Sharper Specs for Smarter Drones: Formalising Requirements with FRET

Oisín Sheridan[1] Leandro Buss Becker[2] Marie Farrell[3] Matt Luckcuck[4] Rosemary Monahan[1]

[1]Department of Computer Science, Maynooth University/Hamilton Institute, Maynooth, Ireland

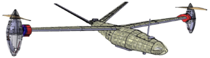
[2]Automation and Systems Department, Federal University of Santa Catarina, Florianópolis, Brazil

[3]Department of Computer Science, The University of Manchester, Manchester, UK

[4]School of Computer Science, University of Nottingham, Nottingham, UK

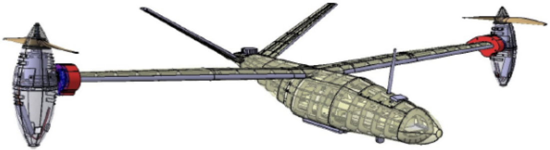
Overview

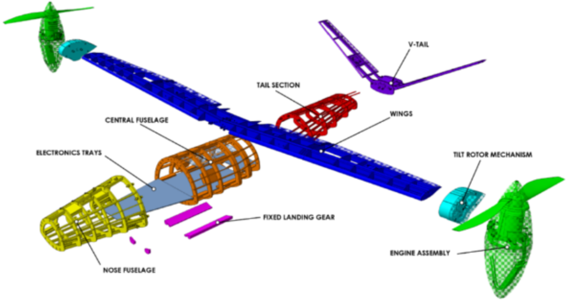
- Formalise natural-language requirements for the ProVANT Emergentia tilt-rotor autonomous drone using FRET.



- Present 4 distinct iterations of requirements set.
- Provide guidance for requirements elicitation and formalisation with FRET.

Case Study: ProVANT Emergentia Tilt-Rotor Autonomous Drone



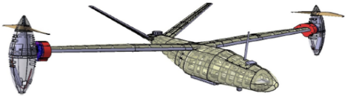


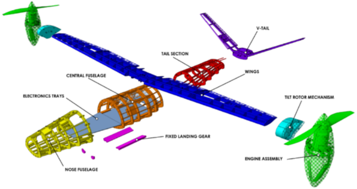
- Collaboration between Federal University of Minas Gerais and Federal University of Santa Catarina (Brazil), and University of Seville (Spain).

Case Study: ProVANT Emergentia Tilt-Rotor Autonomous Drone

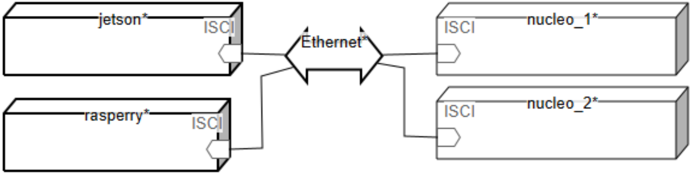
- Performs hovering and Vertical Take-off and Landing (VTOL) manoeuvres, as well as cruise flight as a fixed-wing aircraft.

- Requirements include 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





Case Study: Architecture



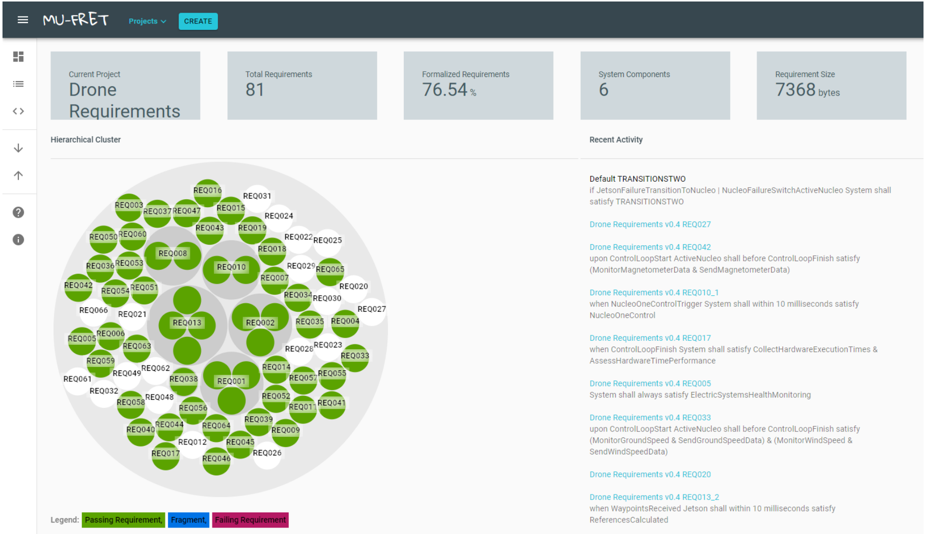
Raspberry Pi: Gathers sensor data and communicates with the Ground Control Station.

Jetson: Processes sensor data and runs the control algorithm.

Nucleos: Active nucleo interfaces with the drone's actuators and some sensors.
Can run a backup control algorithm in the case of a failure.
There are two nucleos for reliability.

Formalisation with FRET

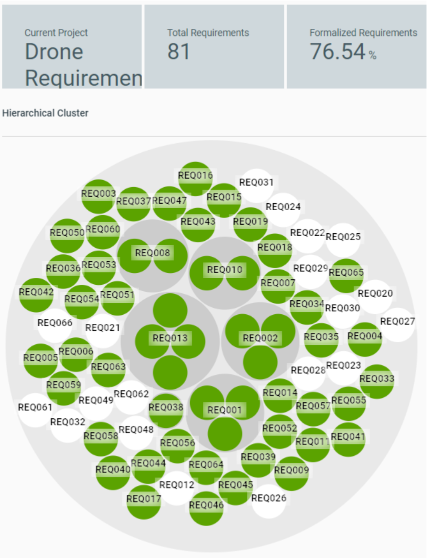
The Formal Requirements Elicitation Tool (FRET)



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.

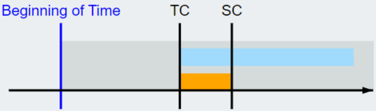
SCOPE CONDITIONS COMPONENT* SHALL* TIMING RESPONSES*

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.



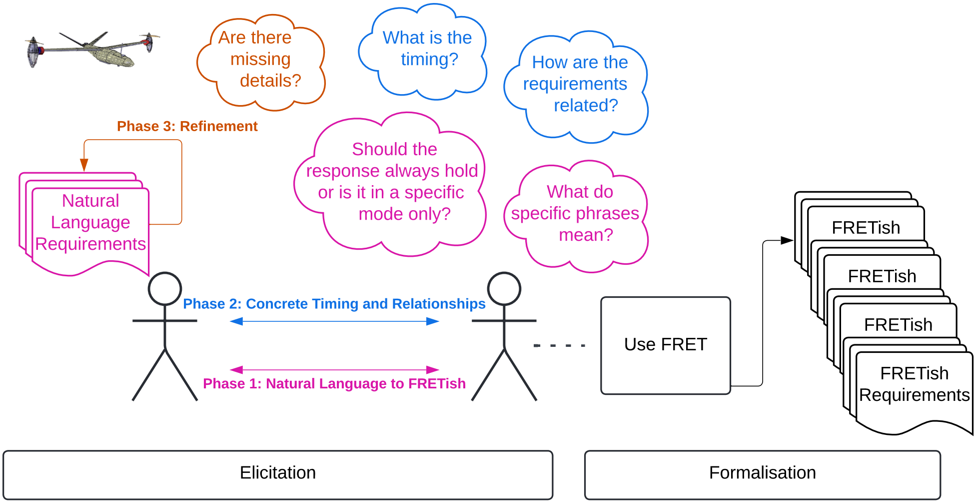
TC = ***(ControlLoopStart)***, SC = ***(ControlLoopFinish)***, Response = ***((MonitorGroundSpeed & SendGroundSpeedData) & (MonitorWindSpeed & SendWindSpeedData))***.

Diagram Semantics

Formalizations

Future Time LTL

Formalising the Requirements


Are there missing details?
What is the timing?
How are the requirements related?
Phase 3: Refinement
Should the response always hold or is it in a specific mode only?
What do specific phrases mean?
Natural Language Requirements
Phase 2: Concrete Timing and Relationships
Phase 1: Natural Language to FRETish
Use FRET
FRETish
FRETish
FRETish
FRETish Requirements
Elicitation
Formalisation

From Natural-Language to FRETISH

ID	Natural-Language Requirement
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)

- 66 natural-language requirements.
- 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)

The First Iteration – FRETISH Version 1

- One-to-one mapping from natural-language to FRETISH.
- Many had a simple form: "System shall always/eventually satisfy [variableName]"
- 20 of the 66 requirements were not translated in this initial set.

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

The Second Iteration – 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 9 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`

The Second Iteration – FRETISH Version 2

Child requirements

- Child requirements express how a requirement should apply to different components or in different situations.
- REQ008_1: Raspberry Pi should transmit the data to the ground station and REQ008_2: Jetson should save the data and send it to the active Nucleo for evaluation

ID	FRETISH **Second Iteration**
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`

The Third Iteration – 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.
- 9 new child requirements specify additional behaviour.

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`

The Third Iteration – 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

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`

The Fourth Iteration – FRETISH Version 4

- 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`

FRETISH Requirements

ID	Final FRETISH Requirements
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

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.

Recommendations for Formalising Requirements

1. 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.

2. Useful to maintain a system where distinct "versions" of the requirements set were created and then analysed, rather than a more continuous development process.

3. 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.

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`.

Robotics: A New Mission for FRET Requirements

Gricel Vázquez[1] Anastasia Mavridou[2] Marie Farrell[3] Tom Pressburger[4]
Radu Calinescu[1]

[1]Department of Computer Science, University of York, York, UK

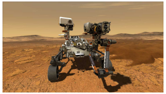
[2]KBR Inc., NASA Ames Research Center, Moffett Field, USA

[3]Department of Computer Science, The University of Manchester, Manchester, UK

[4]NASA Ames Research Center, Moffett Field, USA

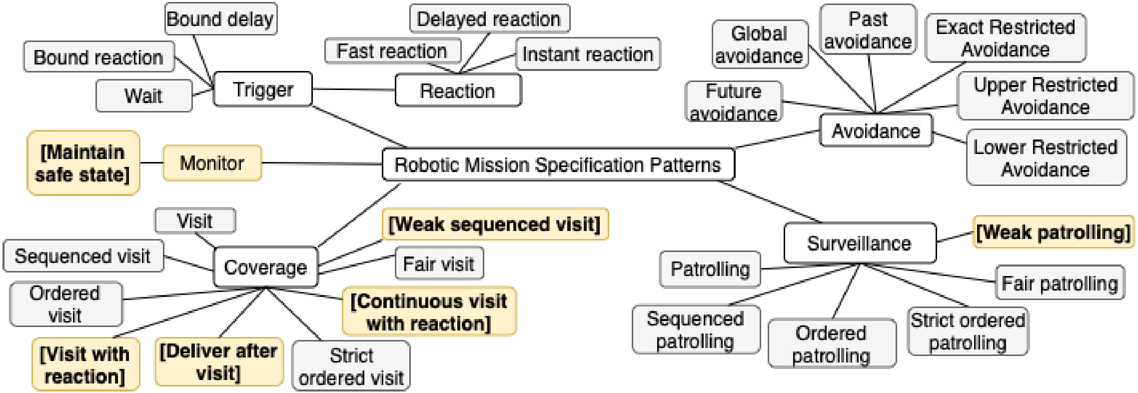
Overview

- A set of 6 newly identified robotic mission specification patterns that were derived from a systematic literature review.



- The specification using FRET of both previously identified patterns and our newly identified patterns.
- A study of the expressibility and applicability of FRET for robotic missions.

Robotic Mission Patterns



Questions?

iFM 2025

17-21 November 2025, Paris, France

The 20th International Conference on Integrated Formal Methods (iFM 2025) will take place on 19–21 November 2025 at Inria Paris, France, with co-located workshops scheduled for 17-18 November 2025.

Important Dates (AoE)

Abstract Submission	~~30 May 2025~~ **13 June 2025**
Paper Submission	~~6 June 2025~~ **20 June 2025**
Author Notification	08 Aug 2025
Artifact Registration	15 Aug 2025
Artifact Submission	22 Aug 2025
Artifact Notification	24 Sep 2025
Camera-Ready Papers	26 Sep 2025

FMAS

7th International Workshop on Formal Methods for Autonomous Systems

- **Submission:** 22nd of August 2025 (AOE)
- **Notification:** 6th of October 2025
- **Workshop:** 17th – 19th November 2025

@iFM 2025, Paris, France, 17th – 19th Nov 2025

Topics related to autonomous systems:

1. Applicable, tool-supported Formal Methods;
2. Runtime Verification and other approaches to bridge the reality gap;
3. Verification against safety assurance arguments or standards;
4. Formal specification and requirements engineering for autonomous systems;
5. Case Studies on challenges in applying formal methods to autonomous systems;
6. Experience Reports with guidance on using formal methods or tools;
7. Discussions on future directions of the field.

Proceedings published in EPTCS: Vision papers (6 pages), Research Previews (6 pages), Experience Reports (15 pages), Regular papers (15 pages)