Sharper Specs for Smarter Drones: Formalising Requirements with FRET

Oisín Sheridan Leandro Buss Becker Marie Farrell Matt Luckcuck Rosemary Monahan

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

Automation and Systems Department, Federal University of Santa Catarina, Florianópolis, Brazil

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

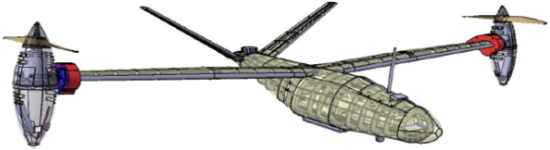
School of Computer Science, University of Nottingham, Nottingham, UK

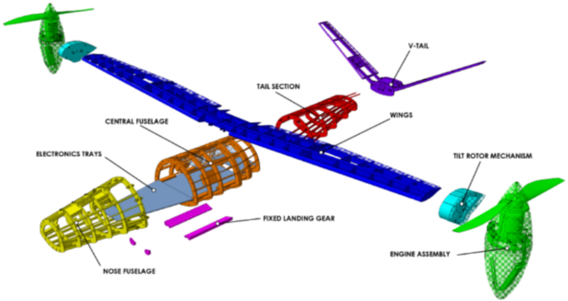
Introduction

Overview

- We describe the process of formalising the natural-language requirements for the ProVANT Emergentia tilt-rotor drone using the Formal Requirements Elicitation Tool (FRET) and FRETish.
- This requirements set evolved over four distinct versions as new information was elicited and incorporated into the FRETish specification.
- Our two concrete outputs are the formalised requirement set, which we will use in our ongoing development and verification of ProVANT; and metrics about the requirements.
- We present guidance for requirements elicitation and formalisation with FRET.
  - Highlight where it was hard to formalise the requirements
  - Describe potential improvements to FRET to address these difficulties.

Case Study – ProVANT Emergentia Tilt-Rotor Drone





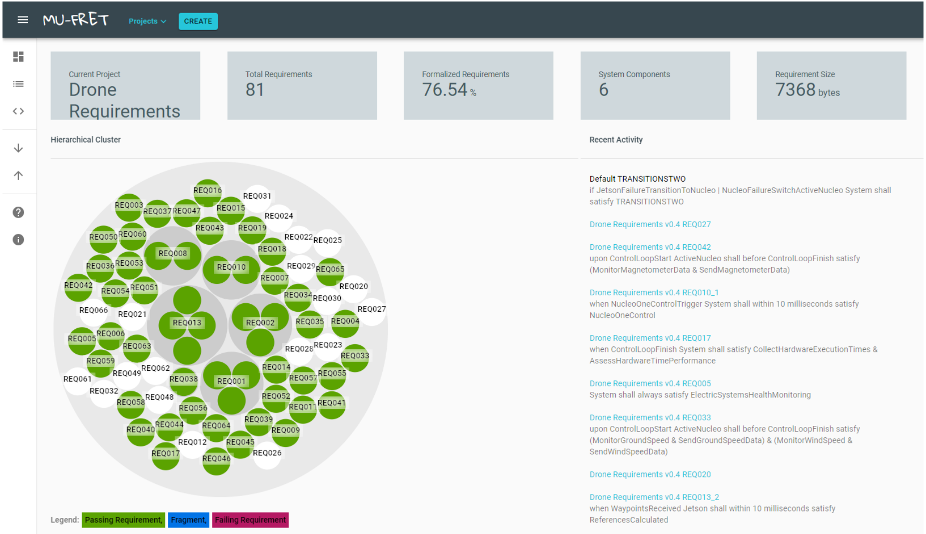
- At the latter stage of development under the ProVANT Emergantia project
- Collaboration between Federal University of Minas Gerais and Federal University of Santa Catarina (Brazil), and University of Seville (Spain).
  - One author of this work is a member of the ProVANT project

Case Study – Tilt-Rotor Drone

- The drone can perform hovering and Vertical Take-off and Landing (VTOL) manoeuvres, as well as cruise flight as a fixed-wing aircraft.
- The architecture for the Drone's computing system comprises four main components:
  - Raspberry Pi: Gathers sensor data and communicates with the Ground Control Station.
  - Jetson: Processes sensor data and runs the control algorithm.
  - Nucleos: the active nucleo interfaces with the drone's actuators and some sensors. Can also run a backup control algorithm in the case of a failure. There are two nucleos for reliability.
- The set of requirements for the ProVANT Emergentia drone includes aspects related to:
  1. operation features present during simulations and during real executions
  2. remote monitoring configurations
  3. timing constraints associated with the control loop
  4. operation modes under failure conditions

Formalisation with FRET

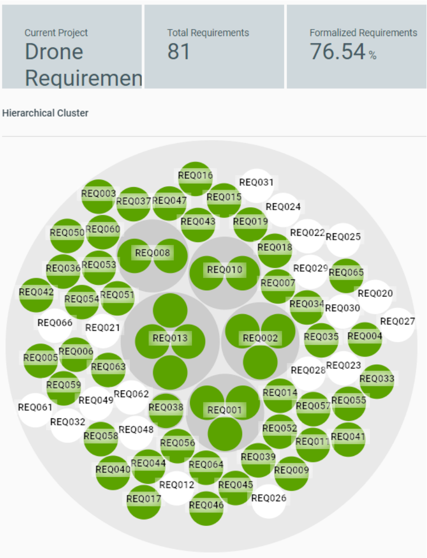
The Formal Requirements Elicitation Tool (FRET)



The Formal Requirements Elicitation Tool (FRET)

FRET

- An open-source requirements engineering tool developed by NASA
- Requirements are written in a structured natural-language called FRETish
- FRET automatically translates FRETISH into temporal logic
  - Unambiguous semantics
  - Enables requirement verification
- Formalised requirements are indicated in green, while those in white have not been formalised



Update Requirement

Requirement ID
REQ033

Parent Requirement ID

Project
Drone Requirements v4

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.

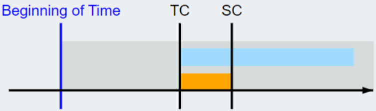
SCOPE CONDITIONS COMPONENT* SHALL* TIMING RESPONSES*

upon ControlLoopStart ActiveNucleo shall before ControlLoopFinish satisfy (MonitorGroundSpeed & SendGroundSpeedData) & (MonitorWindSpeed & SendWindSpeedData)

SEMANTICS

ASSISTANT TEMPLATES GLOSSARY

ENFORCED: in the interval defined by the entire execution. TRIGGER: first point in the interval if ***(ControlLoopStart)*** is true and any point in the interval where ***(ControlLoopStart)*** becomes true (from false). REQUIRES: for every trigger, RES must hold at least once strictly before the state where the stop condition holds. If the stop condition never occurs in the interval, RES does not need to hold. If the stop condition holds at the trigger, the requirement is not satisfied.



TC = ***(ControlLoopStart)***, SC = ***(ControlLoopFinish)***, Response = ***((MonitorGroundSpeed & SendGroundSpeedData) & (MonitorWindSpeed & SendWindSpeedData))***.

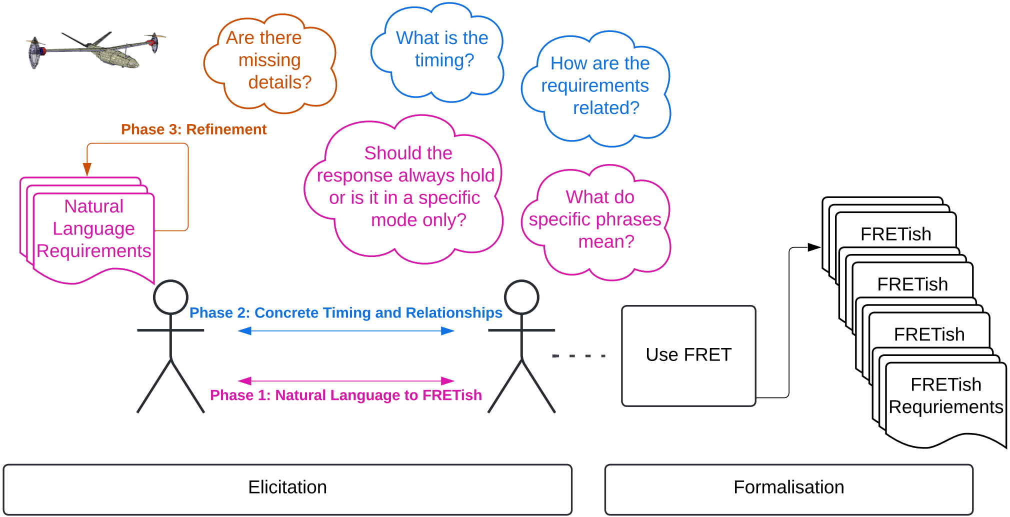
Diagram Semantics

Formalizations

Future Time LTL

Formalising the Requirements

Formalising the Requirements – Methodology



Formalising the Requirements – Starting from Natural Language

ID	Original English text
REQ001	Allow failure simulations between nucleo/jetson and nucleo/nucleo
REQ013	Send references
REQ016	Run each simulation loop within 10ms
REQ018	Present the total time spent
REQ019	Present the time spent in the control algorithm
REQ024	Work on any operational system
REQ033	Monitor linear velocities (ground speed and relative wind speed)

- The formalisation began with 66 natural-language requirements for the tilt-rotor drone.
- Each requirement has an ID number, short description, and metadata:
  - *Category* (Functional/Non-Functional)
  - *Feasibility* (Feasible/Unknown/Unfeasible)
  - *Group* (e.g. Data Monitoring, Failure Analysis, etc)

Formalising the Requirements – FRETish Version 1

- For the first iteration, we mapped the drone's original natural-language requirements one-to-one into FRETISH, matching the vocabulary between both versions where possible.
- Many of these were of a simple form, such as "`System` shall `always/eventually satisfy [variableName]`"
- Twenty of the 66 requirements were not translated in this initial set
- This stage gave us a clearer picture of what information we would need for a more robust formalisation

ID	FRETish First Iteration
REQ001	`System` shall `always satisfy AllowNucleoJetsonSimulation & AllowNucleoNucleoSimulation`
REQ016	`when SimulationLoopStart System` shall `within 10 milliseconds satisfy SimulationLoopFinish`
REQ033	`System` shall `always satisfy MonitorGroundSpeed & MonitorWindSpeed`

Formalising the Requirements – FRETish Version 2

- We discussed the ambiguities that we found and consulted with the use case provider for additional detail, leading to a number of updates.
- The largest update was a distinction between running the system in simulation versus in the real world. We created a `SimulationMode` scope variable in nine requirements.
- 27 requirements gained a scope of "`while MonitoringEnabled`", so that Data Monitoring would be optional when running the system.
- We added the first child requirements to the set: REQ008_1 & _2, and REQ010_1 & _2.

ID	FRETish Second Iteration
REQ001	`in SimulationMode System shall eventually satisfy NucleoJetsonFailure|NucleoNucleoFailure`
REQ033	`while MonitoringEnabled System shall always satisfy MonitorGroundSpeed & MonitorWindSpeed`

Formalising the Requirements – FRETish Version 2

Child requirements

We use child requirements to express how a requirement should apply to different components or in different situations.
REQ008_1 specifies that the Raspberry Pi should transmit the data to the ground station, while REQ008_2 states that the Jetson should save the data and send it to the active Nucleo for evaluation

ID	Final FRETish text
REQ008	Save any desired simulation data
	`after SimulationMode System shall within 100 ticks satisfy SimulationDataSaved`
REQ008_1	`after SimulationMode Raspberry shall within 100 ticks satisfy GroundStationReceivedData`
REQ008_2	`after SimulationMode Jetson shall within 100 ticks satisfy SimulationDataRecorded & NucleoReceivedData`

Formalising the Requirements – FRETish Version 3

- We found that the "`in SimulationMode`" scope didn't fully capture the intention of testing a specific response, so we added `Conditions` for this.
- We introduced 9 new child requirements to specify additional behaviour of some requirements.

ID	FRETish Third Iteration
REQ001	`in SimulationMode whenever SimulateFailureTransitions System` shall `eventually satisfy JetsonFailureTransitionToNucleo | NucleoFailureSwitchActiveNucleo`
REQ001_1	`when JetsonControl & JetsonFailureTransitionToNucleoFailure System` shall `within 100 ticks satisfy !JetsonControl & NucleoControl`
REQ001_2	`when NucleoOneControl & NucleoFailureSwitchActiveNucleo System` shall `within 100 ticks satisfy !NucleoOneControl & NucleoTwoControl`

Formalising the Requirements – FRETish Version 3

- The biggest change in this iteration: we decided to update two of the natural-language requirements - REQ018 and REQ019 - to capture new information.
- Elicitation discussions highlighted details about timing of control loop and control algorithm, which we added to the FRETISH requirements
- Having this information captured in the FRETISH was useful when developing the fourth iteration of the requirements.

Iteration	Natural-language and FRETish for REQ018
Version 1	Present the total time spent
	`System` shall `always satisfy DisplayTotalTimeSpent`
Version 3	The control loop will complete within 12 milliseconds
	`upon ControlLoopStart System` shall `within 12 milliseconds satisfy ControlLoopFinish`

Formalising the Requirements – FRETish Version 4

- This final iteration focused on cleaning up issues that arose during elicitation discussions and re-evaluation of the overall progress made up to that point.
- We returned to the Data Monitoring requirements and found that the idea of the system being run with or without monitoring was incorrect; the system should always monitor these values and transmit the data back to the GCS.
- We used the previous updates to REQ018 to update 23 monitoring requirements from a simple `always` timing to a more detailed structure.

REQ060	Monitor current consumption in each voltage bus
FRETISH v2	`while MonitoringEnabled System` shall `always satisfy MonitorVoltageBusConsumption`
FRETISH v4	`upon ControlLoopStart ActiveNucleo` shall `before ControlLoopFinish satisfy MonitorVoltageBusConsumption & SendVoltageBusConsumptionData`

Formalising the Requirements – Beginning and End

ID	Final FRETish text
REQ001	Allow failure simulations between nucleo/jetson and nucleo/nucleo
	`in SimulationMode whenever SimulateFailureTransitions System shall eventually satisfy JetsonFailureTransitionToNucleo | NucleoFailureSwitchActiveNucleo`
REQ018	The control loop will complete within 12 milliseconds
	`upon ControlLoopStart System shall within 12 milliseconds satisfy ControlLoopFinish`
REQ019	The control algorithm will complete within 6 milliseconds
	`upon ControlAlgorithmStart System shall within 6 milliseconds satisfy ControlAlgorithmFinish`
REQ033	Monitor linear velocities (ground speed and relative wind speed)
	`upon ControlLoopStart ActiveNucleo shall before ControlLoopFinish satisfy (MonitorGroundSpeed & SendGroundSpeedData) & (MonitorWindSpeed & SendWindSpeedData)`

Analysis & Discussion

scope-option	null = 47, in/during = 6, while = 5, after = 4
condition-option	null = 17, trigger(`when/if`) = 39, continual(`whenever`) = 6
timing-option	null/eventually = 4, always = 15, next = 1, within = 18, before = 24
parent-child	28 child requirements were assigned a parent requirement
66 natural-language requirements, of which 47 are expressed in FRETISH. An additional 15 child requirements were created, for a total of 81 requirements in FRET.	

Requirement Metrics for Final Version

- We counted which keywords used for FRETISH `scope`, `condition`, and `timing` fields
- The `scope` field was not often used, as the natural-language requirements did not specify any system modes. `scope` was mostly used for the `SimulationMode`.
- Conversely, almost every requirement in the final requirement set has a defined `timing`, with the few that don't being unchanged from earlier versions of the requirements set.

Analysis & Discussion

Recommendations for Formalising Requirements

- Requirements elicitation and formalisation is best performed as an incremental process, where all parties involved regularly re-examine the requirements in the context of newly-elicited details and newly-uncovered questions.
- We found it very useful to maintain a system where distinct "versions" of the requirements set were created and then analysed, rather than a more continuous development process.
- We encourage requirements engineers to maintain detailed records of prior versions of requirements and the updates made to them, to inform discussions on future development as well as for traceability.

Analysis & Discussion

Improvements to FRET and Other Tools

- At the time, tracking multiple iterations of a FRETISH requirements set was difficult, as the tool did not directly support renaming or cloning projects. The development team have since added cloning functionality.
- Parameterised requirements - which would allow the user to apply a single requirement structure to a number of different variables - would have reduced duplication for the 23 Data Monitoring requirements.
- FRET currently supports adding comments and rationale to requirements, but there is no way to add comments to a project as a whole. This would be useful to precisely define the meanings of variables and reduce reliance on external notes.

Sharper Specs for Smarter Drones: Formalising Requirements with FRET

Conclusion

- We have formalised the requirements for a tilt-rotor UAV drone using FRET.
- These requirements can now be used for runtime verification of the system code.
- From this experience we have compiled recommendations for requirements engineers and tool developers.
- All of our requirements, collected both in spreadsheets and in FRET-compatible JSONs, are available on GitHub at `https://github.com/oisinsheridan/refsq2025`.