3s.eu`
[b] FRET: `https://github.com/NASA-SW-VnV/fret`
[c] Mu-FRET: `https://github.com/valu3s-mu/mu-fret`
[d] Event-B: `http://www.event-b.org/`

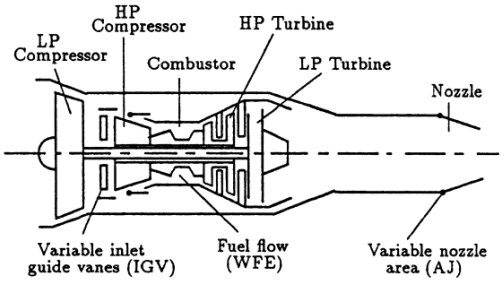
Our Approach

Methodology

- Phase 1: Requirements...
  - Initial requirements
  - Eliciting detail
- Phase 2: Verification...
  - Automatic output from FRET (2A)
  - Guided by requirements in FRET (2B)
- Phase 3: Reporting...
  - Traceability evidence
  - Verification evidence



Aircraft Controller



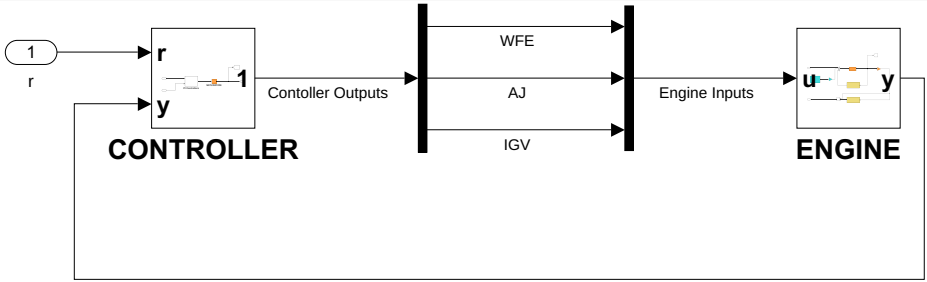
Postlethwaite et al., 1995

Aircraft Engine Software Controller

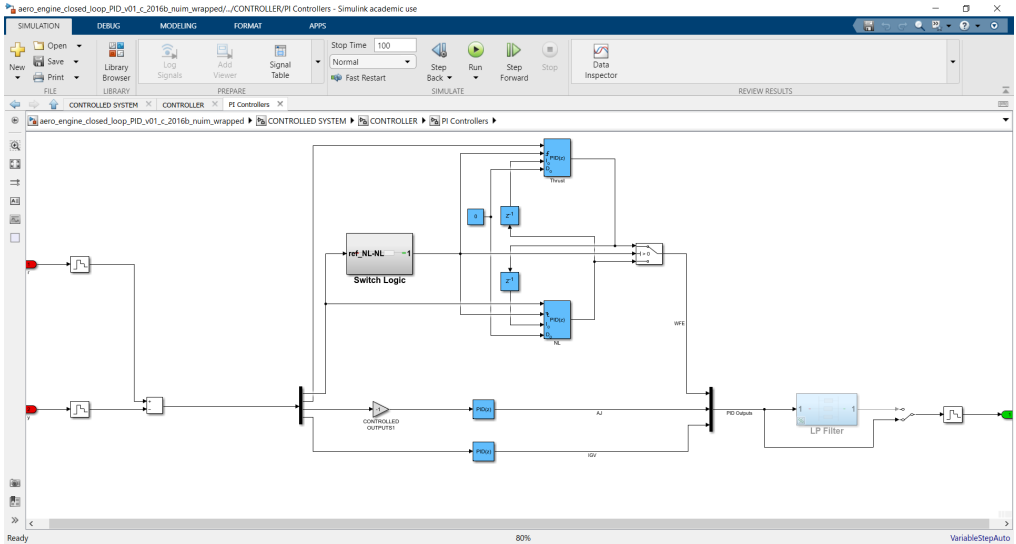
- FADEC: Full Authority Digital Engine Control
- Responds to pilot input and sensor data
- Monitors and controls the engine...
  - Thrust control
  - Fuel control
  - Power management
  - System health monitoring
  - etc

Simulink Diagram

- Supplied by industrial partner
- 'Controlled System' block containing...
  - 'Controller'
  - 'Engine'



Simulink Diagram



Aircraft Controller

Use case 5:
`https://repo.valu3s.eu/use-cases/aircraft-engine-controller`

- 1 Simulink diagram
- 14 Natural-language requirements
- 20 Test cases
- 4 Evaluation scenarios

Requirements Elicitation

Analysis: Aircraft Engine Controller Requirements

- 14 natural-language requirements.

ID	Description
UC5_R_1	Under sensor faults, while tracking pilot commands, control objectives shall be satisfied (e.g., settling time, overshoot, and steady state error will be within predefined, acceptable limits)
UC5_R_2	Under sensor faults, during regulation of nominal system operation (no change in pilot input), control objectives shall be satisfied (e.g., settling time, overshoot, and steady state error will be within predefined, acceptable limits)
UC5_R_3	Under sensor faults, while tracking pilot commands, operating limit objectives shall be satisfied (e.g., respecting upper limit in shaft speed)
UC5_R_4	Under sensor faults, during regulation of nominal system operation (no change in pilot input), operating limit objectives shall be satisfied (e.g., respecting upper limit in shaft speed)

Farrell, M., Luckcuck, M., Sheridan, O., Monahan, R. FRETting about Requirements: Formalised Requirements for an Aircraft Engine Controller. *International Conference on Requirements Engineering: Foundation for Software Quality.* pp. 96–111. Springer (2022).

Requirements

FRETish Example: Requirement 1

- Natural-Language Requirement 1: "*Under sensor faults*, *while tracking pilot commands*, *control objectives shall be satisfied (e.g. settling time, overshoot, and steady state error will be within predefined, acceptable limits)*"
- In FRETish: `if sensorfaults & trackingPilotCommands Controller` shall satisfy `controlObjectives`

Formal Requirements Elicitation Tool: FRET

- supports the formalisation, understanding and analysis of requirements user-friendly interface
- intuitive diagrammatic explanations of requirement semantics
- users specify requirements in restricted natural language, called FRETish, which embodies a temporal logic semantics

Using FRET

UC5_R_1: if sensorfaults & trackingPilotCommands Controller shall satisfy controlObjectives

Future Time LTL

SMV Infinite trace

Format

```
((G (((! (sensorfaults & trackingPilotCommands)) & (X
(sensorfaults & trackingPilotCommands))) -> (X (F
controlObjectives)))) & ((sensorfaults &
trackingPilotCommands) -> (F controlObjectives)))
```

Target: ControlledSystem component.

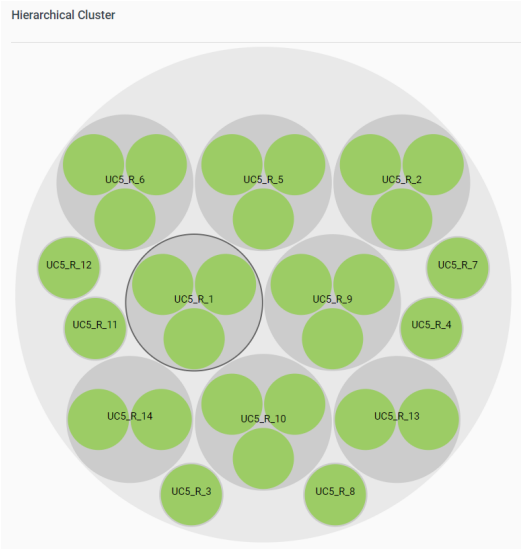
Test Cases?

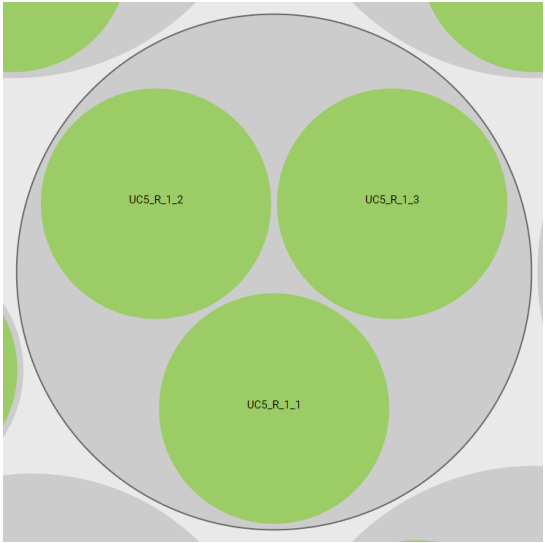
What About the Test Cases... ?

- FRET can link requirements...
  - But no inheritance etc
- Natural-Language Requirements – Parent Requirements
- Test Cases and New Details – Child Requirements

Details in Farrell, M., Luckcuck, M., Sheridan, O., Monahan, R. FRETting about Requirements: Formalised Requirements for an Aircraft Engine Controller. *International Conference on Requirements Engineering: Foundation for Software Quality 2022.*

Using FRET


Hierarchical Cluster
UC5_R_6
UC5_R_5
UC5_R_2
UC5_R_12
UC5_R_1
UC5_R_9
UC5_R_7
UC5_R_11
UC5_R_4
UC5_R_14
UC5_R_10
UC5_R_13
UC5_R_3
UC5_R_8


UC5_R_1_2
UC5_R_1_3
UC5_R_1_1

Using FRET

UC5_R_1: if sensorfaults & trackingPilotCommands Controller shall satisfy controlObjectives

UC5_R_1_1: when (diff_ref_obs > activeThreshold) if ((sensorValue_S > nominalValue + R) | (sensorValue_S < nominalValue - R) | (sensorValue_S = null) & (pilotInput => setThrust = V2) & (observedThrust = V1)) Engine shall until (diff_ref_obs < inactiveThreshold) satisfy (settlingTime >= 0) & (settlingTime <= settlingTimeMax) & (observedThrust = V2)

UC5_R_13: if trackingPilotCommands Controller shall satisfy newMode=nominal | newMode=surgeStallPrevention

UC5_R_13_2: in surgeStallPrevention mode when (diff_setNL_observedNL < NLmax) if (pilotInput => !surgeStallAvoidance) Controller shall until (diff_setNL_observedNL > NLmin) satisfy newMode=nominal

Aircraft Controller

Use case 5:
`https://repo.valu3s.eu/use-cases/aircraft-engine-controller`

- 1 Simulink diagram
- 14 FRETish parent requirements
- 28 FRETish child requirements
- 20 Test cases
- 4 Evaluation scenarios

Refactoring – MU-FRET

Refactoring Requirements

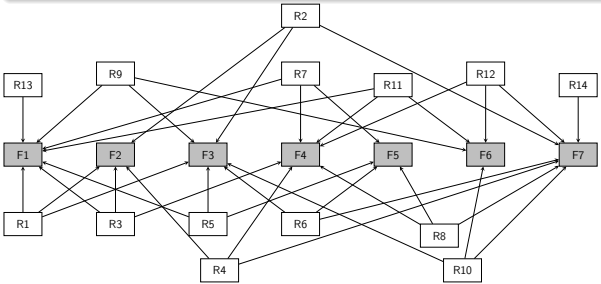
Analysis: Aircraft Engine Controller Requirements

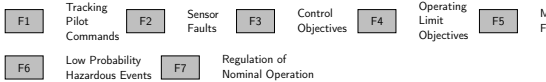
- Traceability: one-to-one mapping in FRETish

  `scope condition component` shall `timing response`

  UC5_R_1: `if((sensorFaults)&(trackingPilotCommands)) Controller` shall satisfy `(controlObjectives)`.

- Repetition of *fragments*.





Analysis: Aircraft Engine Controller Requirements

- 28 child requirements.

ID	FRETish
UC5_R_1.1	when (diff(r(i),y(i)) > E) if((sensorValue(S) > nominalValue + R) | (sensorValue(S) < nominalValue - R) | (sensorValue(S) = null) & (pilotInput => setThrust = V2) & (observedThrust = V1)) Controller shall until (diff(r(i),y(i)) < e) satisfy (settlingTime >= 0) & (settlingTime <= settlingTimeMax) & (observedThrust = V2)
UC5_R_1.2	when (diff(r(i),y(i)) > E) if((sensorValue(S) > nominalValue + R) | (sensorValue(S) < nominalValue - R) | (sensorValue(S) = null) & (pilotInput => setThrust = V2) & (observedThrust = V1)) Controller shall until (diff(r(i),y(i)) < e) satisfy (overshoot >= 0) & (overshoot <= overshootMax) & (observedThrust = V2)
UC5_R_1.3	when (diff(r(i),y(i)) > E) if((sensorValue(S) > nominalValue + R) | (sensorValue(S) < nominalValue - R) | (sensorValue(S) = null) & (pilotInput => setThrust = V2)& (observedThrust = V1)) Controller shall until (diff(r(i),y(i)) < e) satisfy (steadyStateError >= 0) & (steadyStateError <= steadyStateErrorMax) & (observedThrust = V2)

Summary

- Mu-FRET currently implements Extract, Rename and Inline Requirement
- Mu-FRET formally verifies that refactoring does not change the behaviour (using NuSMV)
- More refactorings adapted for FRETish:Move Definition (Ramos et al. Improving the Quality of Requirements with Refactoring. (SBQS 2007))
- Requirements set much easier to maintain
- Mu-FRET:

Mechanical Lung Ventilator

Mechanical Lung Ventilator: ABZ Case Study

Overview

- We describe a methodology that captures the requirements of the ABZ 2024 case study, the Mechanical Lung Ventilator, using the Formal Requirements Elicitation 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.

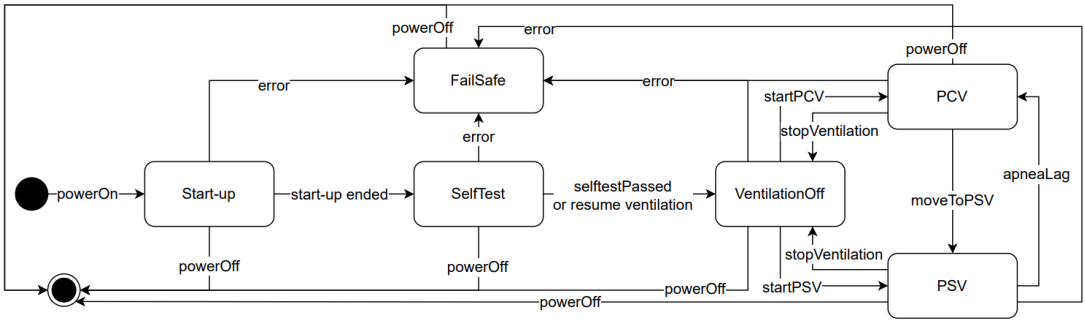


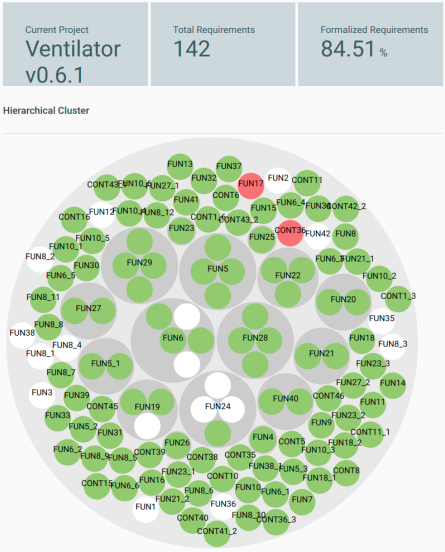
Figure 1: The controller state machine is labelled as Fig 4.1 in the case study documentation.

Formalisation of MLV with FRET

The Formal Requirements Elicitation Tool (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)

Update Requirement

Requirement ID: FUN25 | Parent Requirement ID | Project: Ventilator v0.6.1

Rationale and Comments

Requirement Description

A requirement follows the sentence structure displayed below, where fields are optional unless indicated with "*". For information on a field format, click on its corresponding bubble.

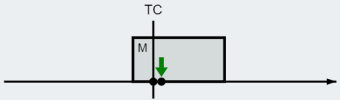
SCOPE | CONDITIONS | COMPONENT* | SHALL* | TIMING | RESPONSES*

in PSVMode when inspiratoryPressure < InhaleTriggerSensitivityPSV System shall at the next timepoint satisfy breathingCycleStart

SEMANTICS

ASSISTANT | TEMPLATES | GLOSSARY

ENFORCED: in every interval where ***PSVMode*** holds. TRIGGER: first point in the interval if ***(inspiratoryPressure < InhaleTriggerSensitivityPSV)*** is true and any point in the interval where ***(inspiratoryPressure < InhaleTriggerSensitivityPSV)*** becomes true (from false). REQUIRES: for every trigger, RES must hold at the next time step.



M = ***PSVMode***$_{\mathrm{in}}$, TC = ***(inspiratoryPressure < InhaleTriggerSensitivityPSV)***, Response = ***(breathingCycleStart)***.

Diagram Semantics

Formalizations

Future Time LTL

Past Time LTL

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 FRETish, 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 system 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.

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 the MLV 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

```
1   MACHINE mac00
2   SEES ctx00
3   VARIABLES mode
4   INVARIANTS  typeof__mode: mode ∈ Mode

5   EVENTS
6   Initialisation
7     then  act1: mode := PoweredOff

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

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

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

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

20    Event StartPCV ≙
21      when  grd0_1: mode = VentilationOff
22                  ∨  mode = PSV
23      then  act0_1: mode := PCV

24    Event StartPSV ≙
25      when  grd0_1: mode = VentilationOff
26                  ∨  mode = PCV
27      then  act0_1: mode := PSV

28    Event StopVentilation ≙
29      when  grd0_1: mode = PCV
30                  ∨  mode = PSV
31      then  act0_1: mode := VentilationOff

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

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

38    Event Error ≙
39      when  grd0_1: mode ≠ PoweredOff
40            grd0_2: mode ≠ Failsafe
41      then  act0_1: mode := Failsafe

42    Event PowerOff ≙
43      when  grd0_1: mode ≠ PoweredOff
44      then  act0_1: mode := PoweredOff
45  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

```
1  Event SelfTestPassedOrSkipped ≙
2    REFINES  SelfTestPassed
3    any timePoweredOff
4      when
5        grd0_1: mode = SelfTest
6        grd1_1: testPowerSwitch ∈ {TestPassed, TestSkipped}
7        grd1_2: testLeaks ∈ {TestPassed, TestSkipped}
8        grd1_3: testFF12 ∈ {TestPassed, TestSkipped}
9        grd1_4: testPS_EXP ∈ {TestPassed, TestSkipped}
10       grd1_5: testOxygenSensor ∈ {TestPassed, TestSkipped}
11       grd1_6: testAlarms ∈ {TestPassed, TestSkipped}
12       grd1_7: timePoweredOff ∈ ℤ
13       grd1_8: timePoweredOff ≤ 15 ∧ is_new_patient = FALSE
14     then
15       act0_1: mode := VentilationOff
16       act1_1: in_valve := ValveClosed
17       act1_2: out_valve := ValveOpen
18 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

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.

Verification in Event-B

- Expected many requirements to become machine invariants, but most basic requirements become machine functionality and are not formally verifiable properties of the machine.
- We were able to discharge all 79 proof obligations generated by Rodin automatically
- Wanted to capture functional and controller requirements, 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."*
- The FRET and Event-B artefacts are available at: `https://github.com/mariefarrell/abz2024`

FRET-Supported Toolchain

CoCoSim and CoPilot

CoCoSim: Contract based **Co**mpositional verification of **Sim**ulink models.

- FRET can generate CoCoSpec assume-guarantee contracts for Simulink blocks.
- CoCoSpec contracts are added to the Simulink diagram.
- Contracts are checked during simulations of the diagram using the Kind2 model checker.

CoPilot: Runtime monitoring framework.

LTL semantics: also used to generate runtime monitors for the implemented system.

FRET requirements without timing constraints: used to specify requirements in other formalisms e.g. Event-B, Dafny.

Using FRET with other Tools: Grasping Algorithm

ID	English-Language Description	FRET Formalisation
R1	The SV shall grasp the TGT at the BGP and draw it closer.	SV shall satisfy (grasp(TGT, BGP) & closer(SV, TGT))
R1.1	The Camera of the SV shall be positioned at least 0.5m from the TGT.	Camera shall satisfy distance(Camera, TGT) $\geq$ 0.5
R1.2	The TGT shall be motionless before contact with the SVA.	TGT shall satisfy if !contact(SVA, TGT) then motionless(TGT)
R1.3	The Camera shall return a valid point cloud.	Camera shall satisfy valid(p)
R1.3.1	The point cloud shall be structured with maximum resolution of 1280×720.	Camera shall satisfy maxRes(p) = 1280*720
R1.3.2	The point cloud shall not be empty.	Camera shall satisfy length(p) $>$ 0
R1.4	The imagepreprocessing shall return a filtered point cloud.	Imagepreprocessing shall satisfy length(filteredimage) $\leq$ length(p) & length(filteredimage) $>$ 0
R1.5	findoptimalgrasp shall return the optimal grasp point (BGP) if one exists.	Findoptimalgrasp shall satisfy if exists(BGP) then return(BGP)
R.1.5.1	The BGP shall be optimal according to the criteria: minimum offset from the TGT nozzle edge of 1cm and finger-surface yaw angle between -20 and 20 degrees.	Findoptimalgrasp shall satisfy offset(BGP, TGT) = 1 & -20 $\leq$ fingersurfaceyaw & fingersurfaceyaw $\leq$ 20
R1.5.2	findoptimalgrasp shall generate several candidate grasping points.	findoptimallgrasp shall satisfy length(grasps) $\geq$ 0
R1.6	If no BGP exists then findoptimalgrasp shall output an error message.	Findoptimalgrasp shall satisfy if !(exists(BGP)) then printerror
R1.7	Controller shall execute a joint trajectory to reach the BGP.	Controller shall satisfy executeJointTrajectory(SVA, BGP)
R1.8	The SVA shall capture the TGT at the BGP.	SVA shall satisfy captured(TGT) $\Rightarrow$ contactpoint(SVA, TGT) = BGP
R1.9	The total pulling distance shall be between 0.3 and 0.5m.	SV shall satisfy totalpullingdistance $\geq$ 0.3 & totalpullingdistance $\leq$ 0.5
R2	The SV shall not collide with the TGT.	SV shall always satisfy !collide(SV, TGT)
R2.1	The position of the SV shall not be equal to the position of the TGT.	SV shall always satisfy !(position(SV) = position(TGT))
R2.2	The SV shall only make contact with the TGT at the BGP using the SVG.	SV shall always satisfy contactpoint(SVG, TGT) = BGP.
R2.2.1	No part of the SV, other than the SVG shall make contact with the TGT.	SV shall satisfy if !grasped then contactpoint(SV, TGT) = null
R2.2.2	The SVG shall only make contact with the TGT at the BGP (within some margin of error).	SV shall satisfy if grasped then contactpoint(SVG,TGT) = BGP + errormargin
R2.3	The SVG shall apply a force of 180N once contact has been made with the TGT.	SVG shall satisfy captured(TGT) $\Rightarrow$ force = 180

Dafny for Program Verification

```
1  datatype Point = Point(x: real, y: real, z: real)
2  datatype Grasp = Grasp(x: real, y: real, z: real, qx: real, qy: real, qz: real, qw: real)
3  datatype Score = Score(areaScore: real, angleScore: real, index: int)
4
5  method imagepreprocessing(t: real, p: array<Point>, v: real, nb: int, rf: real)
6    returns (filteredimage: array<Point>)
7  requires 0 < p.Length; // R1.3.2
8  requires v > 0.0;
9  ensures filteredimage.Length ⩽ p.Length;//R1.4
10 ensures filteredimage.Length > 0 ; // R1.4
11 {
12  filteredimage := removeDepth(p,t); //remove distant points from p.
13  filteredimage := downSample(filteredimage,v); //make voxel representation of p.
14  filteredimage := filter(filteredimage,nb,rf); //removes noise and speckles from p.
15 }
```

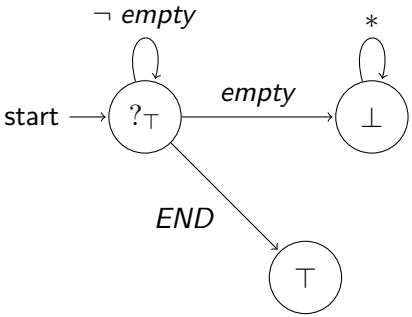
- Refactored the original Dafny model of the algorithm and provided more detailed helper functions[1].
- One of the helper methods became a challenge at VerifyThis 2022.

[1] Farrell, M., Mavrakis, N., Dixon, C., & Gao, Y. (2020). Formal Verification of an Autonomous Grasping Algorithm. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space*. European Space Agency.

Runtime Verification with ROSMonitoring

- Requirements 1.3.1, 1.3.2, 1.4, 1.8, 1.9, 2.1, 2.2 and 2.3 were verified through RV.

- Requirement R1.3.2: *The point cloud shall not be empty.*
- Past LTL: ■ ¬ *empty*
- Monitors were applied to the simulation and real (physical) test bed.



Gaps in the Requirements

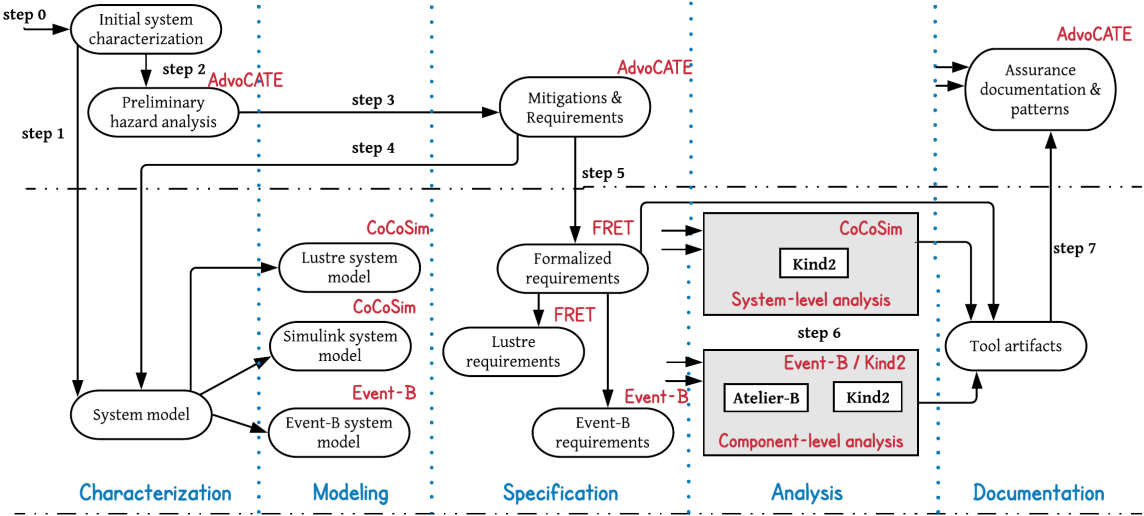
- Monitors helped to identify gaps in the requirements.

R1.9: *The total pulling distance shall be between 0.3 and 0.5 m.*

R2.3: *The SVG shall apply a force of 180N once contact has been made with the TGT.*

- Satisfied in simulation but not on the physical testbed.
- Cause: hardware limitations.

Formalising Requirements



Designing for Verification

Verification is a costly and time-consuming process but it can be mediated by good design practices:

- Detailed requirements formalisation and elicitation
- Modularity
- Isolate critical components
- Heterogeneous/corroborative verification using multiple techniques

Bourbouh, H., Farrell, M., Mavridou, A., Sljvio, I., Dennis, L.A., Fisher, M., Brat, G. Integrating Formal Verification and Assurance: An Inspection Rover Case Study. NASA Formal Methods Symposium, 2021.

Event-B Institution

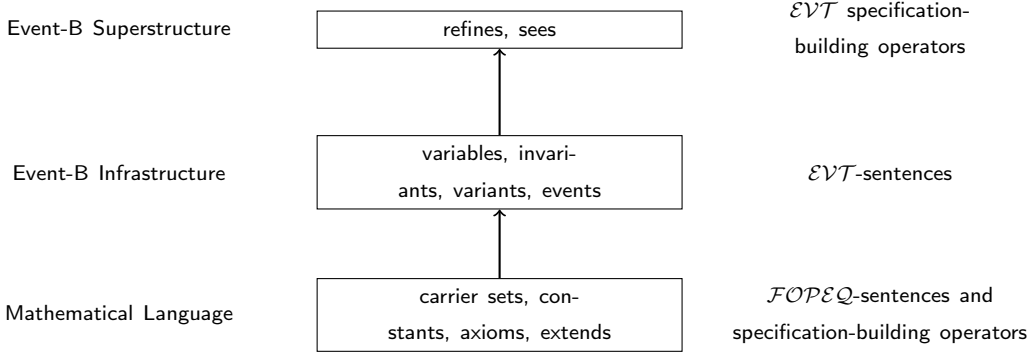
Presents a formalisation of the Event-B formal specification language, through providing an institution for Event-B that gives us:

- a mathematically sound semantics,
- modularisation constructs for Event-B, and
- interoperability with other formalisms.

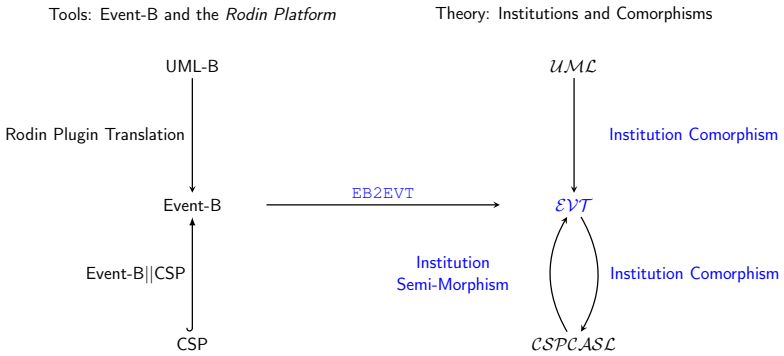
Funded by the Irish Research Council

[Tower of Babel, Pieter Bruegel the Elder, c. 1563]

- $\mathcal{EVT}$ - the institution for Event-B



A Framework for Interoperability



"Reasoning about logical systems in the Coq proof assistant." Science of Computer Programming, 2024

"Building Specifications in the Event-B Institution". Logical Methods in Computer Science, November 9, 2022

"Combining Event-B and CSP: An Institution Theoretic Approach to Interoperability". 19th International Conference on Formal Engineering Methods, 2017.

"Specification Clones: An empirical study of the structure of Event-B specifications". In: 15th International Conference on Software Engineering and Formal Methods, 2017.

Facilitating Paths to Heterogeneous Verification

- Traceability
- Modularity
- Refinement
- Institutions
- Unified Theory of Programming
- Information Exchange
  - VerifyThis
  - Intermediate Verification Languages
  - Contract-LIB
  - ...

Take Home

- Natural-Language Requirement Tracability
  - Difficult to express logically for formal methods.
  - Tools like FRET help.
  - Maintaining requirements is error-prone (repetition and dependencies).
- Our Experience: Aircraft Engine Controller, MLV, Grasping, Drones...
  - Formalise requirements in FRETish.
  - Build **traceability** from requirements to verification conditions.
  - Refactor to improve **maintainence** of requirements.
  - Provide **toolchains and information exchange** for verification.
  - **Heterogeneous/corroborative verification** is the way forward.
- Verification of AI based systems (specifications, learning etc)
- Integration of tools, theories

Publications

- Sheridan, O., Becker, L.B., Farrell, M., Luckcuck, M., Monahan, R. Sharper Specs for Smarter Drones: Formalising Requirements with FRET. *In: Hess, A., Susi, A. (eds) Requirements Engineering: Foundation for Software Quality. REFSQ 2025*
- Farrell, M., Luckcuck, M., Sheridan, O., Monahan, R. Adventures in FRET and Specification *In: Bridging gaps between program specification paradigms (Specify This), ISoLA 2024*
- Farrell, M., Luckcuck, M., Sheridan, O., Monahan, R. FRETting and Formal Modelling: A Mechanical Lung Ventilator *Rigorous State-Based Methods. ABZ 2024*
- Luckcuck, M., Farrell, M., Sheridan., Why just FRET when you can Refactor? Retuning FRETish Requirements. *arXiv preprint arXiv:2202.05816.* 2022
- Luckcuck, M., Farrell, M., Sheridan, O. A Methodology for Developing a Verifiable Aircraft Engine Controller from Formal Requirement, *IEEE Aerospace Conference 2022*.
- Luckcuck, M., Farrell, M., Sheridan, O. Towards Refactoring FRETish Requirements *NASA Formal Methods 2022.*
- Farrell, M., Luckcuck, M., Sheridan, O., Monahan, R. FRETting about Requirements: Formalised Requirements for an Aircraft Engine Controller. *International Conference on Requirements Engineering: Foundation for Software Quality 2022.*
- Bukhari, SAA., Flinkow, T., Inkarbekov, M., Pearlmutter, BA., Monahan, R. Creating a Formally Verified Neural Network for Autonomous Navigation: An Experience Report. *Formal Methods for Autonomous Systems 2024.*
- Flinkow, T., Pearlmutter, BA., Monahan, R. Comparing Differentiable Logics for Learning with Logical Constraints *Formal Methods for Autonomous Systems 2023 and Sci of Comp Prog (under review)*
- Flinkow, T., Pearlmutter, BA., Monahan, R. Immersive Neural Network Exploration: A VR Approach to Human-Centered AI Understanding. *Conference on Human Centered Artificial Intelligence (HCAI-ep) 2023.*

Acknowledgements

Acknowledgements

- Marie Farrell and Conor Reynolds, University of Manchester
- Matt Luckcuck, University of Nottingham
- Oisin Sheridan, Medet Inkarbekov, Ali Bukhari, Thomas Flinkow, Barak Pearlmutter - Maynooth University
- Georgios Giantamidis, Stylianos Basagiannis and Vassilios A. Tsachouridis - United Technologies Research Center, Ireland
- Anastasia Mavridou - KBR/NASA Ames Research Center, USA
- Funding:
  - Science Foundation Ireland SFI/20/FFP-P/8853 and ADAPT FI
  - VALU3S project (grant No 876852)
  - Enterprise Ireland (grant No IR20200054)
  - MU Hume Scholarship

Funding

Funding

Funding:

Further Information and Resources: http://www.cs.nuim.ie/research/pop/